StatLib---Applied Statistics algorithms

The Royal Statistical Society has been publishing algorithms in its journal Applied Statistics since 1968. As of 1989, there are over 250 of them. Most are in Fortran, though a few were in Algol, and some recent ones have been in Pascal. The book - Applied Statistics Algorithms by Griffiths, P. and Hill, I.D., Ellis Horwood: Chichester (1985) contains translations of several algorithms from Algol to Fortran. A few of the other algorithms have been supplied in Fortran translations, though a few are in Algol or Pascal. The index which follows indicates which algorithms are not in Fortran.

The full source code is published in the journal. Those available here have been transcribed manually, mainly within CSIRO Division of Mathematics & Statistics, or have been supplied directly by the authors or by the RSS Algorithms Editor. In some cases, later corrections or improvements published in Applied Statistics, have been incorporated.

It is the policy of the editors of the algorithms section of Applied Statistics that algorithms do not use double precision. Many of the algorithms here have been converted to double precision, though users should be careful to check what precision is used. Many of the algorithms require the use of other algorithms, particularly functions for the gamma function and the normal distribution function. Where such functions or subroutines are required, an appropriate comment has been added to the algorithm.

In a few cases, alternative algorithms from other sources have been added. For instance, three algorithms are included in the file for as66 for calculating the area under the normal curve, and an alternative random number generator is provided in the file for as183. The Applied Statistics algorithm for Nelder-Mead simplex minimization (AS 47) does not include the fitting of a quadratic surface; the CSIRO/ Rothamsted implementation which does this is included with AS 47.

Users must consult the original journal articles for details of the calling arguments; they are not included with these algorithms.

*** Warning. In some cases, there are different arguments (usually more) in the Griffiths & Hill versions of these algorithms. Users should check this.

It has been assumed that the user will be using a Fortran-77 compiler, so functions such as alog and amin1 have been converted to their generic forms (log and min). This simplifies conversion of code between single and double precision. Also all constants in the code, such as 1.0, 0.d0, etc., have been replaced with one, zero, etc., and these are defined in either data or parameter statements. Many of the algorithms have been entered in lower case, which is not acceptable in Fortran, though most compilers accept it.

Some of the algorithms need machine-dependant constants. The user should check this. In most cases, these have been set for compilers which allow a range of floating-point numbers from about 10**(-37) to 10**(+37), though most modern compilers allow a much wider range

in double precision.

No guarantee is given that the algorithms have been entered correctly, or that they perform as claimed in the journal.

To obtain an algorithm, send an E-mail request of the form: send index from apstat send 207 from apstat to statlib@lib.stat.cmu.edu

The full collection of Applied Statistics algorithms is very large. Please only request those algorithms which you need. Requesting large numbers of algorithms places a great strain on the StatLib system and the underlying mail networks.

As of the end of 1997 the Applied Statistics journal does NOT accept algorithms

---

Listing of available Applied Statistics algorithms

No.

Brief description (volume number/year in brackets)

3

Student's t-distribution. (17/1968) See also AS 27.

5

The non-central t-distribution. (17/1968) See also AS 243.

6

Cholesky decomposition of a symmetric +ve definite matrix. (17/1968)

7

Inversion of a symmetric matrix stored in triangular form. (17/1968)

13

Minimum spanning tree. (18/1969) See also AS 40.

14

Printing the minimum spanning tree. (18/1969)

15

Single-linkage cluster analysis. (18/1969)

22

Calculate treatment effects in a complete factorial experiment for any numbers of levels of factors using the extended Yate's method. (19/1970)

27

Upper tail area under Student's t-distribution. (19/1970) See also AS 3.

30

Half-normal plotting. (19/1970)

32

The incomplete gamma integral. (19/1970) See also AS 239.

34

Update inverse of symmetric banded matrix. (19/1970)

38

Calculate R-squared for all possible regression subsets using the Gauss-Jordan method. (20/1971) See also AS 268.

40

Update a minimum spanning tree. (20/1971)

41

Updates corrected sums of squares and products matrices. (20/1971) See also AS 240.

45

Histogram plotting. (20/1971)

46

Gram-Schmidt orthogonalization. (20/1971)

47

Nelder & Mead simplex method of unconstrained minimization without requiring derivatives. Does not include the quadratic surface fitting part of the algorithm. A CSIRO/Rothamsted version of the algorithm, which does include fitting a quadratic surface, is also included. (20/1971) (Updated, 20/Dec/93)

51

Log-linear fit for contingency tables. (21/1972) See also AS 160.

52

Calculation of sums of powers (up to 4) of deviations from the mean. (21/1972)

53

Wishart variate generator. (21/1972)

57

Printing multi-dimensional tables. (22/1973)

58

Allocates observations to clusters to minimize within-cluster sum of squares. (22/1973) See also AS 136.

60

Eigenvalues/vectors of a symmetric matrix. (22/1973)

62

Distribution of the Mann-Whitney U statistic. (22/1973)

63

Incomplete beta function. (22/1973) See also TOMS algorithm 708. TOMS algorithms are available from netlib.

64

Inverse of the incomplete beta function ratio. (22/1973) The file here is actually the Griffiths & Hill version of AS 109.

65

Expands structure formula to a list of binary integers. This is actually remark R82 which replaces the original AS65. (39/1990)

66

The normal distribution function. Two other algorithms (not from Applied Statistics) have also been included. (22/1973)

75

Algorithms for least-squares calculation using square-root free planar rotations (Morven Gentleman's package). (23/1974)

76

An integral useful in calculating noncentral t and bivariate normal probabilities. (23/1974)

77

Calculate exact null distribution of the largest root of a beta matrix. (23/1974)

78

The mediancentre (i.e. the median in a multi-dimensional space). (23/1974) See also AS 143.

83

Complex discrete fast Fourier transform. (24/1975)

84

Measures of multivariate skewness and kurtosis. (24/1975)

88

Generate all nCr combinations by simulating nested Fortran DO-loops. (Jane Gentleman's routines). (24/1975) See also AS 172.

89

Tail probabilities for Spearman's rho. (24/1975)

91

Percentage points of the chi-squared distribution. (24/1975)

93

Calculates frequency distribution for the Ansari-Bradley test statistic. (25/1976) A routine has been added to return the distribution function.

95

Maximum likelihood estimation of scale and location parameters from grouped data. User's distribution function. (25/1976)

96

Finding `nice' scales for graphs. (25/1976)

97

Real discrete fast Fourier transform. Series length must be a power of 2. (25/1976) See also AS 117 and AS 176.

99

Fitting Johnson curves by moments. (25/1976)

100

Normal-Johnson and Johnson-Normal transformations. (25/1976)

103

Psi or digamma function. (25/1976)

107

Calculate operating characteristics and average sampling number for a general class of sequential sampling plans. (26/1977)

108

Multiple linear regression minimizing the sum of absolute errors. (26/1977) See also AS 238 (in Pascal).

109

Inverse of the incomplete beta function. (26/1977)

110

LP-Norm fit of straight line by extension of Schlossmacher. (26/1977)

111

Percentage points of the normal distribution. (26/1977) See also AS 241.

114

Compute the numerator of certain ordinal measures of association (Kendall's tau, Somer's d, Goodman and Kruskal's gamma) when the data are ordered categories. (26/1977)

116

Calculate the tetrachoric correlation and its standard errors. (26/1977)

117

Fast Fourier transform for series of any length using the CHIRP algorithm. (26/1977)

121

Trigamma function. (27/1978)

123

Distribution function of mixtures of beta distributions. (27/1978)

125

Maximum likelihood estimation for censored exponential survival data with covariates. (27/1978)

126

Distribution function of the range for the normal distribution. (27/1978)

127

Generation of random orthogonal matrices. (27/1978)

128

Computes approximate covariance matrix for normal order statistics. (27/1978)

132

Simple regression minimizing the sum of absolute deviations. (27/1978)

133

Finding the global maximum or minimum of a function of 1 variable. (27/1978)

134

Generate random beta variates for alpha < 1 and beta > 1. (28/1979)

135

Min-Max (L-infinity) estimates for linear multiple regression. (28/1979)

136

A K-means clustering algorithm. (28/1979)

138

Maximum likelihood estimates of the mean and standard deviation of the normal distribution with censored or confined observations. (28/1979)

139

Maximum likelihood estimation in a linear model from confined and censored normal data. (28/1979)

140

Clustering the nodes of a directed graph. (28/1979)

141

Inversion of a symmetric matrix ignoring a specified row/column. (28/1979)

142

Exact tests of significance in binary regression. (28/1979)

143

Calculates the median centre. (28/1979)

145

Exact distribution of the largest multinomial frequency. (28/1979)

147

Incomplete gamma function. (29/1980) See also AS 239.

148

Removal of bias in the jackknife procedure. (This is actually ASR 62) (29/1980)

149

Amalgamation of means in the case of simple ordering ('Up-and-Down Blocks' algorithm for isotonic regression). (29/1980)

150

Computes estimate of spectrum of a point process using a centered moving average of the periodogram of the counting process. (29/1980)

151

Smoothed spectral estimate for bivariate point processes. (29/1980)

152

Cumulative hypergeometric probabilities. (This is actually AS R77) (29/1980, revised in 38/1989)

153

Distribution of weighted sum of squares of normal variables. (29/1980) Pan's procedure for the tail probabilities of the Durbin-Watson statistic.

154

Exact maximum likelihood estimation of autoregressive-moving average models by Kalman filtering. (29/1980) See also AS 182.

155

Distribution function of a linear combination of non-central chi- squared random variables. (29/1980) See also AS 204. This is a Fortran translation supplied by the author.

157

The runs-up and runs-down tests. (30/1981)

158

Calculation of probabilities for inferences under simple order restrictions. (30/1981) See also AS 198.

159

Generate random 2-way table with given marginal totals. (30/1981)

160

Partial and marginal association in multi-dimensional contingency tables. (30/1981)

161

Critical regions of an unconditional non-randomized test of homogeneity in 2 x 2 contingency tables. (30/1981)

162

Multivariate Conditional Logistic Analysis of Stratum-matched Case-control Studies. (30/1981) Includes a Fortran version of CACM algorithm 382 for generating all combinations of M out of N items.

163

A Givens Algorithm for Moving from one Linear Model to another without Going back to the Data. (30/1981)

164

Least squares subject to linear constraints. (30/1981)

165

Discriminant analysis of categorical data. (30/1981)

166

Calculates the entanglement matrix for a specified design. (30/1981)

167

Calculates efficiencies of estimation and their generalized inverse. (30/1981)

168

Calculates 'neat' values for plotting scales. (30/1981)

169

Produces scatter plots. (30/1981)

170

Computation of probability and non-centrality parameter of a non- central chi-square distribution. (30/1981)

171

Fisher's exact variance test for the Poisson distribution. (31/1982)

172

Generates indices for simulated nested DO-loops. Actually converts (either way) between a single index and a vector of subscripts. (31/1982)

173

Generates design matrix for balanced factorial experiments. (31/1982)

174

Multivariate rank sum test and median test. (31/1982)

175

Cramer-Wold factorization of self-reciprocal polynomials as the product of two polynomials. (31/1982)

176

Kernel density estimation using the fast Fourier transform. (31/1982) Also contains an alternative set of routines for density estimation.

177

Expected values of normal order statistics. (31/1982)

178

Gauss-Jordan sweep operator with multi-collinearity detection. (31/1982)

179

Enumeration of all permutations of multi-sets with fixed repetition numbers. (31/1982)

180

Linear rank estimate of the standard deviation after symmetric trimming. (31/1982)

181

Withdrawn. See R94 below

182

Finite-sample prediction from ARIMA processes. (31/1982) Uses AS 154.

183

The Wichmann & Hill random number generator. An alternative is also provided. (31/1982)

184

Non-central studentized maximum and related multiple-t probabilities. (31/1982)

185

Backward elimination procedure to find best-fitting log-linear models for contingency tables. (31/1982) Uses AS 51.

186

Discrete Fast Fourier Transform with data permutation. Series length must be a power of 2. (31/1982)

187

Derivatives of the incomplete gamma integral. (31/1982)

188

Estimation of the order of dependence in sequences. (32/1983)

189

Maximum likelihood estimation for the beta binomial distribution. (32/1983)

190

Distribution function & its inverse, for the studentized range. (32/1983)

191

Approximate likelihood calculation for ARMA and seasonal ARMA models. Includes a routine for the Banachiewicz (or modified Cholesky) factorization A = LDL'. (32/1983)

192

Calculate approximate percentage points using Pearson curves and the first three or four moments. (32/1983)

193

The Knuth spectral test for congruential random number generators. (32/1983)

194

Test of goodness of fit of ARMA models. (32/1983)

195

Multivariate normal probabilities for a rectangular region in N- dimensional space. A version which calls IMSL routines is also included. (33/1984) See AS 251 for a special case.

196

Conditional multivariate logistic analysis of stratified case-control studies. (33/1984)

197

Likelihood function for an ARMA process. (33/1984)

198

Calculation of level probabilities for order-restricted inference. (33/1984)

199

Branch and bound algorithm to find the subset which maximizes a quadratic form. (33/1984)

200

Approximate the sum of squares of normal score. (33/1984)

201

Combine predictions about a statistic based on the orderings of a set of means with an F-test of differences between the means. (33/1984)

202

Data-based nonparametric hazard estimation. (33/1984)

203

Maximum likelihood estimation of mixtures of distributions (normal, exponential, Poisson and binomial). (33/1984) See also AS221. This is a translation from Algol into Fortran.

204

Distribution of a sum of non-central chi-squared variables. (33/1984) Translation from Algol into Pascal.

205

Enumeration of all R x C contingency tables with given row and column totals, and calculation of hypergeometric probability for each table. (33/1984)

206

Isotonic regression in two independent variables. (33/1984)

207

Fit a generalized log-linear model. (33/1984)

208

Fit multivariate logistic model by the method of moments. (34/1985)

209

The distribution function of skewness and kurtosis. (34/1985)

210

Fit 5-parameter Johnson SB curves by moments. (34/1985)

211

The F-G diagonalization algorithm. (34/1985) Modifications in ASR 71 & ASR 74 have been added to the file.

212

Fitting the exponential curve by least squares. (34/1985)

213

Generation of correlation matrices with specified eigenvalues. (34/1985)

214

Calculation of Monte Carlo confidence intervals. (34/1985)

215

Maximum likelihood estimation of the parameters of the generalized extreme-value distribution. (34/1985). Updated on [15/Nov/99]

216

Maximum likelihood fitting when model contains a linear part and auxiliary parameters. (34/1985)

217

Computation of the Dip statistic to test for unimodality. (34/1985)

218

Elements of the Fisher information matrix for the smallest extreme- value distribution, also allows for censored data. (35/1986)

219

Height balanced trees. (35/1986)

220

Operating characteristics of James-Stein and Efron-Morris estimation. (35/1986)

221

Maximum likelihood estimation of a mixing distribution. (35/1986)

222

Resistant smoothing using the fast Fourier transform. (36/1987)

223

Optimum ridge parameter selection. (36/1987)

224

Combining component designs to form a design with several orthogonal blocking factors. (36/1987)

225

Minimizing linear inequality-constrained Mahalanobis distances. (36/1987)

226

Cumulative probabilities for the non-central beta distribution. (36/1987)

227

Generate all possible N-bit binary codes. (36/1987)

228

Finding I-projections (maximizing cross-entropy) subject to a finite set of linear inequality constraints. (36/1987)

229

Quantile regression. (36/1987)

230

Distribution of customers in M/G/m queues using an approximation due to Hokstad. (36/1987)

231

Distribution of a noncentral chi-squared variable with non-negative degrees of freedom. (36/1987) In Pascal.

232

Computation of population and sample correlation and partial correlation matrices in MARMA(p,q) time series. (37/1988)

233

Branch and bound algorithm for feature subset selection. (37/1988)

234

Approximating the percentage points of simple linear rank statistics with Cornish-Fisher expansions. (37/1988)

235

Tally frequencies of distinct real values. (37/1988)

236

Recursive enumeration of RxC tables for exact likelihood evaluation. (37/1988) In Pascal.

237

The corner method for identifying autoregressive moving average models. (37/1988)

238

Recursive procedure for the L1-norm fitting of a straight line. (37/1988) In Pascal.

239

Incomplete gamma function. (37/1988)

240

Updating the inverse of corrected sums of squares and products. (37/1988)

241

Inverse normal - more accurate than AS 111. (37/1988)

242

The exact likelihood of a vector autoregressive moving average model. (38/1989)

243

Cumulative distribution function of the non-central t-distribution. (38/1989)

244

Decomposability and collapsibility for log-linear models. (38/1989) In Pascal.

245

Log of the gamma function. (38/1989) The file includes a slower but more accurate algorithm which is not an AS algorithm.

246

Repeated measurements analysis of variance with unknown autoregressive parameter. (38/1989)

247

Updating the sufficient configurations for fitting ZPA (zero partial association) models to multi-dimensional contingency tables. (38/1989) In Pascal.

248

Empirical distribution function goodness-of-fit tests for the uniform, normal and exponential distributions. (38/1989)

249

Mean and covariance estimation for the truncated multivariate normal distribution. (38/1989)

250

Test the equality of dispersion matrices using methods due to Puri & Sen (distribution-free) and Anderson (normal distribution). (38/1989)

251

Multivariate normal probability integrals with product correlation structure. (38/1989)

252

Generating classes for log-linear models. (39/1990)

253

Maximum likelihood estimation of the RC(M) association model. (39/1990)

254

Fit mixture of normal distributions to grouped & truncated data. (39/1990)

255

Fitting two-way tables by means for rows, columns and cross-term. (39/1990) In Algol.

256

Distribution of a quadratic form in normal variables. (39/1990) In Pascal.

257

Isotonic regression for umbrella orderings. (39/1990)

258

Average run lengths for cumulative sum schemes. (39/1990)

259

Extending confidence intervals by the Robbins-Monro search process. (39/1990)

260

Distribution function of the square (R^2) of the sample multiple- correlation coefficient. (40/1991)

261

Quantiles of the distribution of square (R^2) of the sample multiple- correlation coefficient. (40/1991)

262

The Wei-Lachin two-sample test, the generalized Wilcoxon test and the log-rank test for incomplete multivariate observations with censoring. (40/1991)

263

Construction of irredundant test sets. (40/1991)

264

Printing of bit patterns. (40/1991)

265

Waiting time distribution for the G/G/1 queue using the Fast Fourier transform. (40/1991)

266

Maximum likelihood estimation of the parameters of the Dirichlet distribution. (40/1991)

267

Calculate lower bound for the probability of correct selection of a subset containing the best populations from a set of populations. (40/1991)

268

Calculates statistics for all possible subset regressions using an orthogonal decomposition. (40/1991)

269

Calculates the Cornish-Fisher adjustment to distribution functions (from the normal distribution function) using higher cumulants. (41/1992)

270

Maximum likelihood fitting of a 'key' density function, e.g. normal distribution, multiplied by a series in Hermite polynomials. (41/1992)

271

Optimal (smallest mis-classification probability) joint classification procedure. (41/1992) See also AS 276.

272

Produce character Box-plots using supplied quartiles. (41/1992)

273

Compare the least-squares fit of two subsets of regression variables using Spj0tvoll's method. (41/1992)

274

Least squares algorithms to supplement those of Gentleman in AS 75. (41/1992)

274-90

Algorithm 274, translated into Fortran 90.

275

Non-central chi-squared distribution function, number of degrees of freedom must be positive but not necessarily integer. (41/1992)

276

Optimal combinatoric classification for two normal classes. (41/1992) See also AS 271.

277

The Oja bivariate median. (41/1992)

278

The distribution of quadratic forms of multivariate generalized Student variables. (41/1992)

279

P-values for the generalized/alternative Durbin-Watson statistic with arbitrary lag using a Cholesky transformation method. (42/1993)

280

The power function for Fisher's exact test. (42/1993)

281

Scaling and rounding regression coefficients to force them to be integers. (42/1993)

282

High breakdown regression and multivariate estimation. (42/1993)

283

Rapid computation of the permutation paired and grouped t-tests. (42/1993) In ANSI Standard C.

284

Null distribution of a statistic for testing sphericity and additivity: a Jacobi polynomial expansion. (42/1993)

285

Calculate multivariate normal probabilities using Monte Carlo methods for a general region defined by a user-supplied function. (42/1993)

286

Estimation of parameters when there are errors in both the predictor variables and the dependant variable using least squares with a general model (includes implicit non-linear regression). (42/1993)

287

Adaptive rejection sampling (to generate random variables) from log-concave density functions. (42/1993)

288

P-value calculation for the generalized two-sample Smirnov tests. The tests are conditional on ties in the pooled sample. (43/1994)

289

Convolves hypergeometric distributions generated by several 2x2 tables (43/1994)

290

Generate a grid of variance ratios for contour plotting of confidence regions for a pair of parameters in non-linear regression. (43/1994)

291

Calculates overlap capability of subsequence SSEQ of length L. (43/1994)

292

Computes Fisher information matrix elements for time (type I) or failure (type II)

censored units from the smallest extreme value (sev), largest extreme value (lev), normal or logistic distribution. (43/1994)

293

Convolves conditional distributions generated by several 2xK tables (43/1994)

294

Pascal program for ....?

295

Heuristic algorithm to pick N rows of X out of NCAND to maximize the determinant of X'X, using the Fedorov exchange algorithm. (43/1994)

R94

calculates Shapiro-Wilk normality test and P-value for sample sizes 3 <= n <= 5000 . Handles censored or uncensored data. Corrects AS 181, which was found to be inaccurate for n > 50. (Uses function POLY in AS 181) by Patrick Royston (44/1995)

R96

Interval for trapezoidal rule integration to limit error using moment generating function bound for weighted sum of chi-squared(1) random variables.

297

Nonparametric regression using ultraspherical polynomials.

298

Hybrid minimization routine using simulated annealing and any local minimizer supplied by the user.

300

Efficient algorithm for determining a simulated percentile point with a fixed degree of accuracy.

301

Computes the logarithms of F1, P(H) and P(H').

302

Subroutine to compute a multiple isotonic regression for umbrella ordering with known peak.

303

Generation of ordered multinomial frequencies. Given integers N (N >= 0) and K (K >= 1), generates all possible integer arrays NI of size K, whose elements are non-negative, non-decreasing and sum to N.

304

Fisher's non-parametric randomization test for two small independent random samples.

305

Compute the ARL of a CUSUM mean chart with linear drift.

306

Calculation of a BIB product operation on any two matrices.

307

Calculation of the simplicial depth and the halfspace depth

310

Computes the cumulative distribution function of a non-central beta random variable.

314

Inverts matrix with contents subject to modulo arithmetic.

319

A program to implement a quasi-newton method. Using new algorithm Varmet July 1994

N.B. This library was maintained by Alan Miller, CSIRO Division of Mathematics and Statistics, Clayton, Victoria 3169. E-mail: alan@mel.dms.csiro.au Phone: (03)542-2266 or (03)542-2253 (Secretary) FAX: (03)542-2474

Recent algorithms are provided by "T.R.Hopkins" <T.R.Hopkins@ukc.ac.uk>. The StatLib collection of Applied Statistics Algorithms is maintained by Pantelis Vlachos, vlachos@stat.cmu.edu.

Credit where credit is due

If you use an algorithm, dataset, or other information from StatLib, please acknowledge both StatLib and the original contributor of the material.

Last modified: Mon Nov 15 17:25:32 EST 1999 by Pantelis Vlachos

```
CSTART OF AS 3
      REAL FUNCTION PROBST(T, IDF, IFAULT)
C
C        ALGORITHM AS 3  APPL. STATIST. (1968) VOL.17, P.189
C
C        STUDENT T PROBABILITY (LOWER TAIL)
C
      REAL A, B, C, F, G1, S, FK, T, ZERO, ONE, TWO, HALF, ZSQRT, ZATAN
C
C        G1 IS RECIPROCAL OF PI
C
      DATA ZERO, ONE, TWO, HALF, G1
     $     /0.0, 1.0, 2.0,  0.5, 0.3183098862/
C
      ZSQRT(A) = SQRT(A)
      ZATAN(A) = ATAN(A)
C
      IFAULT = 1
      PROBST = ZERO
      IF (IDF .LT. 1) RETURN
      IFAULT = 0
      F = IDF
      A = T / ZSQRT(F)
      B = F / (F + T ** 2)
      IM2 = IDF - 2
      IOE = MOD(IDF, 2)
      S = ONE
      C = ONE
      F = ONE
      KS = 2 + IOE
      FK = KS
      IF (IM2 .LT. 2) GOTO 20
      DO 10 K = KS, IM2, 2
      C = C * B * (FK - ONE) / FK
      S = S + C
      IF (S .EQ. F) GOTO 20
      F = S
      FK = FK + TWO
   10 CONTINUE
   20 IF (IOE .EQ. 1) GOTO 30
      PROBST = HALF + HALF * A * ZSQRT(B) * S
      RETURN
   30 IF (IDF .EQ. 1) S = ZERO
      PROBST = HALF + (A * B * S + ZATAN(A)) * G1
      RETURN
      END
CEND OF AS 3
```

```
CSTART OF AS 5
      REAL FUNCTION PRNCST(ST, IDF, D, IFAULT)
C
C        ALGORITHM AS 5  APPL. STATIST. (1968) VOL.17, P.193
C
C        COMPUTES LOWER TAIL AREA OF NON-CENTRAL T-DISTRIBUTION
C
      REAL ST, D, G1, G2, G3, ZERO, ONE, TWO, HALF, EPS, EMIN, F,
     $  A, B, RB, DA, DRB, FMKM1, FMKM2, SUM, AK, FK, FKM1,
     $  ALNORM, TFN, ALOGAM, ZSQRT, ZEXP
C
C        CONSTANTS - G1 IS 1.0 / SQRT(2.0 * PI)
C                    G2 IS 1.0 / (2.0 * PI)
C                    G3 IS SQRT(2.0 * PI)
C
      DATA G1, G2, G3 /0.3989422804, 0.1591549431, 2.5066282746/
      DATA ZERO, ONE, TWO, HALF,    EPS, EMIN
     $     /0.0, 1.0, 2.0,  0.5, 1.0E-6, 12.5/
C
      ZSQRT(A) = SQRT(A)
      ZEXP(A) = EXP(A)
C
      F = IDF
      IF (IDF .GT. 100) GOTO 50
      IFAULT = 0
      IOE = MOD(IDF, 2)
      A = ST / ZSQRT(F)
      B = F / (F + ST ** 2)
      RB = ZSQRT(B)
      DA = D * A
      DRB = D * RB
      SUM = ZERO
      IF (IDF .EQ. 1) GOTO 30
      FMKM2 = ZERO
      IF (ABS(DRB) .LT. EMIN) FMKM2 = A * RB * ZEXP(-HALF * DRB ** 2)
     $  * ALNORM(A * DRB, .FALSE.) * G1
      FMKM1 = B * DA * FMKM2
      IF (ABS(D) .LT. EMIN)
     $  FMKM1 = FMKM1 + B * A * G2 * ZEXP(-HALF * D ** 2)
      IF (IOE .EQ. 0) SUM = FMKM2
      IF (IOE .EQ. 1) SUM = FMKM1
      IF (IDF .LT. 4) GOTO 20
      IFM2 = IDF - 2
      AK = ONE
      FK = TWO
      DO 10 K = 2, IFM2, 2
      FKM1 = FK - ONE
      FMKM2 = B * (DA * AK * FMKM1 + FMKM2) * FKM1 / FK
      AK = ONE / (AK * FKM1)
      FMKM1 = B * (DA * AK * FMKM2 + FMKM1) * FK / (FK + ONE)
      IF (IOE .EQ. 0) SUM = SUM + FMKM2
      IF (IOE .EQ. 1) SUM = SUM + FMKM1
      AK = ONE / (AK * FK)
      FK = FK + TWO
   10 CONTINUE
   20 IF (IOE .EQ. 0) GOTO 40
   30 PRNCST = ALNORM(DRB, .TRUE.) + TWO * (SUM + TFN(DRB, A))
      RETURN
   40 PRNCST = ALNORM(D, .TRUE.) + SUM * G3
      RETURN
C
```

```
C        NORMAL APPROXIMATION - K IS NOT TESTED AFTER THE TWO CALLS
C        OF ALOGAM, BECAUSE A FAULT IS IMPOSSIBLE WHEN F EXCEEDS 100
C
   50 IFAULT = 1
      A = ZSQRT(HALF * F) * ZEXP(ALOGAM(HALF * (F - ONE), K)
     $  - ALOGAM(HALF * F, K)) * D
      PRNCST = ALNORM((ST - A) / ZSQRT(F * (ONE + D ** 2)
     $  / (F - TWO) - A ** 2), .FALSE.)
      RETURN
      END
CEND OF AS 5
```

```
C This file contains AS6 and the enhanced version ASR44.   See AS7 also.
C
C
      SUBROUTINE CHOL (A,N,NN,U,NULLTY,IFAULT)
C
C       Algorithm AS6, Applied Statistics, vol.17, (1968)
C
C       Given a symmetric matrix order n as lower triangle in a( )
C       calculates an upper triangle, u( ), such that uprime * u = a.
C       a must be positive semi-definite.  eta is set to multiplying
C       factor determining effective zero for pivot.
C
C       arguments:-
C       a()     = input, a +ve definite matrix stored in lower-triangula
C                 form.
C       n       = input, the order of a
C       nn      = input, the size of the a and u arrays      n*(n+1)/2
C       u()     = output, a lower triangular matrix such that u*u' = a.
C                 a & u may occupy the same locations.
C       nullty  = output, the rank deficiency of a.
C       ifault  = output, error indicator
C                       = 1 if n < 1
C                       = 2 if a is not +ve semi-definite
C                       = 3 if nn < n*(n+1)/2
C                       = 0 otherwise
C
C*********************************************************************
C
      DOUBLE PRECISION A(NN),U(NN),ETA,ETA2,X,W,ZERO
C
C       The value of eta will depend on the word-length of the
C       computer being used.  See introductory text.
C
      DATA ETA,ZERO/1.D-9,0.0D0/
C
      IFAULT=1
      IF (N.LE.0) RETURN
      IFAULT=3
      IF (NN.LT.N*(N+1)/2) RETURN
      IFAULT=2
      NULLTY=0
      J=1
      K=0
      ETA2=ETA*ETA
      II=0
C
C       Factorize column by column, icol = column no.
C
      DO 80 ICOL=1,N
        II=II+ICOL
        X=ETA2*A(II)
        L=0
        KK=0
C
C       IROW = row number within column ICOL
C
        DO 40 IROW=1,ICOL
          KK=KK+IROW
          K=K+1
          W=A(K)
          M=J
```

```
            DO 10 I=1,IROW
              L=L+1
              IF (I.EQ.IROW) GO TO 20
              W=W-U(L)*U(M)
              M=M+1
 10         CONTINUE
 20         IF (IROW.EQ.ICOL) GO TO 50
            IF (U(L).EQ.ZERO) GO TO 30
            U(K)=W/U(L)
            GO TO 40
 30         IF (W*W.GT.ABS(X*A(KK))) RETURN
            U(K)=ZERO
 40       CONTINUE
 50       IF (ABS(W).LE.ABS(ETA*A(K))) GO TO 60
          IF (W.LT.ZERO) RETURN
          U(K)=SQRT(W)
          GO TO 70
 60       U(K)=ZERO
          NULLTY=NULLTY+1
 70       J=J+ICOL
 80     CONTINUE
        IFAULT=0
        END
C
C
C
C
        SUBROUTINE SUBCHL (A,B,N,U,NULLTY,IFAULT,NDIM,DET)
C
C       REMARK ASR 44  APPL. STATIST. (1982) VOL. 31, NO. 3
C
C       A revised and enhanced version of
C       ALGORITHM AS 6  APPL. STATIST. (1968) VOL. 17, NO. 2
C
C       Given a symmetric matrix of order N as lower triangle in A(),
C       calculates an upper triangle, U(), such that U'U = the sub-matrix
C       of A whose rows and columns are specified in the integer array
C       B().
C       U() may coincide with A().   A() must be +ve semi-definite.
C       ETA is set to multiplying factor determining effective zero for
C       a pivot.
C       NULLTY is returned as number of effective zero pivots.
C       IFAULT is returned as 1 if N <= 0, 2 if A() is not +ve semi-
C       definite, otherwise 0 is returned.
C
        DOUBLE PRECISION A(NDIM),U(NDIM),DET
        INTEGER B(N)
C
C       Local variables
C
        DOUBLE PRECISION ETA,ONE,ZERO,W,ETA2
C
C       The value of ETA below will depend upon the word length of the
C       computer being used.
C
        DATA ETA/1.D-14/,ONE/1.D0/,ZERO/0.D0/
C
        IFAULT=1
        IF (N.LE.0) GO TO 90
        IFAULT=2
        NULLTY=0
```

```
      DET=ONE
      J=1
      K=0
      ETA2=ETA*ETA
      DO 80 ICOL=1,N
        IJ=B(ICOL)*(B(ICOL)-1)/2
        II=IJ+B(ICOL)
        X=ETA2*A(II)
        L=0
        DO 40 IROW=1,ICOL
          KK=B(IROW)*(B(IROW)+1)/2
          K=K+1
          JJ=IJ+B(IROW)
          W=A(JJ)
          M=J
          DO 10 I=1,IROW
            L=L+1
            IF (I.EQ.IROW) GO TO 20
            W=W-U(L)*U(M)
            M=M+1
 10       CONTINUE
 20       IF (IROW.EQ.ICOL) GO TO 50
          IF (U(L).EQ.ZERO) GO TO 30
          U(K)=W/U(L)
          GO TO 40
 30       IF (W*W.GT.ABS(X*A(KK))) GO TO 90
          U(K)=ZERO
 40     CONTINUE
 50     IF (ABS(W).LE.ABS(ETA*A(KK))) GO TO 60
        IF (W.LT.ZERO) GO TO 90
        U(K)=SQRT(W)
        GO TO 70
 60     U(K)=ZERO
        NULLTY=NULLTY+1
 70     J=J+ICOL
        DET=DET*U(K)*U(K)
 80   CONTINUE
C
      IFAULT=0
 90   RETURN
      END
```

```
c This file contains AS7 and an enhanced alternative - ASR44.  See also AS6.
c
c
      subroutine syminv(a, n, nn, c, w, nullty, ifault)
c
c     Algorithm AS7, Applied Statistics, vol.17, 1968, p.198.
c
c     Forms in c( ) as lower triangle, a generalised inverse
c     of the positive semi-definite symmetric matrix a( )
c     order n, stored as lower triangle.
c
c     arguments:-
c     a()     = input, the symmetric matrix to be inverted, stored in
c               lower triangular form
c     n       = input, order of the matrix
c     nn      = input, the size of the a and c arrays     n*(n+1)/2
c     c()     = output, the inverse of a (a generalized inverse if c is
c               singular), also stored in lower triangular.
c               c and a may occupy the same locations.
c     w()     = workspace, dimension at least n.
c     nullty  = output, the rank deficiency of a.
c     ifault  = output, error indicator
c                    = 1 if n < 1
c                    = 2 if a is not +ve semi-definite
c                    = 3 if nn < n*(n+1)/2
c                    = 0 otherwise
c
c*********************************************************************
c
      double precision a(nn), c(nn), w(n), x, zero, one
c
      data zero, one /0.0d0, 1.0d0/
c
c     cholesky factorization of a, result in c
c
      call chol(a, n, nn, c, nullty, ifault)
      if(ifault.ne.0) return
c
c     invert c & form the product (cinv)'*cinv, where cinv is the inverse
c     of c, row by row starting with the last row.
c     irow = the row number, ndiag = location of last element in the row.
c
      irow=n
      ndiag=nn
   10 l=ndiag
      if (c(ndiag) .eq. zero) goto 60
      do 20 i=irow,n
        w(i)=c(l)
        l=l+i
   20 continue
      icol=n
      jcol=nn
      mdiag=nn
   30 l=jcol
      x=zero
      if(icol.eq.irow) x=one/w(irow)
      k=n
   40 if(k.eq.irow) go to 50
      x=x-w(k)*c(l)
      k=k-1
      l=l-1
```

```
            if(l.gt.mdiag) l=l-k+1
            go to 40
      50    c(l)=x/w(irow)
            if(icol.eq.irow) go to 80
            mdiag=mdiag-icol
            icol=icol-1
            jcol=jcol-1
            go to 30
      60    do 70 j=irow,n
              c(l)=zero
              l=l+j
      70    continue
      80    ndiag=ndiag-irow
            irow=irow-1
            if(irow.ne.0) go to 10
            return
            end
c
c
c
c
            subroutine subinv(a, nm, b, n, c, w, nullty, ifault, ndim, det)
            implicit double precision (a-h,o-z)
c
c       Remark asr 44.1   Appl. Statist. (1982) Vol.31, No.3
c
c       A revised and enhanced version of
c       algorithm as7, applied statistics, vol.17, 1968.
c
c       Forms in c() as lower triangle, a generalised inverse
c       of a sub-matrix of the positive semi-definite symmetric
c       matrix a() of order n, stored as lower triangle.
c       c() may co-incide with a().  nullty is returned as the nullity
c       of a().  ifault is returned as 1 if n.lt.1, 2 if the
c       submatrix is not positive semi-definite, 3 if the elements
c       of b are inadmissible, otherwise zero.
c       w() is a work array of length at least n that is allocated by
c       the calling routine.
c
c       arguments:-
c       a()     = input, the symmetric matrix to be inverted, stored in
c                 lower triangular form
c       nm      = input, the order of a
c       n       = input, order of the sub-matrix to be inverted
c       b       = input, an array containing the numbers of rows and
c                 columns of a() to be included.
c       c()     = output, the inverse of a (a generalized inverse if c is
c                 singular), also stored in lower triangular.
c                 c and a may occupy the same locations.
c       w()     = workspace, dimension at least n.
c       nullty  = output, the rank deficiency of a.
c       ifault  = output, error indicator
c                       = 1 if n < 1
c                       = 2 if the submatrix of a is not +ve semi-definite
c                       = 3 if elements of b are inadmissible
c                       = 0 otherwise
c
c     Auxiliary routine required: SUBCHL from ASR44 (see AS6)
c
c*******************************************************************
c
```

```
      dimension a(ndim),c(ndim),w(n)
      integer b(n)
      data zero/0.0d0/, one/1.0d0/
c
      ifault = 3
      if (n .gt. nm) return
      if (b(1) .lt.1 .or. b(1) .gt. nm-n+1) return
      if (n .eq. 1) goto 19
      do 18 i = 2, n
        if (b(i) .le. b(i-1) .or. b(i) .gt. nm-n+i) return
   18 continue
   19 nrow=n
      ifault=1
      if(nrow.le.0) go to 100
      ifault=0
c
c     Cholesky factorization of a, result in c
c
      call subchl(a,b,nrow,c,nullty,ifault,ndim,det)
      if(ifault.ne.0) go to 100
c
c     invert c & form the product (cinv)'*cinv, where cinv is the inverse
c     of c, row by row starting with the last row.
c     irow = the row number, ndiag = location of last element in the row.
c
      nn=nrow*(nrow+1)/2
      irow=nrow
      ndiag=nn
   10 if(c(ndiag).eq.zero) go to 60
      l=ndiag
      do 20 i=irow,nrow
        w(i)=c(l)
        l=l+i
   20 continue
      icol=nrow
      jcol=nn
      mdiag=nn
   30 l=jcol
      x=zero
      if(icol.eq.irow) x=one/w(irow)
      k=nrow
   40 if(k.eq.irow) go to 50
      x=x-w(k)*c(l)
      k=k-1
      l=l-1
      if(l.gt.mdiag) l=l-k+1
      go to 40
   50 c(l)=x/w(irow)
      if(icol.eq.irow) go to 80
      mdiag=mdiag-icol
      icol=icol-1
      jcol=jcol-1
      go to 30
   60 l=ndiag
      do 70 j=irow,nrow
        c(l)=zero
        l=l+j
   70 continue
   80 ndiag=ndiag-irow
      irow=irow-1
      if(irow.ne.0) go to 10
```

```
      100   return
            end
```

```
CSTART OF AS 13
      SUBROUTINE PRTREE(N, M, A, DLARGE, D, B, C, IFAULT)
C
C        ALGORITHM AS 13  APPL. STATIST. (1969) VOL.18, P.103
C
C        COMPUTES THE MINIMUM SPANNING TREE OF A DISTANCE MATRIX
C
      REAL D(M), C(N), AM, DIST, DLARGE
      INTEGER B(N)
      LOGICAL A(N)
      IFAULT = 1
      IF (N .LT. 2) RETURN
      IFAULT = 2
      IF (N * (N - 1) / 2 .NE. M) RETURN
      IFAULT = 0
C
C        A(I) IS .FALSE. IF I IS ALREADY ASSIGNED TO
C        THE TREE (INITIALLY CONSISTING OF NO. 1 ONLY),
C        OR .TRUE. OTHERWISE
C
      DO 10 I = 2, N
      A(I) = .TRUE.
      B(I) = 0
      C(I) = DLARGE
   10 CONTINUE
      J = 1
      DO 40 I = 2, N
      AM = DLARGE
      DO 30 K = 2, N
      IF (.NOT. A(K)) GOTO 30
      IF (J .GT. K) L = (J - 1) * (J - 2) / 2 + K
      IF (J .LE. K) L = (K - 1) * (K - 2) / 2 + J
      DIST = D(L)
      IF (DIST .GE. C(K)) GOTO 20
      C(K) = DIST
      B(K) = J
   20 IF (AM .LE. C(K)) GOTO 30
      AM = C(K)
      NEXT = K
   30 CONTINUE
      J = NEXT
      A(J) = .FALSE.
   40 CONTINUE
      RETURN
      END
CEND OF AS 13
```

```
CSTART OF AS 14
      SUBROUTINE MTP(N, B, C, HIST, ROUTE, XPRINT)
C
C        ALGORITHM AS 14  APPL. STATIST. (1969) VOL.18, P.105
C
C        THIS SUBROUTINE ENABLES THE MINIMUM SPANNING TREE TO BE
C        DRAWN RAPIDLY WITHOUT HAVING TO SEARCH FOR END POINTS,
C        AND IS ESPECIALLY USEFUL WHEN N EXCEEDS 100. THE OUTPUT
C        OF SUBROUTINE PRTREE IS USED
C
      INTEGER B(N), HIST(N), ROUTE(N)
      REAL C(N)
      DO 10 I = 1, N
   10 HIST(I) = 0
      DO 20 I = 2, N
      J = B(I)
      HIST(J) = HIST(J) + 1
   20 CONTINUE
      ROUTE(1) = 1
      J = 1
      K = 1
C
C        ROUTE(J) IS THE CURRENT END POINT AND IF HIST(ROUTE(I))
C        IS NOT ZERO A FURTHER LINE CAN BE FOUND. HIST(I) MUST
C        BE NON-ZERO INITIALLY BECAUSE THE TREE IS CONNECTED
C
      DO 80 I = 2, N
   30 IF (HIST(K) .NE. 0) GOTO 40
      J = J - 1
      K = ROUTE(J)
      GOTO 30
   40 HIST(K) = HIST(K) - 1
      DO 50 M = 2, N
      IF (K .EQ. B(M)) GOTO 60
   50 CONTINUE
   60 CALL XPRINT(J, K, M, C(M))
      J = J + 1
      ROUTE(J) = M
      K = M
      B(M) = -B(M)
   80 CONTINUE
      DO 90 I = 2, N
   90 B(I) = IABS(B(I))
      RETURN
      END
CEND OF AS 14
```

```
CSTART OF AS 15
      SUBROUTINE SLINK(N, N1, N20, DELTA, B, C, DLARGE,
     $  G, H, X, W1, W2, GROUPP, TOPP, PRINTX, IFAULT)
C
C        ALGORITHM AS 15  APPL. STATIST. (1969) VOL.18, P.106
C
C        PERFORMS SINGLE LINKAGE CLUSTERING. INFORMATION IS
C        SUPPLIED BY SUBROUTINE PRTREE, IN ARRAYS B AND C.
C        POINTS ARE LISTED IN SORTED ORDER IN ARRAY G, AND THE
C        CORRESPONDING ARRAY H MARKS THE LAST MEMBER OF EACH
C        CLUSTER WITH A 1, OTHERWISE THE ENTRY IS 0. THE ARRAY
C        X STORES CODE NUMBERS FOR OUTPUT OF THE DENDROGRAM
C
      INTEGER P, Q, R, S, T, U, V, W, B(N), G(N), H(N),
     $  X(N20), W1(N1), W2(N1)
      REAL DELTA, C(N), DLARGE, D, LEVEL, ONE, ZINT
C
      DATA ONE /1.0/
C
      ZINT(D) = AINT(D)
C
      IFAULT = 1
      IF (N1 .NE. N + 1) RETURN
      IF (N20 .NE. 20 * N) RETURN
      IFAULT = 0
C
C        CLUSTERING STARTS AT THE FIRST INTEGRAL MULTIPLE OF
C        DELTA WHICH IS GREATER THAN D, THE SHORTEST LINK OF
C        THE MINIMUM SPANNING TREE
C
      D = C(2)
      IF (N .LT. 3) GOTO 15
      DO 10 I = 3, N
   10 IF (D .GT. C(I)) D = C(I)
   15 DO 20 I = 1, N
      G(I) = I
      H(I) = 1
      X(I) = 3
   20 CONTINUE
      P = 0
      LEVEL = DELTA * (ONE + ZINT(D / DELTA))
C
C        FOR EACH LINK IN ARRAY C THAT IS SHORTER THAN LEVEL,
C        TWO CLUSTERS ARE AMALGAMATED. THE AMALGAMATION
C        INVOLVES A REORDERING OF ARRAYS G AND H AND REMOVAL
C        OF THE END-OF-CLUSTER MARKER FROM THE EARLIER
C        CLUSTER. LINKS ONCE USED ARE INCREASED BY DLARGE TO
C        PREVENT RE-USE
C
   30 P = P + 1
      DO 150 I = 2, N
      IF (C(I) .GE. LEVEL) GOTO 150
      J = B(I)
      C(I) = C(I) + DLARGE
      K = I
      DO 40 M = 1, N
      IF (G(M) .EQ. J) Q = M
      IF (G(M) .EQ. K) R = M
   40 CONTINUE
      IF (Q .LE. R) GOTO 50
      M = R
```

```
      R = Q
      Q = M
   50 DO 60 S = Q, N
      IF (H(S) .NE. 0) GOTO 70
   60 CONTINUE
   70 T = R
   80 T = T - 1
      IF (T .LT. 1) GOTO 90
      IF (H(T) .EQ. 0) GOTO 80
   90 T = T + 1
      H(S) = 0
      L = 0
      DO 100 R = T, N
      L = L + 1
      W1(L) = G(R)
      W2(L) = H(R)
      IF (H(R) .NE. 0) GOTO 110
  100 CONTINUE
  110 W = S + 1
      M = T
      L = R + 1
  120 M = M - 1
      IF (M .LT. W) GOTO 130
      L = L - 1
      G(L) = G(M)
      H(L) = H(M)
      GOTO 120
  130 U = R - T + 1
      L = W - 1
      DO 140 M = 1, U
      L = L + 1
      G(L) = W1(M)
      H(L) = W2(M)
  140 CONTINUE
  150 CONTINUE
      CALL GROUPP(LEVEL, N, G, H)
      W = N * P
      U = 0
      V = 0
      K = 0
C
C        DENDROGRAM INDICATORS ARE NOW COMPILED AND STORED
C        IN X. POINTS ARE EXAMINED IN THE ORDER DEFINED BY
C        THE ARRAY G. S IS THE CORRESPONDING ENTRY ON THE
C        PREVIOUS ITERATION, U = 0 FOR THE FIRST MEMBER OF
C        A CLUSTER, V = 1 WHEN AMALGAMATIONS OCCUR, FROM
C        THE LAST MEMBER OF THE FIRST COMPONENT CLUSTER
C        UNTIL THE LAST MEMBER OF THE AMALGAMATED CLUSTER.
C        K IS THE TOTAL NUMBER OF CLUSTERS
C
      DO 160 I = 2, N
  160 K = K + H(I)
      IF (P .GT. 19) GOTO 270
      DO 260 I = 1, N
      J = G(I)
      L = J + W - N
      S = X(L)
      IF (U .NE. 0) GOTO 190
      IF (H(I) .NE. 1) GOTO 170
      T = 3
      GOTO 250
```

```
  170 IF (S .NE. 3) GOTO 180
      T = 1
      U = 1
      V = 1
      GOTO 250
  180 T = 0
      U = 1
      GOTO 250
  190 IF (H(I) .NE. 1) GOTO 210
      IF (V .NE. 0) GOTO 200
      T = 3
      U = 0
      GOTO 250
  200 T = 2
      U = 0
      V = 0
      GOTO 250
  210 IF (S .LT. 2 .OR. S .GT. 3) GOTO 230
      IF (V .NE. 0) GOTO 220
      T = 1
      U = 1
      V = 1
      GOTO 250
  220 T = 5
      U = 1
      GOTO 250
  230 IF (V .NE. 0) GOTO 240
      T = 0
      U = 1
      GOTO 250
  240 T = 4
  250 L = J + W
      X(L) = T
  260 CONTINUE
  270 LEVEL = LEVEL + DELTA
      IF (K .NE. 1) GOTO 30
      CALL TOPP
      DO 280 I = 1, N
  280 CALL PRINTX(I, G, N, P, X, N20)
      RETURN
      END
CEND OF AS 15
```

```
      SUBROUTINE YATES (Y, LEVS, KODE, NFAC, X, EXTR)
C
C     ALGORITHM  AS 22 APPL.STATIST. (1969) VOL.18, NO.3
C
C     Yates algorithm for general complete factorial experiments.
C     Computes  Y = (IL..X..IR).Y
C
C     where   . denotes  ordinary matrix product
C            ..          direct matrix product
C             IL         unit matrix of order NLFT
C             IR         unit matrix of order NRGT
C           Y( )         multi-way table stored as a vector
C           X( )         a square matrix stored as a vector
C
      INTEGER LEVS(*), KODE, NFAC
      REAL Y(*), X(*), EXTR(*)
C
C     Local variables
C
      INTEGER NLFT, NRGT, ILEV, NLEV, JUMP, ILFT, IRGT, JUMPHO, IEL,
     *        JEL, JUMPER
C
      NLFT = 1
      NRGT = 1
      DO 3 ILEV = 1, NFAC
        IF (ILEV - KODE .LT. 0) THEN
          NLFT = NLFT * LEVS(ILEV)
        ELSE IF (ILEV - KODE .GT. 0) THEN
          NRGT = NRGT * LEVS(ILEV)
        END IF
    3 CONTINUE
      NLEV = LEVS(KODE)
      JUMP = 0
C
C     Loop over left unit matrix
C
      DO 7 ILFT = 1, NLFT
        JUMP = JUMP + 1
C
C     Loop over right unit matrix
C
        DO 6 IRGT = 1, NRGT
          JUMPHO = JUMP
          JUMP = JUMP - NRGT
C
C     Extract vector elements for linear combination
C
          DO 4 ILEV = 1, NLEV
            JUMP = JUMP + NRGT
            EXTR(ILEV) = Y(JUMP)
    4     CONTINUE
          JUMP = JUMPHO - NRGT
          ILEV = 0
C
C     Begin matrix by vector product.
C     Loop over the various contrasts.
C
          DO 5 IEL = 1, NLEV
            JUMP = JUMP + NRGT
            Y(JUMP) = 0.0
C
```

```
C     Form linear combination.
C
            DO 5 JEL = 1, NLEV
              ILEV = ILEV + 1
              Y(JUMP) = Y(JUMP) + X(ILEV) * EXTR(JEL)
    5     CONTINUE
          JUMPER = JUMP
          JUMP = JUMPHO + 1
    6   CONTINUE
        JUMP = JUMPER
    7 CONTINUE
C
      RETURN
      END
```

```
      REAL FUNCTION STUDNT (T, DOFF, IFAULT)
C
C     ALGORITHM AS 27  APPL. STATIST. VOL.19, NO.1
C
C     Calculate the upper tail area under Student's t-distribution
C
C     Translated from Algol by Alan Miller
C
      INTEGER IFAULT
      REAL T, DOFF
C
C     Local variables
C
      REAL V, X, TT, TWO, FOUR, ONE, ZERO, HALF
      REAL A1, A2, A3, A4, A5, B1, B2,
     *     C1, C2, C3, C4, C5, D1, D2,
     *     E1, E2, E3, E4, E5, F1, F2,
     *     G1, G2, G3, G4, G5, H1, H2,
     *     I1, I2, I3, I4, I5, J1, J2
      LOGICAL POS
      DATA TWO /2.0/, FOUR /4.0/, ONE /1.0/, ZERO /0.0/, HALF /0.5/
      DATA A1, A2, A3, A4, A5 /0.09979441, -0.581821, 1.390993,
     *     -1.222452, 2.151185/, B1, B2 /5.537409, 11.42343/
      DATA C1, C2, C3, C4, C5 /0.04431742, -0.2206018, -0.03317253,
     *     5.679969, -12.96519/, D1, D2 /5.166733, 13.49862/
      DATA E1, E2, E3, E4, E5 /0.009694901, -0.1408854, 1.88993,
     *     -12.75532, 25.77532/, F1, F2 /4.233736, 14.3963/
      DATA G1, G2, G3, G4, G5 /-9.187228E-5, 0.03789901, -1.280346,
     *     9.249528, -19.08115/, H1, H2 /2.777816, 16.46132/
      DATA I1, I2, I3, I4, I5 /5.79602E-4, -0.02763334, 0.4517029,
     *     -2.657697, 5.127212/, J1, J2 /0.5657187, 21.83269/
C
C     Check that number of degrees of freedom > 4.
C
      IF (DOFF .LT. TWO) THEN
        IFAULT = 1
        STUDNT = - ONE
        RETURN
      END IF
C
      IF (DOFF .LE. FOUR) THEN
        IFAULT = DOFF
      ELSE
        IFAULT = 0
      END IF
C
C     Evaluate series.
C
      V = ONE / DOFF
      POS = (T .GE. ZERO)
      TT = ABS(T)
      X = HALF * (ONE +
     *    TT * (((A1 + V * (A2 + V * (A3 + V * (A4 + V * A5)))) /
     *        (ONE - V * (B1 - V * B2))) +
     *    TT * (((C1 + V * (C2 + V * (C3 + V * (C4 + V * C5)))) /
     *        (ONE - V * (D1 - V * D2))) +
     *    TT * (((E1 + V * (E2 + V * (E3 + V * (E4 + V * E5)))) /
     *        (ONE - V * (F1 - V * F2))) +
     *    TT * (((G1 + V * (G2 + V * (G3 + V * (G4 + V * G5)))) /
     *        (ONE - V * (H1 - V * H2))) +
     *    TT * ((I1 + V * (I2 + V * (I3 + V * (I4 + V * I5)))) /
```

```
     *        (ONE - V * (J1 - V * J2))) ))))) ** (-8)
      IF (POS) THEN
        STUDNT = X
      ELSE
        STUDNT = ONE - X
      END IF
C
      RETURN
      END
```

```
      SUBROUTINE HNPLOT (NTITLE,OBS,N)
C
C        Algorithm as 30 j.r.statist.soc. c. (1970) vol.19. no.2.
C
C        Half-normal plotting
C
      DIMENSION OBS(*), A(6), B(4), P(100), NTITLE(20)
      DIMENSION OUT(101), XPR(11), D(100), YPR(6)
      CHARACTER*1 OUT,DOT,BLANK,PLUS
 10   FORMAT (1H1,20A4,///)
 20   FORMAT (1H ,F11.4,101A1)
 30   FORMAT (1H ,11X,1H.)
 40   FORMAT (1H ,F11.4,101H.....................................
     1.....................................................)
 50   FORMAT (1H ,11X,101H.      .       .       .          .         .
     1         .      .                 .                .)
 60   FORMAT (1H ,8X,F6.2,F7.2,F8.2,F8.2,F11.2,F10.2,F11.2,F7.2,16X,
     1 F5.2,12X,F5.2)
 70   FORMAT (1H ,11X,101A1)
 80   FORMAT (1H ,F11.4,1H.)
C
C        Set iwrite equal to the output device number (system dependent)
C
      DATA IWRITE/6/
C
C     set constants for probability scale, and for printing
C
      DATA A(1),A(2),A(3),A(4),A(5),A(6)/1.048,-0.8559,0.363,0.108392,
     1 0.328117,1.253314/
      DATA B(1),B(2),B(3),B(4)/-0.001416,-0.039811,-0.6256,0.401703/
      DATA XPR(1),XPR(2),XPR(3),XPR(4),XPR(5),XPR(6),XPR(7),XPR(8),
     1XPR(9),XPR(10)/50.0,60.0,70.0,80.0,90.0,95.0,98.0,99.0,99.9,99.99/
      DATA DOT,BLANK,PLUS/'.',' ','+'/
C
      OUT(1)=DOT
C
C        Sort data into ascending order
C
      DO 100 I=1,N
        DO 100 J=I,N
        IF (OBS(J)-OBS(I)) 90,100,100
 90     TEMP=OBS(I)
        OBS(I)=OBS(J)
        OBS(J)=TEMP
 100  CONTINUE
C
C     Calculate scales for the axes (the x-axis is assumed to be
C     100 units long, and the y-axis 50 units long
C
      XSCALE=0.03719
      OBSMAX=OBS(N)
      KOUNT=0
 110  IF (OBSMAX.GT.100.0) GO TO 120
      IF (OBSMAX.GE.10.0) GO TO 130
      OBSMAX=OBSMAX*10.0
      KOUNT=KOUNT-1
      GO TO 110
 120  OBSMAX=OBSMAX/10.0
      KOUNT=KOUNT+1
      GO TO 110
 130  IF (AMOD(OBSMAX,5.0).EQ.0) GO TO 140
```

```
      OBSMAX=(AINT(OBSMAX/5.0)+1.0)*5.0
  140 OBSMAX=OBSMAX*(10.0**KOUNT)
      YSCALE=OBSMAX/50.0
      YPR(1)=OBSMAX
      YPR(6)=0.0
      OBSMAX=OBSMAX/5.0
      DO 150 I=2,5
  150   YPR(I)=YPR(I-1)-OBSMAX
C
C        Calculate positioning of points on probability scale
C
      X=50.0/(2.0*FLOAT(N))
      P(1)=50.0+X
      DO 160 K=2,N
  160   P(K)=P(K-1)+2.0*X
      DO 200 K=1,N
        W=0.02*P(K)-1.0
        IF (P(K).GT.95.5) GO TO 180
        W2=W*W
        D(K)=A(1)
        DO 170 I=2,6
  170     D(K)=D(K)*W2+A(I)
        D(K)=D(K)*W
        GO TO 200
  180   Z=LOG(1.0-W)
        D(K)=B(1)
        DO 190 I=2,4
  190     D(K)=D(K)*Z+B(I)
  200 CONTINUE
      WRITE (IWRITE,10) (NTITLE(I),I=1,20)
      L=N
      QR=YSCALE/1000.0
C
C        Print output line by line, except for the last one, with the
C        scale value printed on every tenth line
C
      DO 280 I=1,50
C
C        Calculate position on vertical scale, and which points - of
C        those not yet printed - exceed this value
C
        PR=YPR(1)-FLOAT(I-1)*YSCALE-QR
        IF (OBS(L).LT.PR) GO TO 260
C
C        Set output array to blanks
C
        DO 210 IX=2,101
  210     OUT(IX)=BLANK
C
C        Calculate position of point on probability scale and put print
C        symbol in appropriate array element.  Repeat for further points
C        at this value, and print line of output
C
  220   JP=(D(L)/XSCALE)+1.0
        OUT(JP)=PLUS
        L=L-1
        IF (L.EQ.0) GO TO 230
        IF (OBS(L).GE.PR) GO TO 220
  230   IF (MOD(I-1,10).NE.0) GO TO 240
        I1=((I-1)/10)+1
        WRITE (IWRITE,20) YPR(I1),(OUT(I2),I2=1,101)
```

```
        GO TO 250
 240    WRITE (IWRITE,70) (OUT(I2),I2=1,101)
C
C        If all points have been printed, or points remain after all
C        lines have been printed (except the last) leave the main loop
C        and complete the printing elsewhere
C
 250    IF (L.GT.0) GO TO 280
        IF (I.GE.50) GO TO 320
        IL=I+1
        GO TO 290
 260    IF (MOD(I-1,10).NE.0) GO TO 270
        I1=((I-1)/10)+1
        WRITE (IWRITE,80) YPR(I1)
        GO TO 280
 270    WRITE (IWRITE,30)
 280  CONTINUE
      GO TO 320
C
C        Complete printing when all points have already been output
C
 290  DO 310 I=IL,50
        IF (MOD(I-1,10).NE.0) GO TO 300
        I1=((I-1)/10)+1
        WRITE (IWRITE,80) YPR(I1)
        GO TO 310
 300    WRITE (IWRITE,30)
 310  CONTINUE
 320  IF (L.LE.0) GO TO 350
C
C        Print last line, including the probability axis and any
C        remaining points
C
      DO 330 IX=2,101
 330    OUT(IX)=OUT(1)
 340  JP=(D(L)/XSCALE)+1.0
      OUT(JP)=PLUS
      L=L-1
      IF (L.GT.0) GO TO 340
      WRITE (IWRITE,20) YPR(6),(OUT(I2),I2=1,101)
      GO TO 360
 350  WRITE (IWRITE,40) YPR(6)
C
C        Print probability scale
C
 360  WRITE (IWRITE,50)
      WRITE (IWRITE,60) (XPR(IP),IP=1,10)
      RETURN
      END
```

```
      double precision function gamain (x,p,g,ifault)
      implicit double precision (a-h,o-z)
c
c        Algorithm AS 32 J.R. Statist. Soc. C.  (1970) Vol.19 No. 3
c        Algorithm AS 239 is recommended as an alternative.
c
c        Computes incomplete gamma ratio for positive values of
c        arguments x and p.  G must be calculated and should be equal
c        to ln(gamma(p)).   Algorithm AS 245 may be used for this.
c        ifault=1 if p.le.0 else 2 if x.lt.0 else 0.
c        Uses series expansion if p.gt.x or x.le.1, otherwise
c        continued fraction approximation.
c
c        Revised to incorporate the recommendations of Rice & Das,
c        Appl. Statist., 34, 326, 1985, and of Cran, Appl. Statist.,
c        38, 423, 1989.
c
      dimension pn(6)
      data zero/0.0d0/, one/1.0d0/, uflo/1.0d-37/, two/2.0d0/
      data acu/1.0d-8/,oflo/1.0d37/
c
      g=alngam(p)
c
c        Define accuracy and initialize
c
      gin=zero
      ifault=0
c
c        Test for admissibility of arguments
c
      if (p.le.zero) ifault=1
      if(x.lt.zero) ifault=2
      if (ifault .gt. 0 .or. x .eq. zero) go to 50
      arg=p*log(x)-x-g
      if(arg.lt.log(uflo)) then
        ifault=3
        go to 50
      end if
      factor=exp(arg)
      if(x.gt.one .and. x.ge.p) goto 30
c
c        Calculation by series expansion
c
      gin=one
      term=one
      rn=p
   20 rn=rn+one
      term=term*x/rn
      gin=gin+term
      if(term.gt.acu) goto 20
      gin=gin*factor/p
      goto 50
c
c        Calculation by continued fraction
c
   30 a=one-p
      b=a+x+one
      term=zero
      pn(1)=one
      pn(2)=x
      pn(3)=x+one
```

```
      pn(4)=x*b
      gin=pn(3)/pn(4)
c
   32 a=a+one
      b=b+two
      term=term+one
      an=a*term
      do 33 i=1,2
   33 pn(i+4)=b*pn(i+2)-an*pn(i)
      if(pn(6).eq.zero) goto 35
      rn=pn(5)/pn(6)
      dif=abs(gin-rn)
      if(dif.gt.acu) goto 34
      if(dif.le.acu*rn) goto 42
   34 gin=rn
   35 do 36 i=1,4
   36 pn(i)=pn(i+2)
      if(abs(pn(5)).lt.oflo) goto 32
      do 41 i=1,4
   41 pn(i)=pn(i)/oflo
      goto 32
   42 gin=one-factor*gin
   50 gamain=gin
      return
      end
```

```
      SUBROUTINE BANINV(N, K, SIGMA, S, G)
C
C     ALGORITHM AS 34 APPL. STATIST. (1970) VOL.19, NO.3
C
C     Update inverse of symmetric banded matrices
C
      INTEGER N, K
      DOUBLE PRECISION SIGMA(N), S(N), G(N)
C
C     Local variables
C
      INTEGER I, J, INDX, M, JJ, MM, NN
      DOUBLE PRECISION B, C, CG, U, V, ZERO, ONE
      DATA ZERO /0.D0/, ONE /1.D0/
C
C     Form C
C
      C = S(1)
      I = N
      DO 1 J = 2, K
        C = C - S(J) * G(I)
        I = I - 1
    1 CONTINUE
      C = ONE / C
C
C     Form SIGMA at N+1
C
      INDX = 0
      M = N * (N+1) / 2
      DO 2 J = 1, N
        CG = C * G(J)
        M = M + 1
        SIGMA(M) = -CG
        DO 2 JJ = 1, J
          INDX = INDX + 1
          SIGMA(INDX) = SIGMA(INDX) + CG * G(JJ)
    2 CONTINUE
      SIGMA(M+1) = C
C
C     Form G at N+1
C
      B = ZERO
      DO 4 J = 2, K
    4 B = B + S(J) * G(J-1)
      NN = N / 2
      M = N - 2 * NN
      B = B * C
      U = G(1)
      G(1) = -B
      DO 5 J = 1, NN
        MM = N - J + 1
        G(MM+1) = G(MM) + B * U
        V = G(J+1)
        G(J+1) = U + B * G(MM)
        U = V
    5 CONTINUE
      IF (M .EQ. 0) GO TO 7
      G(NN+2) = U + B * U
    7 RETURN
      END
```

```
      SUBROUTINE NEXTGJ(A, IA, K, IWK)
C
C     A FORTRAN translation of Garside's algorithm AS38, but without
C     checks and with only the upper triangle of the SSP-matrix stored.
C     Array IWK must contain (K-1) zeroes initially.
C
      DOUBLE PRECISION A(IA)
      INTEGER IWK(K)
C
C     Local variables
C
      DOUBLE PRECISION ONE, AA, AIP
      DATA ONE/1.D0/
C
      IP = K-1
C
C     IREM = 1 if a variable is being removed from the model,
C          = 0 if it is being added.
C
   10 IREM = IWK(IP)
C
C     Pivot variable no. IP in or out of the model.
C
      IPP = (IP-1)*(K+K+2-IP)/2 + 1
      IJ = IPP+K+1-IP
      AA = -A(IPP)
      A(IPP) = AA
      AA = ONE/AA
      IP1 = IP+1
      DO 30 I = IP1, K
        IPP = IPP+1
        AIP = A(IPP)*AA
        IPJ = IPP
        DO 20 J = I, K
          A(IJ) = A(IJ) + AIP*A(IPJ)
          IJ = IJ+1
          IPJ = IPJ+1
   20   CONTINUE
   30 CONTINUE
C
C     Change the indicator in array IWK for variable IP.
C     If variable was deleted, move onto the next lower numbered
C     variable, otherwise return.
C
      IWK(IP) = 1-IREM
      IF(IREM.EQ.0) RETURN
      IP = IP-1
      IF(IP.GT.0) GO TO 10
C
      RETURN
      END
```

```
      SUBROUTINE MSTUPD(KN, KI, DK, DIST, NUM, IREF, IPUSH)
C
C     ALGORITHM AS 40  APPL. STATIST. (1971) VOL.20, NO.2
C
C     This subroutine updates the minimal spanning tree of KN points,
C     defined by KI(J) being the point nearest to J on the chain leading
C     from J to the point KN and DK(J) being the length link J to KI(J),
C     with the new point (KN+1) whose distances from the points of the
C     original M.S.T. are given in the array DIST.
C     The update version of the M.S.T. is left in the arrays KI and DK
C     suitable for re-entry to the subroutine with the next point.
C     A working space of three arrays NUM, IREF and IPUSH, each at least
C     of size KN, is required.
C
      INTEGER KN, KI(KN), NUM(KN), IREF(KN), IPUSH(KN)
      REAL DK(KN), DIST(KN)
C
      KT = KN - 1
      IP = 0
C
C     If J = IREF(I) > 0 it refers to link J to KI(J), while if < 0 it
C     refers to the link from point (KN+1) to (-J).
C
      DO 1 I = 1, KN
        IREF(I) = -I
    1 CONTINUE
C
C     NUM(I) holds the number of links pointing to I.   Hence I is an end
C     point if NUM(I) = 0.
C
      DO 2 I = 1, KT
        K = KI(I)
        NUM(K) = NUM(K) + 1
    2 CONTINUE
C
C     Start of algorithm
C     NUM(I) = 1 if the link I to KI(I) is included in the new M.S.T.
C     otherwise it is 0 or < 0.
C
      DO 3 NI = 1, KT
        IF (NUM(NI) .NE. 0) GO TO 3
        I = NI
   31   J = KI(I)
        IF (DIST(I) .GT. DK(I)) GO TO 34
        IF (DK(I) .GE. DIST(J)) GO TO 32
        IREF(J) = I
        DIST(J) = DK(I)
   32   IF (IREF(I) .GT. 0) GO TO 33
        IP = IP + 1
        IPUSH(IP) = -IREF(I)
        GO TO 36
   33   K = IREF(I)
        NUM(K) = 1
        GO TO 36
   34   IF (DIST(I) .GE. DIST(J)) GO TO 35
        IREF(J) = IREF(I)
        DIST(J) = DIST(I)
   35   NUM(I) = 1
   36   NUM(J) = NUM(J) - 1
        IF (NUM(J) .NE. 0) GO TO 3
        I = J
```

```
            NUM(I) = -1
            GO TO 31
        3 CONTINUE
C
          K = -IREF(KN)
          IF (K .GE. 0) GO TO 42
          K = -K
          NUM(K) = 1
C
C     KI is updated so that all the chains point towards (KN+1).
C
       41 K = IPUSH(IP)
          IP = IP - 1
       42 KT = KN + 1
          X = DIST(K)
       43 KP = KI(K)
          KI(K) = KT
          Y = DK(K)
          DK(K) = X
          X = Y
          KT = K
          K = KP
          IF (NUM(KT) .GT. 0) GO TO 43
          IF (IP .NE. 0) GO TO 41
C
          RETURN
          END
```

```
      subroutine dssp(x,xmean,xssp,wt,sumwt,nvar,nunit,ifault)
c
c        Algorithm AS 41 j.r.statist.soc.c. (1971) vol. 20 no.2
c
c          This subroutine updates the mean vector xmean (length nvar)
c          and the matrix of corrected sums of squares and products xssp
c          (length nvar(nvar+1)/2, stored by lower triangle), when a
c          data vector x (length nvar) with weight wt is either included
c          (wt.gt.0) or excluded (wt.lt.0).  sumwt is the current sum of
c          weights on entry and the updated sum on exit and nunit is
c          the current and updated sample size.  ifault=0 indicates normal
c          exit,  ifault=1 indicates zero or negative value of sumwt,
c          ifault=2 indicates zero or negative nunit, ifault=3 indicates
c          nvar.lt.1.  Note that x, xmean, xssp, wt and sumwt are double
c          precision and must be declared as such in the calling program.
c
      dimension x(*), xmean(*), xssp(*)
      double precision x, xmean, xssp, wt, sumwt, b, c, co
      data co/0.0d0/
c
c          Check variates, weights and sample size
c
      ifault = 0
      if(nvar.lt.1) goto 103
      if(wt)107,100,106
  107 nunit = nunit-1
      go to 105
  106 nunit = nunit+1
  105 sumwt = sumwt+wt
      if(sumwt.le.co) go to 101
      k = 0
      b = wt/sumwt
      if(nunit-1)102,120,110
c
c          Update means and ssp for sample size greater than 1
c
  110 c = wt-b*wt
      do 111 i = 1,nvar
        x(i) = x(i)-xmean(i)
        xmean(i) = xmean(i)+b*x(i)
        do 111 j = 1,i
          k = k+1
          xssp(k) = xssp(k)+c*x(i)*x(j)
  111 continue
      return
c
c          Initialise means and ssp for sample size  =  1
c
  120 do 121 i = 1,nvar
        if(wt.lt.co) goto 122
        xmean(i) = x(i)
        goto 123
  122   xmean(i) = xmean(i)+b*(x(i)-xmean(i))
  123   x(i) = co
        do 121 j = 1,i
          k = k+1
          xssp(k) = co
  121 continue
      return
c
c          Set fault indicators
```

```
c
  103 ifault = ifault+1
  102 ifault = ifault+1
  101 ifault = ifault+1
  100 return
      end
```

```
CSTART OF AS 45
      SUBROUTINE HISTGM(IWRITE, IWIDTH, FREQ, DAT, NN, MM,
     $  LENG, IND, IFAULT)
C
C        ALGORITHM AS 45  APPL. STATIST. (1971) VOL.20, P.332
C
C        GIVEN A VECTOR OF FREQUENCIES, OR A VECTOR OF RAW DATA,
C        A HISTOGRAM IS PLOTTED SHOWING THE FREQUENCY DISTRIBUTION
C
      DIMENSION IOUT(28), OUT(28), DAT(NN), FREQ(MM)
      INTEGER FREQ, SCALE
      LOGICAL IND
C
C        DEFINE CHARACTERS FOR PRINTING, AND MAGNITUDE OF
C        SMALLEST ACCEPTABLE NUMBER.
C
      DATA IBLANK, ISTAR, IDASH /1H , 1H*, 1H-/
      DATA ETA /1.0E-38/
C
    1 FORMAT(6H EACH , A1, 8H EQUALS , I4, 7H POINTS /)
    2 FORMAT(1H , I8, 3X, 28(4X, A1))
    3 FORMAT(12H INTERVAL  ), 14(F8.3, 2X))
    4 FORMAT(12H MID-POINTS), 5X, 14(F8.3, 2X))
    5 FORMAT(24H0THE PRINTED VALUES MUST/ 22H BE MULTIPLIED BY 10**, I3)
    6 FORMAT(12H0FREQUENCY  , 28I5)
    7 FORMAT(1H , 120A1)
    8 FORMAT(16H CELLS COMBINED,, I3, 3H AT / 21H A TIME, TO FIT WIDTH)
C
C        CHECK INPUT PARAMETERS AND TYPE OF INPUT
C
      IFAULT = 0
      IF (MM .GT. 28 .OR. IWIDTH .LT. 26) IFAULT = 1
      M = MM
      N = NN
      K = (IWIDTH - 15) / 5
      IF (M .GT. K .AND. IND) M = K
      LENGTH = LENG - 10
      IF (LENGTH .LE. 0) IFAULT = 2
      IF (IFAULT .NE. 0) RETURN
      FM = M
      IF (IND) GOTO 15
      IF (M .LE. K) GOTO 120
      KEY = M / K
      IF (K * KEY .NE. M) KEY = KEY + 1
      L = 1
      DO 10 I = 1, K
      IOUT(I) = 0
      DO 10 J = 1, KEY
      IOUT(I) = IOUT(I) + FREQ(L)
      L = L + 1
      IF (L .GT. M) GOTO 12
   10 CONTINUE
   12 M = I
      DO 14 I = 1, M
   14 FREQ(I) = IOUT(I)
      WRITE (IWRITE, 8) KEY
      GOTO 120
C
C        DEFINE A SUITABLE SCALE
C
   15 DO 20 I = 1, M
```

```
     20 FREQ(I) = 0
        XMIN = DAT(1)
        XMAX = XMIN
        DO 30 I = 2, N
        DT = DAT(I)
        IF (DT .LT. XMIN) XMIN = DT
        IF (DT .GT. XMAX) XMAX = DT
     30 CONTINUE
        IF (XMAX - XMIN .GE. ETA * FM) GOTO 35
        IFAULT = 3
        RETURN
     35 KEY = 1
        KOUNT = 0
     40 R = XMAX - XMIN
        B = XMIN
     50 IF (R .GT. 1.0) GOTO 60
        KOUNT = KOUNT + 1
        R = R * 10.0
        GOTO 50
     60 IF (R .LE. 10.0) GOTO 70
        KOUNT = KOUNT - 1
        R = R / 10.0
        GOTO 60
     70 IF (KEY .GT. 2) GOTO 80
        TK = 10.0 ** KOUNT
        B = B * TK
        IF (B .LT. 0.0 .AND. B .NE. AINT(B)) B = B - 1.0
        B = AINT(B) / TK
        R = (XMAX - B) / FM
        KOUNT = 0
        KEY = KEY + 2
        GOTO 50
     80 STEP = AINT(R)
        IF (STEP .NE. R) STEP = STEP + 1.0
        IF (R .LT. 1.5) STEP = STEP - 0.5
        STEP = STEP / 10.0 ** KOUNT
        IF (KEY .EQ. 4) GOTO 90
        IF (XMAX - XMIN .GT. 0.8 * FM * STEP) GOTO 90
        KOUNT = 1
        KEY = 2
        GOTO 40
     90 XMIN = B
        C = STEP * AINT(B / STEP)
        IF (C .LT. 0.0 .AND. C .NE. B) C = C - STEP
        IF (C + FM * STEP .GE. XMAX) XMIN = C
C
C          CALCULATE FREQUENCIES FOR EACH INTERVAL
C
        DO 110 I = 1, N
        J = (DAT(I) - XMIN) / STEP + 1.0
        FREQ(J) = FREQ(J) + 1
    110 CONTINUE
C
C          PRINT FREQUENCY VECTOR
C
    120 WRITE (IWRITE, 6) (FREQ(I), I = 1, M)
        LINE = M * 5 + 15
        WRITE (IWRITE, 7) (IDASH, I = 1, LINE)
C
C          FIND LARGEST FREQUENCY AND SCALE IF NECESSARY
C
```

```
      MAX = 0
      DO 130 I = 1, M
      IF (FREQ(I) .GT. MAX) MAX = FREQ(I)
  130 CONTINUE
      SCALE = 1
      DIV = 1.0
      IF (MAX .LE. LENGTH) GOTO 140
      SCALE = (MAX + LENGTH - 1) / LENGTH
      WRITE (IWRITE, 1) ISTAR, SCALE
      DIV = 1.0 / FLOAT(SCALE)
C
C        CLEAR OUTPUT TO BLANKS
C
  140 DO 150 I = 1, M
  150 IOUT(I) = IBLANK
C
C        FOR EACH LINE OF PRINT, PLACE OUTPUT CHARACTERS IN
C        THEIR APPROPRIATE POSITIONS IN THE OUTPUT VECTOR
C
      MAX = FLOAT(MAX) * DIV + 0.5
      DO 170 I = 1, MAX
      K = MAX + 1 - I
      DO 160 J = 1, M
      INDEX = FLOAT(FREQ(J)) * DIV + 0.5
      IF (INDEX .EQ. K) IOUT(J) = ISTAR
  160 CONTINUE
      L = K * SCALE
C
C        PRINT LINE OF FREQUENCIES
C
      WRITE (IWRITE, 2) L, (IOUT(J), J = 1, M)
  170 CONTINUE
      WRITE (IWRITE, 7) (IDASH, I = 1, LINE)
      IF (.NOT. IND) RETURN
C
C        COMPUTE INTERVAL MID-POINTS AND SCALE IF NECESSARY
C
      K = 0
      XMIN = XMIN + STEP * 0.5
      XMAX = XMIN + STEP * FLOAT(M - 1)
      XM = AMIN1(ABS(XMIN), ABS(XMAX))
      IF (XM .LT. ETA) XM = XM + STEP
  180 IF (XM .GE. 0.1) GOTO 190
      K = K + 1
      XM = XM * 10.0
      GOTO 180
  190 XM = AMAX1(XMAX, -XMIN)
  200 IF (XM .LT. 1000.0) GOTO 210
      K = K - 1
      XM = XM / 10.0
      GOTO 200
  210 TK = 10.0 ** K
      STEP = STEP * TK
      OUT(1) = XMIN * TK
      DO 220 I = 2, M
  220 OUT(I) = OUT(I - 1) + STEP
C
C        PRINT INTERVAL MID-POINTS
C
      WRITE (IWRITE, 3) (OUT(J), J = 1, M, 2)
      WRITE (IWRITE, 4) (OUT(J), J = 2, M, 2)
```

```
      K = -K
      IF (K .NE. 0) WRITE (IWRITE, 5) K
      RETURN
      END
CEND OF AS 45
```

```
      subroutine gmsmod (x,n,m,v,c,d,nullty)
c
c        Algorithm AS 46 Applied Statistics (J.R.Statist.Soc C),
c       (1971) Vol.20, No.3
c
c       Expresses the n*m matrix x as the product of the n*m matrix v
c       and the m*m upper unit triangular matrix c.  v-transpose.v is
c       a diagonal matrix and is returned in d.  nullty is the nullity
c       of d.  v may coincide with x.
c
        double precision x(*),v(*),c(*),d(*)
        double precision w0, w1, w2, w3, eta
        double precision test, zero, one
c
      data eta/1.0d-12/, zero/0.0d0/, one/1.0d0/
c
      nullty=0
      ic=0
      i1=-n
      do 70 icol=1,m
        i1=i1+n
        w0=zero
        i2=i1
        do 10 irow=1,n
          i2=i2+1
          w1=x(i2)
          v(i2)=w1
   10   w0=w0+w1*w1
        i3=0
        i4=0
        do 70 icolw=1,icol
          i2=i1
          w2=zero
          do 20 irow=1,n
            i2=i2+1
            i3=i3+1
            w3=v(i2)
            w1=v(i3)
   20     w2=w2+w1*w3
          if(icolw.lt.icol) goto 30
          test=w2/w0
          if(test.ge.eta) goto 25
          do 21 irow=1,n
            i4=i4+1
   21     v(i4)=zero
          i4=i4-n
          nullty=nullty+1
          w2=zero
   25     d(icol)=w2
          i4=i4+n
          w1=one
          goto 60
   30     w1=zero
          w3=d(icolw)
          if(w3.eq.zero) goto 40
          w1=w2/w3
   40     i2=i1
          do 50 irow=1,n
            i2=i2+1
            i4=i4+1
   50     v(i2)=v(i2)-w1*v(i4)
```

```
60     ic=ic+1
       c(ic)=w1
70  continue
    return
    end
```

```
c This file contains two versions of the Nelder & Mead simplex algorithm
c for function minimization.   The first is that published in the journal
c of Applied Statistics.   This does not include the fitting of a quadratic
c surface, which provides the only satisfactory method of testing whether
c a minimum has been found.   The search for a minimum is liable to
c premature termination.
c The second version is one which has been developed jointly by CSIRO and
c Rothamsted, and does include the quadratic-surface fitting.
c
c
      subroutine nelmin(fn, n, start, xmin, ynewlo, reqmin, step,
     #     konvge, kcount, icount, numres, ifault)
      implicit double precision (a-h,o-z)
c
c     Simplex function minimisation procedure due to Nelder+Mead(1965),
c     as implemented by O'Neill(1971, Appl.Statist. 20, 338-45), with
c     subsequent comments by Chambers+Ertel(1974, 23, 250-1), Benyon(1976,
c     25, 97) and Hill(1978, 27, 380-2)
c
c        Algorithm AS 47  Applied Statistics (J.R. Statist. Soc. C),
c        (1971) Vol.20, No. 3
c
c     The Nelder-Mead Simplex Minimisation Procedure
c
c
c        Purpose :: To find the minimum value of a user-specified
c                   function
c
c        Formal parameters ::
c
c            fn :        : The name of the function to be minimized.
c             n :  input : The number of variables over which we are
c                        : minimising
c          start :  input : Array; Contains the coordinates of the
c                           starting point.
c                  output : The values may be overwritten.
c           xmin : output : Array; Contains the coordinates of the
c                        : minimum.
c        ynewlo : output : The minimum value of the function.
c        reqmin :  input : The terminating limit for the variance of
c                        : function values.
c           step :  input : Array; Determines the size and shape of the
c                        : initial simplex.  The relative magnitudes of
c                        : its n elements should reflect the units of
c                        : the n variables.
c        konvge :  input : The convergence check is carried out every
c                        : konvge iterations.
c        kcount :  input : Maximum number of function evaluations.
c        icount : output : Function evaluations performed
c        numres : output : Number of restarts.
c        ifault : output : 1 if reqmin, n, or konvge has illegal value;
c                        : 2 if terminated because kcount was exceeded
c                        :   without convergence;
c                        : 0 otherwise.
c
c        All variables and arrays are to be declared in the calling
c        program as double precision.
c
c
c        Auxiliary algorithm :: The double precision function
c        subprogram fn(a) calculates the function value at point a.
```

```
c        a is double precision with n elements.
c
c
c        Reference :: Nelder,J.A. and Mead,R.(1965).  A simplex method
c        for function minimization.  Computer J., Vol.7,308-313.
c
c*************************************************************************
c
      double precision start(n), xmin(n), ynewlo, reqmin, step(n),
     1   p(20,21), pstar(20), p2star(20), pbar(20), y(21),
     2   dn, dnn, z, ylo, rcoeff, ystar, ecoeff, y2star, ccoeff,
     3   rq, x, del, fn, one, half, zero, eps
      external fn
c
      data rcoeff/1.0d0/, ecoeff/2.0d0/, ccoeff/5.0d-1/
      data one/1.0d0/, half/0.5d0/, zero/0.0d0/, eps/0.001d0/
c        reflection, extension and contraction coefficients.
c
c       validity checks on input parameters.
c
      ifault=1
      if(reqmin .le. zero .or. n .lt. 1 .or. n .gt. 20
     #   .or. konvge .lt. 1) return
      ifault=2
      icount=0
      numres=0
c
      jcount=konvge
      dn=float(n)
      nn=n+1
      dnn=float(nn)
      del=one
      rq=reqmin*dn
c
c        construction of initial simplex.
c
   10 do 20 i=1,n
   20 p(i,nn)=start(i)
      y(nn)=fn(start)
      do 40 j=1,n
        x=start(j)
        start(j)=start(j)+step(j)*del
        do 30 i=1,n
   30   p(i,j)=start(i)
        y(j)=fn(start)
        start(j)=x
   40 continue
      icount=icount+nn
c
c       simplex construction complete
c
c       find highest and lowest y values.  ynewlo (=y(ihi) ) indicates
c       the vertex of the simplex to be replaced.
c
   43 ylo=y(1)
      ilo=1
      do 47 i=2,nn
        if(y(i).ge.ylo) goto 47
        ylo=y(i)
        ilo=i
   47 continue
```

```
     50 ynewlo=y(1)
        ihi=1
        do 70 i=2,nn
          if(y(i) .le. ynewlo) goto 70
          ynewlo=y(i)
          ihi=i
     70 continue
c
c      calculate pbar,the centroid of the simplex vertices
c          excepting that with y value ynewlo.
c
        do 90 i=1,n
          z=zero
          do 80 j=1,nn
     80   z=z+p(i,j)
          z=z-p(i,ihi)
          pbar(i)=z/dn
     90 continue
c
c      reflection through the centroid
c
        do 100 i=1,n
    100 pstar(i)=pbar(i) + rcoeff * (pbar(i) - p(i,ihi))
        ystar=fn(pstar)
        icount=icount+1
        if (ystar.ge.ylo) goto 140
c
c      successful reflection,so extension
c
        do 110 i=1,n
    110 p2star(i)=pbar(i) + ecoeff * (pstar(i)-pbar(i))
        y2star=fn(p2star)
        icount=icount+1
c
c       check extension
c
        if(y2star .ge. ystar) goto 133
c
c       retain extension or contraction.
c
        do 130 i=1,n
    130 p(i,ihi)=p2star(i)
        y(ihi)=y2star
        goto 230
c
c     retain reflection
c
    133 do 137 i=1,n
    137 p(i,ihi)=pstar(i)
        y(ihi)=ystar
        goto 230
c
c     no extension
c
    140 l=0
        do 150 i=1,nn
          if (y( i).gt.ystar) l=l+1
    150 continue
        if (l.gt.1) goto 133
        if (l.eq.0) goto 170
c
```

```
c     contraction on the reflection side of the centroid.
c
      do 160 i=1,n
  160 p2star(i) = pbar(i) + ccoeff * (pstar(i) - pbar(i))
      y2star = fn(p2star)
      icount=icount+1
      if(y2star .le. ystar) goto 182
c
c        retain reflection
c
      do 165 i=1,n
  165 p(i,ihi)=pstar(i)
      y(ihi)=ystar
      goto 230
c
c      contraction on the  y(ihi) side of the centriod.
c
  170 do 180 i=1,n
  180 p2star(i)=pbar(i) + ccoeff * (p(i,ihi) - pbar(i))
      y2star=fn(p2star)
      icount=icount+1
      if (y2star .gt. y(ihi)) goto 188
c
c        retain contraction
c
  182 do 185 i=1,n
  185 p(i,ihi) = p2star(i)
      y(ihi)=y2star
      goto 230
c
c       contract whole simplex.
c
  188 do 200 j=1,nn
        do 190 i=1,n
          p(i,j)=(p(i,j)+p(i,ilo))*half
          xmin(i)=p(i,j)
  190   continue
        y(j)=fn(xmin)
  200 continue
      icount=icount+nn
      if (icount .gt. kcount) go to 260
      goto 43
c
c        Check if ylo improved
c
  230 if (y(ihi) .ge. ylo) goto 235
      ylo=y(ihi)
      ilo=ihi
  235 jcount=jcount-1
      if(jcount .ne. 0) goto 50
c
c     check to see if minimum reached.
c
      if (icount.gt.kcount) goto 260
      jcount=konvge
      z=zero
      do 240 i=1, nn
  240 z = z+y(i)
      x=z / dnn
      z=zero
      do 250 i=1,nn
```

```
  250 z = z+(y(i)-x) ** 2
      if (z .gt. rq) goto 50
c
c       factorial tests to check that ynewlo is a local minimum.
c
  260 do 270 i=1,n
  270 xmin(i)=p(i,ilo)
      ynewlo=y(ilo)
      if (icount.gt.kcount) return
      do 280 i=1,n
        del=step(i)*eps
        xmin(i)=xmin(i)+del
        z=fn(xmin)
        icount=icount+1
        if (z.lt.ynewlo) goto 290
        xmin(i)=xmin(i)-del-del
        z=fn(xmin)
        icount=icount+1
        if (z.lt.ynewlo) goto 290
        xmin(i)=xmin(i)+del
  280 continue
      ifault = 0
      return
c
c     restart procedure
c
  290 do 300 i=1,n
  300 start(i) = xmin(i)
      del=eps
      numres = numres + 1
      goto 10
      end
c
c-----------------------------------------------------------------------
c
      SUBROUTINE MINIM(P,STEP,NOP,FUNC,MAX,IPRINT,STOPCR,NLOOP,IQUAD,
     1  SIMP,VAR,FUNCTN,IFAULT)
C
C     A PROGRAM FOR FUNCTION MINIMIZATION USING THE SIMPLEX METHOD.
C     The minimum found will often be a local, not a global, minimum.
C
C     FOR DETAILS, SEE NELDER & MEAD, THE COMPUTER JOURNAL, JANUARY 1965
C
C     PROGRAMMED BY D.E.SHAW,
C     CSIRO, DIVISION OF MATHEMATICS & STATISTICS
C     P.O. BOX 218, LINDFIELD, N.S.W. 2070
C
C     WITH AMENDMENTS BY R.W.M.WEDDERBURN
C     ROTHAMSTED EXPERIMENTAL STATION
C     HARPENDEN, HERTFORDSHIRE, ENGLAND
C
C     Further amended by Alan Miller,
C     CSIRO, Division of Mathematics & Statistics
C     Private Bag 10, CLAYTON, VIC. 3168
C
C     ARGUMENTS:-
C     P()     = INPUT, STARTING VALUES OF PARAMETERS
C               OUTPUT, FINAL VALUES OF PARAMETERS
C     STEP()  = INPUT, INITIAL STEP SIZES
C     NOP     = INPUT, NO. OF PARAMETERS, INCL. ANY TO BE HELD FIXED
C     FUNC    = OUTPUT, THE FUNCTION VALUE CORRESPONDING TO THE FINAL
```

```
C               PARAMETER VALUES
C     MAX     = INPUT, THE MAXIMUM NO. OF FUNCTION EVALUATIONS ALLOWED
C     IPRINT  = INPUT, PRINT CONTROL PARAMETER
C                    < 0 NO PRINTING
C                    = 0 PRINTING OF PARAMETER VALUES AND THE FUNCTION
C                        VALUE AFTER INITIAL EVIDENCE OF CONVERGENCE.
C                    > 0 AS FOR IPRINT = 0 PLUS PROGRESS REPORTS AFTER
C                        EVERY IPRINT EVALUATIONS, PLUS PRINTING FOR THE
C                        INITIAL SIMPLEX.
C     STOPCR  = INPUT, STOPPING CRITERION
C     NLOOP   = INPUT, THE STOPPING RULE IS APPLIED AFTER EVERY NLOOP
C               FUNCTION EVALUATIONS.
C     IQUAD   = INPUT, = 1 IF THE FITTING OF A QUADRATIC SURFACE IS REQUIRED
C                      = 0 IF NOT
C     SIMP    = INPUT, CRITERION FOR EXPANDING THE SIMPLEX TO OVERCOME
C               ROUNDING ERRORS BEFORE FITTING THE QUADRATIC SURFACE.
C     VAR()   = OUTPUT, CONTAINS THE DIAGONAL ELEMENTS OF THE INVERSE OF
C               THE INFORMATION MATRIX.
C     FUNCTN  = INPUT, NAME OF THE USER'S SUBROUTINE - ARGUMENTS (P,FUNC)
C               WHICH RETURNS THE FUNCTION VALUE FOR A GIVEN SET OF
C               PARAMETER VALUES IN ARRAY P.
C****   FUNCTN MUST BE DECLARED EXTERNAL IN THE CALLING PROGRAM.
C       IFAULT  = OUTPUT, = 0 FOR SUCCESSFUL TERMINATION
C                         = 1 IF MAXIMUM NO. OF FUNCTION EVALUATIONS EXCEEDED
C                         = 2 IF INFORMATION MATRIX IS NOT +VE SEMI-DEFINITE
C                         = 3 IF NOP < 1
C                         = 4 IF NLOOP < 1
C
C       Advice on usage:
C       If the function minimized can be expected to be smooth in the vicinity
C       of the minimum, users are strongly urged to use the quadratic-surface
C       fitting option.   This is the only satisfactory way of testing that the
C       minimum has been found.   The value of SIMP should be set to at least
C       1000 times the rounding error in calculating the fitted function.
C       e.g. in double precision on a micro- or mini-computer with about 16
C       decimal digit representation of floating-point numbers, the rounding
C       errors in calculating the objective function may be of the order of
C       1.E-12 say in a particular case.   A suitable value for SIMP would then
C       be 1.E-08.   However, if numerical integration is required in the
C       calculation of the objective function, it may only be accurate to say
C       1.E-05 and an appropriate value for SIMP would be about 0.1.
C       If the fitted quadratic surface is not +ve definite (and the function
C       should be smooth in the vicinity of the minimum), it probably means
C       that the search terminated prematurely and you have not found the
C       minimum.
C
C       N.B. P, STEP AND VAR (IF IQUAD = 1) MUST HAVE DIMENSION AT LEAST NOP
C            IN THE CALLING PROGRAM.
C       THE DIMENSIONS BELOW ARE FOR A MAXIMUM OF 20 PARAMETERS.
C      The dimension of BMAT should be at least NOP*(NOP+1)/2.
C
C****      N.B. This version is in DOUBLE PRECISION throughout
C
C       LATEST REVISION - 11 August 1991
C
C*************************************************************************
C
      implicit double precision (a-h, o-z)
      external FUNCTN
      DIMENSION P(NOP),STEP(NOP),VAR(NOP)
      DIMENSION G(21,20),H(21),PBAR(20),PSTAR(20),PSTST(20),AVAL(20),
```

```
     1  BMAT(210),PMIN(20),VC(210),TEMP(20)
      DATA ZERO/0.D0/, ONE/1.D0/, TWO/2.D0/, THREE/3.D0/, HALF/0.5D0/
C
C     A = REFLECTION COEFFICIENT, B = CONTRACTION COEFFICIENT, AND
C     C = EXPANSION COEFFICIENT.
C
      DATA A,B,C/1.D0, 0.5D0, 2.D0/
C
C     SET LOUT = LOGICAL UNIT NO. FOR OUTPUT
C
      DATA LOUT/6/
C
C     IF PROGRESS REPORTS HAVE BEEN REQUESTED, PRINT HEADING
C
      IF(IPRINT.GT.0) WRITE(LOUT,1000) IPRINT
 1000 FORMAT(' PROGRESS REPORT EVERY',I4,' FUNCTION EVALUATIONS'/,
     1  ' EVAL.  FUNC.',15X,'PARAMETER VALUES')
C
C     CHECK INPUT ARGUMENTS
C
      IFAULT=0
      IF(NOP.LE.0) IFAULT=3
      IF(NLOOP.LE.0) IFAULT=4
      IF(IFAULT.NE.0) RETURN
C
C     SET NAP = NO. OF PARAMETERS TO BE VARIED, I.E. WITH STEP.NE.0
C
      NAP=0
      LOOP=0
      IFLAG=0
      DO 10 I=1,NOP
        IF(STEP(I).NE.ZERO) NAP=NAP+1
   10 CONTINUE
C
C     IF NAP = 0 EVALUATE FUNCTION AT THE STARTING POINT AND RETURN
C
      IF(NAP.GT.0) GO TO 30
      CALL FUNCTN(P,FUNC)
      RETURN
C
C     SET UP THE INITIAL SIMPLEX
C
   30 DO 40 I=1,NOP
   40 G(1,I)=P(I)
      IROW=2
      DO 60 I=1,NOP
        IF(STEP(I).EQ.ZERO) GO TO 60
        DO 50 J=1,NOP
   50   G(IROW,J)=P(J)
        G(IROW,I)=P(I)+STEP(I)
        IROW=IROW+1
   60 CONTINUE
      NP1=NAP+1
      NEVAL=0
      DO 90 I=1,NP1
        DO 70 J=1,NOP
   70   P(J)=G(I,J)
        CALL FUNCTN(P,H(I))
        NEVAL=NEVAL+1
        IF(IPRINT.LE.0) GO TO 90
        WRITE(LOUT,1010) NEVAL,H(I),(P(J),J=1,NOP)
```

```
 1010   FORMAT(/I4, 2X, G12.5, 2X, 5G12.5, 3(/20X, 5G12.5))
   90 CONTINUE
C
C     START OF MAIN CYCLE.
C
C     FIND MAX. & MIN. VALUES FOR CURRENT SIMPLEX (HMAX & HMIN).
C
  100 LOOP=LOOP+1
      IMAX=1
      IMIN=1
      HMAX=H(1)
      HMIN=H(1)
      DO 120 I=2,NP1
        IF(H(I).LE.HMAX) GO TO 110
        IMAX=I
        HMAX=H(I)
        GO TO 120
  110   IF(H(I).GE.HMIN) GO TO 120
        IMIN=I
        HMIN=H(I)
  120 CONTINUE
C
C     FIND THE CENTROID OF THE VERTICES OTHER THAN P(IMAX)
C
      DO 130 I=1,NOP
  130 PBAR(I)=ZERO
      DO 150 I=1,NP1
        IF(I.EQ.IMAX) GO TO 150
        DO 140 J=1,NOP
  140   PBAR(J)=PBAR(J)+G(I,J)
  150 CONTINUE
      DO 160 J=1,NOP
      FNAP = NAP
  160 PBAR(J)=PBAR(J)/FNAP
C
C     REFLECT MAXIMUM THROUGH PBAR TO PSTAR,
C     HSTAR = FUNCTION VALUE AT PSTAR.
C
      DO 170 I=1,NOP
  170 PSTAR(I)=A*(PBAR(I)-G(IMAX,I))+PBAR(I)
      CALL FUNCTN(PSTAR,HSTAR)
      NEVAL=NEVAL+1
      IF(IPRINT.LE.0) GO TO 180
      IF(MOD(NEVAL,IPRINT).EQ.0) WRITE(LOUT,1010) NEVAL,HSTAR,
     1  (PSTAR(J),J=1,NOP)
C
C     IF HSTAR < HMIN, REFLECT PBAR THROUGH PSTAR,
C     HSTST = FUNCTION VALUE AT PSTST.
C
  180 IF(HSTAR.GE.HMIN) GO TO 220
      DO 190 I=1,NOP
  190 PSTST(I)=C*(PSTAR(I)-PBAR(I))+PBAR(I)
      CALL FUNCTN(PSTST,HSTST)
      NEVAL=NEVAL+1
      IF(IPRINT.LE.0) GO TO 200
      IF(MOD(NEVAL,IPRINT).EQ.0) WRITE(LOUT,1010) NEVAL,HSTST,
     1  (PSTST(J),J=1,NOP)
C
C     IF HSTST < HMIN REPLACE CURRENT MAXIMUM POINT BY PSTST AND
C     HMAX BY HSTST, THEN TEST FOR CONVERGENCE.
C
```

```
  200 IF(HSTST.GE.HMIN) GO TO 320
      DO 210 I=1,NOP
        IF(STEP(I).NE.ZERO) G(IMAX,I)=PSTST(I)
  210 CONTINUE
      H(IMAX)=HSTST
      GO TO 340
C
C     HSTAR IS NOT < HMIN.
C     TEST WHETHER IT IS < FUNCTION VALUE AT SOME POINT OTHER THAN
C     P(IMAX).   IF IT IS REPLACE P(IMAX) BY PSTAR & HMAX BY HSTAR.
C
  220 DO 230 I=1,NP1
        IF(I.EQ.IMAX) GO TO 230
        IF(HSTAR.LT.H(I)) GO TO 320
  230 CONTINUE
C
C     HSTAR > ALL FUNCTION VALUES EXCEPT POSSIBLY HMAX.
C     IF HSTAR <= HMAX, REPLACE P(IMAX) BY PSTAR & HMAX BY HSTAR.
C
      IF(HSTAR.GT.HMAX) GO TO 260
      DO 250 I=1,NOP
        IF(STEP(I).NE.ZERO) G(IMAX,I)=PSTAR(I)
  250 CONTINUE
      HMAX=HSTAR
      H(IMAX)=HSTAR
C
C     CONTRACTED STEP TO THE POINT PSTST,
C     HSTST = FUNCTION VALUE AT PSTST.
C
  260 DO 270 I=1,NOP
  270 PSTST(I)=B*G(IMAX,I) + (1.d0-B)*PBAR(I)
      CALL FUNCTN(PSTST,HSTST)
      NEVAL=NEVAL+1
      IF(IPRINT.LE.0) GO TO 280
      IF(MOD(NEVAL,IPRINT).EQ.0) WRITE(LOUT,1010) NEVAL,HSTST,
     1  (PSTST(J),J=1,NOP)
C
C     IF HSTST < HMAX REPLACE P(IMAX) BY PSTST & HMAX BY HSTST.
C
  280 IF(HSTST.GT.HMAX) GO TO 300
      DO 290 I=1,NOP
        IF(STEP(I).NE.ZERO) G(IMAX,I)=PSTST(I)
  290 CONTINUE
      H(IMAX)=HSTST
      GO TO 340
C
C     HSTST > HMAX.
C     SHRINK THE SIMPLEX BY REPLACING EACH POINT, OTHER THAN THE CURRENT
C     MINIMUM, BY A POINT MID-WAY BETWEEN ITS CURRENT POSITION AND THE
C     MINIMUM.
C
  300 DO 315 I=1,NP1
        IF(I.EQ.IMIN) GO TO 315
        DO 310 J=1,NOP
          IF(STEP(J).NE.ZERO) G(I,J)=(G(I,J)+G(IMIN,J))*HALF
          P(J)=G(I,J)
  310   CONTINUE
        CALL FUNCTN(P,H(I))
        NEVAL=NEVAL+1
        IF(IPRINT.LE.0) GO TO 315
        IF(MOD(NEVAL,IPRINT).EQ.0) WRITE(LOUT,1010) NEVAL,H(I),
```

```
     1             (P(J),J=1,NOP)
  315 CONTINUE
      GO TO 340
C
C     REPLACE MAXIMUM POINT BY PSTAR & H(IMAX) BY HSTAR.
C
  320 DO 330 I=1,NOP
        IF(STEP(I).NE.ZERO) G(IMAX,I)=PSTAR(I)
  330 CONTINUE
      H(IMAX)=HSTAR
C
C     IF LOOP = NLOOP TEST FOR CONVERGENCE, OTHERWISE REPEAT MAIN CYCLE.
C
  340 IF(LOOP.LT.NLOOP) GO TO 100
C
C     CALCULATE MEAN & STANDARD DEVIATION OF FUNCTION VALUES FOR THE
C     CURRENT SIMPLEX.
C
      HSTD=ZERO
      HMEAN=ZERO
      DO 350 I=1,NP1
  350 HMEAN=HMEAN+H(I)
      FNP1 = NP1
      HMEAN=HMEAN/FNP1
      DO 360 I=1,NP1
  360 HSTD=HSTD+(H(I)-HMEAN)**2
      HSTD=SQRT(HSTD/FLOAT(NP1))
C
C     IF THE RMS > STOPCR, SET IFLAG & LOOP TO ZERO AND GO TO THE
C     START OF THE MAIN CYCLE AGAIN.
C
      IF(HSTD.LE.STOPCR.OR.NEVAL.GT.MAX) GO TO 410
      IFLAG=0
      LOOP=0
      GO TO 100
C
C     FIND THE CENTROID OF THE CURRENT SIMPLEX AND THE FUNCTION VALUE THERE.
C
  410 DO 380 I=1,NOP
        IF(STEP(I).EQ.ZERO) GO TO 380
        P(I)=ZERO
        DO 370 J=1,NP1
  370   P(I)=P(I)+G(J,I)
        FNP1 = NP1
        P(I)=P(I)/FNP1
  380 CONTINUE
      CALL FUNCTN(P,FUNC)
      NEVAL=NEVAL+1
      IF(IPRINT.LE.0) GO TO 390
      IF(MOD(NEVAL,IPRINT).EQ.0) WRITE(LOUT,1010) NEVAL,FUNC,
     1  (P(J),J=1,NOP)
C
C     TEST WHETHER THE NO. OF FUNCTION VALUES ALLOWED, MAX, HAS BEEN
C     OVERRUN; IF SO, EXIT WITH IFAULT = 1.
C
  390 IF(NEVAL.LE.MAX) GO TO 420
      IFAULT=1
      IF(IPRINT.LT.0) RETURN
      WRITE(LOUT,1020) MAX
 1020 FORMAT(' NO. OF FUNCTION EVALUATIONS EXCEEDS',I5)
      WRITE(LOUT,1030) HSTD
```

```
 1030 FORMAT(' RMS OF FUNCTION VALUES OF LAST SIMPLEX =',G14.6)
      WRITE(LOUT,1040)(P(I),I=1,NOP)
 1040 FORMAT(' CENTROID OF LAST SIMPLEX =',4(/1X,6G13.5))
      WRITE(LOUT,1050) FUNC
 1050 FORMAT(' FUNCTION VALUE AT CENTROID =',G14.6)
      RETURN
C
C     CONVERGENCE CRITERION SATISFIED.
C     IF IFLAG = 0, SET IFLAG & SAVE HMEAN.
C     IF IFLAG = 1 & CHANGE IN HMEAN <= STOPCR THEN SEARCH IS COMPLETE.
C
  420 IF(IPRINT.LT.0) GO TO 430
      WRITE(LOUT,1060)
 1060 FORMAT(/' EVIDENCE OF CONVERGENCE')
      WRITE(LOUT,1040)(P(I),I=1,NOP)
      WRITE(LOUT,1050) FUNC
  430 IF(IFLAG.GT.0) GO TO 450
      IFLAG=1
  440 SAVEMN=HMEAN
      LOOP=0
      GO TO 100
  450 IF(ABS(SAVEMN-HMEAN).GE.STOPCR) GO TO 440
      IF(IPRINT.LT.0) GO TO 460
      WRITE(LOUT,1070) NEVAL
 1070 FORMAT(//' MINIMUM FOUND AFTER',I5,' FUNCTION EVALUATIONS')
      WRITE(LOUT,1080)(P(I),I=1,NOP)
 1080 FORMAT(' MINIMUM AT',4(/1X,6G13.6))
      WRITE(LOUT,1090) FUNC
 1090 FORMAT(' FUNCTION VALUE AT MINIMUM =',G14.6)
  460 IF(IQUAD.LE.0) RETURN
C-----------------------------------------------------------------------
C
C     QUADRATIC SURFACE FITTING
C
      IF(IPRINT.GE.0) WRITE(LOUT,1110)
 1110 FORMAT(/' QUADRATIC SURFACE FITTING ABOUT SUPPOSED MINIMUM'/)
C
C     EXPAND THE FINAL SIMPLEX, IF NECESSARY, TO OVERCOME ROUNDING
C     ERRORS.
C
      NEVAL=0
      DO 490 I=1,NP1
  470   TEST=ABS(H(I)-FUNC)
        IF(TEST.GE.SIMP) GO TO 490
        DO 480 J=1,NOP
          IF(STEP(J).NE.ZERO) G(I,J)=(G(I,J)-P(J))+G(I,J)
          PSTST(J)=G(I,J)
  480   CONTINUE
        CALL FUNCTN(PSTST,H(I))
        NEVAL=NEVAL+1
        GO TO 470
  490 CONTINUE
C
C     FUNCTION VALUES ARE CALCULATED AT AN ADDITIONAL NAP POINTS.
C
      DO 510 I=1,NAP
        I1=I+1
        DO 500 J=1,NOP
  500   PSTAR(J)=(G(1,J)+G(I1,J))*HALF
        CALL FUNCTN(PSTAR,AVAL(I))
        NEVAL=NEVAL+1
```

```
  510 CONTINUE
C
C     THE MATRIX OF ESTIMATED SECOND DERIVATIVES IS CALCULATED AND ITS
C     LOWER TRIANGLE STORED IN BMAT.
C
      A0=H(1)
      DO 540 I=1,NAP
        I1=I-1
        I2=I+1
        IF(I1.LT.1) GO TO 540
        DO 530 J=1,I1
          J1=J+1
          DO 520 K=1,NOP
  520     PSTST(K)=(G(I2,K)+G(J1,K))*HALF
          CALL FUNCTN(PSTST,HSTST)
          NEVAL=NEVAL+1
          L=I*(I-1)/2+J
          BMAT(L)=TWO*(HSTST+A0-AVAL(I)-AVAL(J))
  530   CONTINUE
  540 CONTINUE
      L=0
      DO 550 I=1,NAP
        I1=I+1
        L=L+I
        BMAT(L)=TWO*(H(I1)+A0-TWO*AVAL(I))
  550 CONTINUE
C
C     THE VECTOR OF ESTIMATED FIRST DERIVATIVES IS CALCULATED AND
C     STORED IN AVAL.
C
      DO 560 I=1,NAP
        I1=I+1
        AVAL(I)=TWO*AVAL(I)-(H(I1)+THREE*A0)*HALF
  560 CONTINUE
C
C     THE MATRIX Q OF NELDER & MEAD IS CALCULATED AND STORED IN G.
C
      DO 570 I=1,NOP
  570 PMIN(I)=G(1,I)
      DO 580 I=1,NAP
        I1=I+1
        DO 580 J=1,NOP
        G(I1,J)=G(I1,J)-G(1,J)
  580 CONTINUE
      DO 590 I=1,NAP
        I1=I+1
        DO 590 J=1,NOP
          G(I,J)=G(I1,J)
  590 CONTINUE
C
C     INVERT BMAT
C
      CALL SYMINV(BMAT,NAP,BMAT,TEMP,NULLTY,IFAULT,RMAX)
      IF(IFAULT.NE.0) GO TO 600
      IRANK=NAP-NULLTY
      GO TO 610
  600 IF(IPRINT.GE.0) WRITE(LOUT,1120)
 1120 FORMAT(/' MATRIX OF ESTIMATED SECOND DERIVATIVES NOT +VE DEFN.'/
     1  ' MINIMUM PROBABLY NOT FOUND'/)
      IFAULT=2
      RETURN
```

```
C
C     BMAT*A/2 IS CALCULATED AND STORED IN H.
C
  610 DO 650 I=1,NAP
        H(I)=ZERO
        DO 640 J=1,NAP
          IF(J.GT.I) GO TO 620
          L=I*(I-1)/2+J
          GO TO 630
  620     L=J*(J-1)/2+I
  630     H(I)=H(I)+BMAT(L)*AVAL(J)
  640   CONTINUE
  650 CONTINUE
C
C     FIND THE POSITION, PMIN, & VALUE, YMIN, OF THE MINIMUM OF THE
C     QUADRATIC.
C
      YMIN=ZERO
      DO 660 I=1,NAP
  660 YMIN=YMIN+H(I)*AVAL(I)
      YMIN=A0-YMIN
      DO 670 I=1,NOP
        PSTST(I)=ZERO
        DO 670 J=1,NAP
  670 PSTST(I)=PSTST(I)+H(J)*G(J,I)
      DO 680 I=1,NOP
  680 PMIN(I)=PMIN(I)-PSTST(I)
      IF(IPRINT.LT.0) GO TO 682
      WRITE(LOUT,1130) YMIN,(PMIN(I),I=1,NOP)
 1130 FORMAT(' MINIMUM OF QUADRATIC SURFACE =',G14.6,' AT',
     1  4(/1X,6G13.5))
      WRITE(LOUT,1150)
 1150 FORMAT(' IF THIS DIFFERS BY MUCH FROM THE MINIMUM ESTIMATED',
     1  1X,'FROM THE MINIMIZATION,'/
     2  ' THE MINIMUM MAY BE FALSE &/OR THE INFORMATION MATRIX MAY BE',
     3  1X,'INACCURATE'/)
c
c     Calculate true function value at the minimum of the quadratic.
c
  682 neval = neval + 1
      call functn(pmin, hstar)
c
c     If HSTAR < FUNC, replace search minimum with quadratic minimum.
c
      if (hstar .ge. func) go to 690
      func = hstar
      do 684 i = 1, nop
  684 p(i) = pmin(i)
      write(lout, 1140) func
 1140 format(' True func. value at minimum of quadratic = ', g14.6/)
C
C     Q*BMAT*Q'/2 IS CALCULATED & ITS LOWER TRIANGLE STORED IN VC
C
  690 DO 760 I=1,NOP
        DO 730 J=1,NAP
          H(J)=ZERO
          DO 720 K=1,NAP
            IF(K.GT.J) GO TO 700
            L=J*(J-1)/2+K
            GO TO 710
  700       L=K*(K-1)/2+J
```

```
  710       H(J)=H(J)+BMAT(L)*G(K,I)*HALF
  720     CONTINUE
  730   CONTINUE
        DO 750 J=I,NOP
          L=J*(J-1)/2+I
          VC(L)=ZERO
          DO 740 K=1,NAP
  740     VC(L)=VC(L)+H(K)*G(K,J)
  750   CONTINUE
  760 CONTINUE
C
C     THE DIAGONAL ELEMENTS OF VC ARE COPIED INTO VAR.
C
      J=0
      DO 770 I=1,NOP
        J=J+I
        VAR(I)=VC(J)
  770    CONTINUE
      IF(IPRINT.LT.0) RETURN
      WRITE(LOUT,1160) IRANK
 1160 FORMAT(' RANK OF INFORMATION MATRIX =',I3/
     1  ' GENERALIZED INVERSE OF INFORMATION MATRIX:-')
      IJK=1
      GO TO 880
  790 CONTINUE
      WRITE(LOUT,1170)
 1170 FORMAT(/' IF THE FUNCTION MINIMIZED WAS -LOG(LIKELIHOOD),'/
     1  ' THIS IS THE COVARIANCE MATRIX OF THE PARAMETERS'/
     2  ' IF THE FUNCTION WAS A SUM OF SQUARES OF RESIDUALS'/
     3  ' THIS MATRIX MUST BE MULTIPLIED BY TWICE THE ESTIMATED',
     4  1X'RESIDUAL VARIANCE'/' TO OBTAIN THE COVARIANCE MATRIX.'/)
      CALL SYMINV(VC,NAP,BMAT,TEMP,NULLTY,IFAULT,RMAX)
C
C     BMAT NOW CONTAINS THE INFORMATION MATRIX
C
      WRITE(LOUT,1190)
 1190 FORMAT(' INFORMATION MATRIX:-'/)
      IJK=3
      GO TO 880
c
c     Calculate correlations of parameter estimates, put into VC.
c
  800 IJK=2
      II=0
      IJ=0
      DO 840 I=1,NOP
        II=II+I
        IF(VC(II).GT.ZERO) THEN
          VC(II)=ONE/SQRT(VC(II))
        ELSE
          VC(II)=ZERO
        END IF
        JJ=0
        DO 830 J=1,I-1
          JJ=JJ+J
          IJ=IJ+1
          VC(IJ)=VC(IJ)*VC(II)*VC(JJ)
  830   CONTINUE
        IJ=IJ+1
  840 CONTINUE
      WRITE(LOUT,1200)
```

```
 1200 FORMAT(/' CORRELATION MATRIX:-')
      II=0
      DO 850 I=1,NOP
        II=II+I
        IF(VC(II).NE.ZERO) VC(II)=ONE
  850 CONTINUE
      GO TO 880
  860 WRITE(LOUT,1210) NEVAL
 1210 FORMAT(/' A FURTHER',I4,' FUNCTION EVALUATIONS HAVE BEEN USED'/)
      RETURN
c
c     Pseudo-subroutine to print VC if IJK = 1 or 2, or
c     BMAT if IJK = 3.
c
  880 L=1
  890 IF(L.GT.NOP) GO TO (790,860,800),IJK
      II=L*(L-1)/2
      DO 910 I=L,NOP
        I1=II+L
        II=II+I
        I2=MIN(II,I1+5)
        IF(IJK.EQ.3) GO TO 900
        WRITE(LOUT,1230)(VC(J),J=I1,I2)
        GO TO 910
  900   WRITE(LOUT,1230)(BMAT(J),J=I1,I2)
  910 CONTINUE
 1230 FORMAT(1X,6G13.5)
      WRITE(LOUT,1240)
 1240 FORMAT(/)
      L=L+6
      GO TO 890
      END




      SUBROUTINE SYMINV(A,N,C,W,NULLTY,IFAULT,RMAX)
C
C     ALGORITHM AS7, APPLIED STATISTICS, VOL.17, 1968.
C
C     ARGUMENTS:-
C     A()     = INPUT, THE SYMMETRIC MATRIX TO BE INVERTED, STORED IN
C               LOWER TRIANGULAR FORM
C     N       = INPUT, ORDER OF THE MATRIX
C     C()     = OUTPUT, THE INVERSE OF A (A GENERALIZED INVERSE IF C IS
C               SINGULAR), ALSO STORED IN LOWER TRIANGULAR.
C               C AND A MAY OCCUPY THE SAME LOCATIONS.
C     W()     = WORKSPACE, DIMENSION AT LEAST N.
C     NULLTY  = OUTPUT, THE RANK DEFICIENCY OF A.
C     IFAULT  = OUTPUT, ERROR INDICATOR
C                     = 1 IF N < 1
C                     = 2 IF A IS NOT +VE SEMI-DEFINITE
C                     = 0 OTHERWISE
C     RMAX    = OUTPUT, APPROXIMATE BOUND ON THE ACCURACY OF THE DIAGONAL
C               ELEMENTS OF C.  E.G. IF RMAX = 1.E-04 THEN THE DIAGONAL
C               ELEMENTS OF C WILL BE ACCURATE TO ABOUT 4 DEC. DIGITS.
C
C     LATEST REVISION - 18 October 1985
C
C*********************************************************************
C
```

```
      implicit double precision (a-h, o-z)
      DIMENSION A(*),C(*),W(N)
      DATA ZERO/0.D0/, ONE/1.D0/
C
      NROW=N
      IFAULT=1
      IF(NROW.LE.0) GO TO 100
      IFAULT=0
C
C     CHOLESKY FACTORIZATION OF A, RESULT IN C
C
      CALL CHOLA(A,NROW,C,NULLTY,IFAULT,RMAX,W)
      IF(IFAULT.NE.0) GO TO 100
C
C     INVERT C & FORM THE PRODUCT (CINV)'*CINV, WHERE CINV IS THE INVERSE
C     OF C, ROW BY ROW STARTING WITH THE LAST ROW.
C     IROW = THE ROW NUMBER, NDIAG = LOCATION OF LAST ELEMENT IN THE ROW.
C
      NN=NROW*(NROW+1)/2
      IROW=NROW
      NDIAG=NN
   10 IF(C(NDIAG).EQ.ZERO) GO TO 60
      L=NDIAG
      DO 20 I=IROW,NROW
        W(I)=C(L)
        L=L+I
   20 CONTINUE
      ICOL=NROW
      JCOL=NN
      MDIAG=NN
   30 L=JCOL
      X=ZERO
      IF(ICOL.EQ.IROW) X=ONE/W(IROW)
      K=NROW
   40 IF(K.EQ.IROW) GO TO 50
      X=X-W(K)*C(L)
      K=K-1
      L=L-1
      IF(L.GT.MDIAG) L=L-K+1
      GO TO 40
   50 C(L)=X/W(IROW)
      IF(ICOL.EQ.IROW) GO TO 80
      MDIAG=MDIAG-ICOL
      ICOL=ICOL-1
      JCOL=JCOL-1
      GO TO 30
C
C     Special case, zero diagonal element.
C
   60 L=NDIAG
      DO 70 J=IROW,NROW
        C(L)=ZERO
        L=L+J
   70 CONTINUE
C
C      End of row.
C
   80 NDIAG=NDIAG-IROW
      IROW=IROW-1
      IF(IROW.NE.0) GO TO 10
  100 RETURN
```

```
      END




      SUBROUTINE CHOLA(A, N, U, NULLTY, IFAULT, RMAX, R)
C
C     ALGORITHM AS6, APPLIED STATISTICS, VOL.17, 1968, WITH
C     MODIFICATIONS BY A.J.MILLER
C
C     ARGUMENTS:-
C     A()     = INPUT, A +VE DEFINITE MATRIX STORED IN LOWER-TRIANGULAR
C               FORM.
C     N       = INPUT, THE ORDER OF A
C     U()     = OUTPUT, A LOWER TRIANGULAR MATRIX SUCH THAT U*U' = A.
C               A & U MAY OCCUPY THE SAME LOCATIONS.
C     NULLTY  = OUTPUT, THE RANK DEFICIENCY OF A.
C     IFAULT  = OUTPUT, ERROR INDICATOR
C                     = 1 IF N < 1
C                     = 2 IF A IS NOT +VE SEMI-DEFINITE
C                     = 0 OTHERWISE
C     RMAX    = OUTPUT, AN ESTIMATE OF THE RELATIVE ACCURACY OF THE
C               DIAGONAL ELEMENTS OF U.
C     R()     = OUTPUT, ARRAY CONTAINING BOUNDS ON THE RELATIVE ACCURACY
C               OF EACH DIAGONAL ELEMENT OF U.
C
C     LATEST REVISION - 18 October 1985
C
C*********************************************************************
C
      implicit double precision (a-h, o-z)
      DIMENSION A(*),U(*),R(N)
C
C     ETA SHOULD BE SET EQUAL TO THE SMALLEST +VE VALUE SUCH THAT
C     1.0 + ETA IS CALCULATED AS BEING GREATER THAN 1.0 IN THE ACCURACY
C     BEING USED.
C
      DATA ETA/1.D-16/, ZERO/0.D0/, FIVE/5.D0/
C
      IFAULT=1
      IF(N.LE.0) GO TO 100
      IFAULT=2
      NULLTY=0
      RMAX=ETA
      R(1)=ETA
      J=1
      K=0
C
C     FACTORIZE COLUMN BY COLUMN, ICOL = COLUMN NO.
C
      DO 80 ICOL=1,N
        L=0
C
C     IROW = ROW NUMBER WITHIN COLUMN ICOL
C
        DO 40 IROW=1,ICOL
          K=K+1
          W=A(K)
          IF(IROW.EQ.ICOL) RSQ=(W*ETA)**2
          M=J
```

```
            DO 10 I=1,IROW
              L=L+1
              IF(I.EQ.IROW) GO TO 20
              W=W-U(L)*U(M)
              IF(IROW.EQ.ICOL) RSQ=RSQ+(U(L)**2*R(I))**2
              M=M+1
   10       CONTINUE
   20       IF(IROW.EQ.ICOL) GO TO 50
            IF(U(L).EQ.ZERO) GO TO 30
            U(K)=W/U(L)
            GO TO 40
   30       U(K)=ZERO
            IF(ABS(W).GT.ABS(RMAX*A(K))) GO TO 100
   40     CONTINUE
C
C       END OF ROW, ESTIMATE RELATIVE ACCURACY OF DIAGONAL ELEMENT.
C
   50     RSQ=SQRT(RSQ)
          IF(ABS(W).LE.FIVE*RSQ) GO TO 60
          IF(W.LT.ZERO) GO TO 100
          U(K)=SQRT(W)
          R(I)=RSQ/W
          IF(R(I).GT.RMAX) RMAX=R(I)
          GO TO 70
   60     U(K)=ZERO
          NULLTY=NULLTY+1
   70     J=J+ICOL
   80   CONTINUE
        IFAULT=0
C
  100   RETURN
        END
```

```
CSTART OF AS 51
      SUBROUTINE LOGLIN(NVAR, DIM, NCON, CONFIG, NTAB, TABLE, FIT,
     $  LOCMAR, NMAR, MARG, NU, U, MAXDEV, MAXIT, DEV, NLAST, IFAULT)
C
C        ALGORITHM AS 51  APPL. STATIST. (1972) VOL.21, P.218
C
C        PERFORMS AN ITERATIVE PROPORTIONAL FIT OF THE MARGINAL
C        TOTALS OF A CONTINGENCY TABLE.
C
C        THIS VERSION PERMITS UP TO SEVEN VARIABLES. IF THIS LIMIT
C        IS TOO SMALL, CHANGE THE VALUE OF MAXVAR, AND OF THE
C        DIMENSION IN THE DECLARATION OF CHECK AND ICON - SEE
C        ALSO THE CHANGES NEEDED IN AS 51.1 AND AS 51.2
C
      REAL MARG(NMAR), U(NU), TABLE(NTAB), FIT(NTAB), MAXDEV,
     $  DEV(MAXIT), ZERO, X, Y, XMAX
      INTEGER DIM(NVAR), CONFIG(NVAR, NCON), LOCMAR(NCON),
     $  POINT, SIZE, ICON(7)
      LOGICAL CHECK(7)
C
      DATA MAXVAR, ZERO /7, 0.0/
C
      IFAULT = 0
      NLAST = 0
C
C        CHECK VALIDITY OF NVAR, THE NUMBER OF VARIABLES,
C        AND OF MAXIT, THE MAXIMUM NUMBER OF ITERATIONS
C
      IF (NVAR .GT. 0 .AND. NVAR .LE. MAXVAR .AND. MAXIT .GT. 0) GOTO 10
    5 IFAULT = 4
      RETURN
C
C        LOOK AT TABLE AND FIT CONSTANTS
C
   10 SIZE = 1
      DO 30 J = 1, NVAR
      IF (DIM(J) .LE. 0) GOTO 5
      SIZE = SIZE * DIM(J)
   30 CONTINUE
      IF (SIZE .LE. NTAB) GOTO 40
   35 IFAULT = 2
      RETURN
   40 X = ZERO
      Y = ZERO
      DO 60 I = 1, SIZE
      IF (TABLE(I) .LT. ZERO .OR. FIT(I) .LT. ZERO) GOTO 5
      X = X + TABLE(I)
      Y = Y + FIT(I)
   60 CONTINUE
C
C        MAKE A PRELIMINARY ADJUSTMENT TO OBTAIN THE FIT
C        TO AN EMPTY CONFIGURATION LIST
C
      IF (Y .EQ. ZERO) GOTO 5
      X = X / Y
      DO 80 I = 1, SIZE
   80 FIT(I) = X * FIT(I)
      IF (NCON .LE. 0 .OR. CONFIG(1, 1) .EQ. 0) RETURN
C
C        ALLOCATE MARGINAL TABLES
C
```

```
      POINT = 1
      DO 150 I = 1, NCON
C
C        A ZERO BEGINNING A CONFIGURATION INDICATES
C        THAT THE LIST IS COMPLETED
C
      IF (CONFIG(1, I) .EQ. 0) GOTO 160
C
C        GET MARGINAL TABLE SIZE. WHILE DOING THIS
C        TASK, SEE IF THE CONFIGURATION LIST CONTAINS
C        DUPLICATIONS OR ELEMENTS OUT OF RANGE.
C
      SIZE = 1
      DO 90 J = 1, NVAR
   90 CHECK(J) = .FALSE.
      DO 120 J = 1, NVAR
      K = CONFIG(J, I)
C
C        A ZERO INDICATES THE END OF THE STRING
C
      IF (K .EQ. 0) GOTO 130
C
C        SEE IF ELEMENT VALID
C
      IF (K .GE. 0 .AND. K .LE. NVAR) GOTO 100
   95 IFAULT = 1
      RETURN
C
C        CHECK FOR DUPLICATION
C
  100 IF (CHECK(K)) GOTO 95
      CHECK(K) = .TRUE.
C
C        GET SIZE
C
      SIZE = SIZE * DIM(K)
  120 CONTINUE
C
C        SINCE U IS USED TO STORE FITTED MARGINALS,
C        SIZE MUST NOT EXCEED NU
C
  130 IF (SIZE .GT. NU) GOTO 35
C
C        LOCMAR POINTS TO MARGINAL TABLES TO BE PLACED IN MARG
C
      LOCMAR(I) = POINT
      POINT = POINT + SIZE
  150 CONTINUE
C
C        GET N, NUMBER OF VALID CONFIGURATIONS
C
      I = NCON + 1
  160 N = I - 1
C
C        SEE IF MARG CAN HOLD ALL MARGINAL TABLES
C
      IF (POINT .GT. NMAR + 1) GOTO 35
C
C        OBTAIN MARGINAL TABLES
C
      DO 190 I = 1, N
```

```
      DO 180 J = 1, NVAR
  180 ICON(J) = CONFIG(J, I)
      CALL COLLAP(NVAR, TABLE, MARG, LOCMAR(I), NTAB, NMAR, DIM, ICON)
  190 CONTINUE
C
C        PERFORM ITERATIONS
C
      DO 220 K = 1, MAXIT
C
C        XMAX IS MAXIMUM DEVIATION OBSERVED BETWEEN
C        FITTED AND TRUE MARGINAL DURING A CYCLE
C
      XMAX = ZERO
      DO 210 I = 1, N
      DO 200 J = 1, NVAR
  200 ICON(J) = CONFIG(J, I)
      CALL COLLAP(NVAR, FIT, U, 1, NTAB, NU, DIM, ICON)
      CALL ADJUST(NVAR, FIT, U, MARG, LOCMAR(I), NTAB, NU, NMAR,
     $  DIM, ICON, XMAX)
  210 CONTINUE
C
C        TEST CONVERGENCE
C
      DEV(K) = XMAX
      IF (XMAX .LT. MAXDEV) GOTO 240
  220 CONTINUE
      IF (MAXIT .GT. 1) GOTO 230
      NLAST = 1
      RETURN
C
C        NO CONVERGENCE
C
  230 IFAULT = 3
      NLAST = MAXIT
      RETURN
C
C        NORMAL TERMINATION
C
  240 NLAST = K
      RETURN
      END
C
      SUBROUTINE COLLAP(NVAR, X, Y, LOCY, NX, NY, DIM, CONFIG)
C
C        ALGORITHM AS 51.1  APPL. STATIST. (1972) VOL.21, P.218
C
C        COMPUTES A MARGINAL TABLE FROM A COMPLETE TABLE.
C        ALL PARAMETERS ARE ASSUMED VALID WITHOUT TEST.
C
C        IF THE VALUE OF NVAR IS TO BE GREATER THAN 7, THE
C        DIMENSIONS IN THE DECLARATIONS OF SIZE AND COORD MUST
C        BE INCREASED TO NVAR+1 AND NVAR RESPECTIVELY.
C
      INTEGER SIZE(8), DIM(NVAR), CONFIG(NVAR), COORD(7)
C
C        THE LARGER TABLE IS X AND THE SMALLER ONE IS Y
C
      REAL X(NX), Y(NY), ZERO
      DATA ZERO /0.0/
C
C        INITIALISE ARRAYS
```

```
C
      SIZE(1) = 1
      DO 10 K = 1, NVAR
      L = CONFIG(K)
      IF (L .EQ. 0) GOTO 20
      SIZE(K + 1) = SIZE(K) * DIM(L)
   10 CONTINUE
C
C        FIND NUMBER OF VARIABLES IN CONFIGURATION
C
      K = NVAR + 1
   20 N = K - 1
C
C        INITIALISE Y. FIRST CELL OF MARGINAL TABLE IS
C        AT Y(LOCY) AND TABLE HAS SIZE(K) ELEMENTS
C
      LOCU = LOCY + SIZE(K) - 1
      DO 30 J = LOCY, LOCU
   30 Y(J) = ZERO
C
C        INITIALISE COORDINATES
C
      DO 50 K = 1, NVAR
   50 COORD(K) = 0
C
C        FIND LOCATIONS IN TABLES
C
      I = 1
   60 J = LOCY
      DO 70 K = 1, N
      L = CONFIG(K)
      J = J + COORD(L) * SIZE(K)
   70 CONTINUE
      Y(J) = Y(J) + X(I)
C
C        UPDATE COORDINATES
C
      I = I + 1
      DO 80 K = 1, NVAR
      COORD(K) = COORD(K) + 1
      IF (COORD(K) .LT. DIM(K)) GOTO 60
      COORD(K) = 0
   80 CONTINUE
      RETURN
      END
C
      SUBROUTINE ADJUST(NVAR, X, Y, Z, LOCZ, NX, NY, NZ, DIM, CONFIG, D)
C
C        ALGORITHM AS 51.2  APPL. STATIST. (1972) VOL.21, P.218
C
C        MAKES PROPORTIONAL ADJUSTMENT CORRESPONDING TO CONFIG.
C        ALL PARAMETERS ARE ASSUMED VALID WITHOUT TEST.
C
C        IF THE VALUE OF NVAR IS TO BE GREATER THAN 7, THE
C        DIMENSIONS IN THE DECLARATIONS OF SIZE AND COORD MUST
C        BE INCREASED TO NVAR+1 AND NVAR RESPECTIVELY.
C
      INTEGER SIZE(8), DIM(NVAR), CONFIG(NVAR), COORD(7)
      REAL X(NX), Y(NY), Z(NZ), D, E, ZERO, ZABS
C
      DATA ZERO /0.0/
```

```
C
      ZABS(E) = ABS(E)
C
C        SET SIZE ARRAY
C
      SIZE(1) = 1
      DO 10 K = 1, NVAR
      L = CONFIG(K)
      IF (L .EQ. 0) GOTO 20
      SIZE(K + 1) = SIZE(K) * DIM(L)
   10 CONTINUE
C
C        FIND NUMBER OF VARIABLES IN CONFIGURATION
C
      K = NVAR + 1
   20 N = K - 1
C
C        TEST SIZE OF DEVIATION
C
      L = SIZE(K)
      J = 1
      K = LOCZ
      DO 30 I = 1, L
      E = ZABS(Z(K) - Y(J))
      IF (E .GT. D) D = E
      J = J + 1
      K = K + 1
   30 CONTINUE
C
C        INITIALIZE COORDINATES
C
      DO 40 K = 1, NVAR
   40 COORD(K) = 0
      I = 1
C
C        PERFORM ADJUSTMENT
C
   50 J = 0
      DO 60 K = 1, N
      L = CONFIG(K)
      J = J + COORD(L) * SIZE(K)
   60 CONTINUE
      K = J + LOCZ
      J = J + 1
C
C        NOTE THAT Y(J) SHOULD BE NON-NEGATIVE
C
      IF (Y(J) .LE. ZERO) X(I) = ZERO
      IF (Y(J) .GT. ZERO) X(I) = X(I) * Z(K) / Y(J)
C
C        UPDATE COORDINATES
C
      I = I + 1
      DO 70 K = 1, NVAR
      COORD(K) = COORD(K) + 1
      IF (COORD(K) .LT. DIM(K)) GOTO 50
      COORD(K) = 0
   70 CONTINUE
      RETURN
      END
CEND OF AS 51
```

```
CSTART OF AS 52
      SUBROUTINE MOMNTS(X, K, N, S1, S2, S3, S4, IFAULT)
C
C        ALGORITHM AS 52  APPL. STATIST. (1972) VOL.21, P.226
C
C        ADDS A NEW VALUE, X, WHEN CALCULATING A MEAN, AND SUMS
C        OF POWERS OF DEVIATIONS. N IS THE CURRENT NUMBER OF
C        OBSERVATIONS, AND MUST BE SET TO ZERO BEFORE FIRST ENTRY
C
      REAL X, S1, S2, S3, S4, AN, AN1, DX, DX2, ZERO, ONE, TWO,
     $  THREE, FOUR, SIX
      DATA ZERO, ONE, TWO, THREE, FOUR, SIX
     $     /0.0, 1.0, 2.0,   3.0,  4.0, 6.0/
C
      IF (K .GT. 0 .AND. K .LT. 5 .AND. N .GE. 0) GOTO 10
      IFAULT = 1
      RETURN
   10 IFAULT = 0
      N = N + 1
      IF (N .GT. 1) GOTO 20
C
C        FIRST ENTRY, SO INITIALISE
C
      S1 = X
      S2 = ZERO
      S3 = ZERO
      S4 = ZERO
      RETURN
C
C        SUBSEQUENT ENTRY, SO UPDATE
C
   20 AN = N
      AN1 = AN - ONE
      DX = (X - S1) / AN
      DX2 = DX * DX
      GOTO (60, 50, 40, 30), K
   30 S4 = S4 - DX * (FOUR * S3 - DX * (SIX * S2 + AN1 *
     $  (ONE + AN1 ** 3) * DX2))
   40 S3 = S3 - DX * (THREE * S2 - AN * AN1 * (AN - TWO) * DX2)
   50 S2 = S2 + AN * AN1 * DX2
   60 S1 = S1 + DX
      RETURN
      END
CEND OF AS 52
```

```
      SUBROUTINE WSHRT(D, N, NP, NNP, SB, SA)
C
C     ALGORITHM AS 53  APPL. STATIST. (1972) VOL.21, NO.3
C
C     Wishart variate generator.  On output, SA is an upper-triangular
C     matrix of size NP * NP (written in linear form, column ordered)
C     whose elements have a Wishart(N, SIGMA) distribution.
C
C     D is an upper-triangular array such that SIGMA = D'D (see AS 6)
C
C     Auxiliary function required: a random no. generator called RAND.
C     The Wichmann & Hill generator is included here.   It should be
C     initialized in the calling program.
C
      INTEGER N, NP, NNP
      REAL D(NNP), SB(NNP), SA(NNP)
C
C     Local variables
C
      INTEGER K, NS, I, J, NR, IP, NQ, II
      REAL DF, U1, U2, RN, C
      REAL ZERO, ONE, TWO, NINE
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, NINE /9.0/
C
      K = 1
    1 CALL RNORM(U1, U2)
C
C     Load SB with independent normal (0, 1) variates
C
      SB(K) = U1
      K = K + 1
      IF (K .GT. NNP) GO TO 2
      SB(K) = U2
      K = K + 1
      IF (K .LE. NNP) GO TO 1
    2 NS = 0
C
C     Load diagonal elements with square root of chi-square variates
C
      DO 3 I = 1, NP
        DF = N - I + 1
        NS = NS + I
        U1 = TWO / (NINE * DF)
        U2 = ONE - U1
        U1 = SQRT(U1)
C
C     Wilson-Hilferty formula for approximating chi-square variates
C
        SB(NS) = SQRT(DF * (U2 + SB(NS) * U1)**3)
    3 CONTINUE
C
      RN = N
      NR = 1
      DO 5 I = 1, NP
        NR = NR + I - 1
        DO 5 J = I, NP
          IP = NR
          NQ = (J*J - J) / 2 + I - 1
          C = ZERO
          DO 4 K = I, J
            IP = IP + K - 1
```

```
              NQ = NQ + 1
              C = C + SB(IP) * D(NQ)
        4     CONTINUE
              SA(IP) = C
        5 CONTINUE
C
          DO 7 I = 1, NP
            II = NP - I + 1
            NQ = NNP - NP
            DO 7 J = 1, I
              IP = (II*II - II) / 2
              C = ZERO
              DO 6 K = I, NP
                IP = IP + 1
                NQ = NQ + 1
                C = C + SA(IP) * SA(NQ)
        6     CONTINUE
              SA(NQ) = C / RN
              NQ = NQ - 2 * NP + I + J - 1
        7 CONTINUE
C
          RETURN
          END
C
C
          SUBROUTINE RNORM(U1, U2)
C
C     ALGORITHM AS 53.1  APPL. STATIST. (1972) VOL.21, NO.3
C
C     Sets U1 and U2 to two independent standardized random normal
C     deviates.   This is a Fortran version of the method given in
C     Knuth(1969).
C
C     Function RAND must give a result randomly and rectangularly
C     distributed between the limits 0 and 1 exclusive.
C
          REAL U1, U2
          REAL RAND
C
C     Local variables
C
          REAL X, Y, S, ONE, TWO
          DATA ONE /1.0/, TWO /2.0/
C
        1 X = RAND()
          Y = RAND()
          X = TWO * X - ONE
          Y = TWO * Y - ONE
          S = X * X + Y * Y
          IF (S .GT. ONE) GO TO 1
          S = SQRT(- TWO * LOG(S) / S)
          U1 = X * S
          U2 = Y * S
          RETURN
          END
```

```
CSTART OF AS 57
      SUBROUTINE TABWRT(TITLE, NT1, NT2, TABLE, NTAB, DIM, NVAR,
     $  LOC, COL, DEC, VARNAM, WORDS, CATNAM, MAXCAT, COLLAB,
     $  VERT, RESTOR, LINE, SKIP, PAGE, WIDTH, UNIT, IFAULT)
C
C        ALGORITHM AS 57  APPL. STATIST. (1973) VOL.22, P.118
C
C        TABWRT PRINTS MULTIDIMENSIONAL TABLES.
C
C        A TABLE CAN HAVE AT MOST MAXVAR VARIABLES. NOTE THAT
C        SIZE HAS DIMENSION MAXVAR+1 AND COORD HAS DIMENSION MAXVAR
C
      LOGICAL RESTOR
      INTEGER DIM(NVAR), COL, LOC(COL), DEC(COL), WORDS, VERT
      INTEGER SKIP, PAGE, WIDTH, UNIT
      INTEGER SIZE(8), COORD(7), HOR, DOWN, SPACE, ONEPAG
C
C        ARRAYS CONTAINING DATA AND LABELS
C
      REAL TITLE(NT1, NT2), TABLE(NTAB), VARNAM(WORDS, NVAR),
     $  CATNAM(WORDS, MAXCAT, NVAR), COLLAB(WORDS, COL)
      REAL OUT(33), CHAR(3)
      REAL TAB1, TAB2, XLAB1, XLAB2, XLAB3
C
C        VARIABLE FORMATS.
C
      REAL FMT(15), FMTT(7)
      DATA FMTT(1),FMTT(2),FMTT(3),FMTT(4),FMTT(5),FMTT(6),FMTT(7)
     $    /4H(A1, ,1H     ,2HX,   ,1H     ,1HA    ,1H     ,1H)    /
C
      DATA FMT(1),FMT(2),FMT(3),FMT(4),FMT(5),FMT(6),FMT(7),FMT(8)
     $    /4H(A1,,1H    ,2HX,  ,1H    ,4H(4X,,1H    ,1HA   ,1H    /
C
      DATA FMT(9),FMT(10),FMT(11),FMT(12),FMT(13),FMT(14),FMT(15)
     $    /2H),  ,1H     ,1HF    ,1H     ,1H.    ,1H0    ,1H)    /
C
C        CARRIAGE CONTROL.
C
      DATA CHAR(1), CHAR(2), CHAR(3) /1H , 1H0, 1H1/
C
C        ADJUSTABLE CONSTANTS - SEE INTRODUCTORY TEXT
C
      DATA LENGTH, MAXVAR /4, 7/
C
C        SET ERROR INDICATOR AND GET SIZES.
C
      IFAULT = 0
      IF (NVAR .GT. MAXVAR) GOTO 98
      SIZE(1) = 1
      DO 1 I = 1, NVAR
      COORD(I) = 1
      IF (DIM(I) .LE. 0) GOTO 2
      IF (DIM(I) .GT. MAXCAT) GOTO 98
      SIZE(I + 1) = SIZE(I) * DIM(I)
    1 CONTINUE
      I = NVAR + 1
    2 N = I - 1
      DOWN = VERT
C
C        ALLOCATE VARIABLES TO HORIZONTAL AND VERTICAL SCALES.
C
```

```
      IF (DOWN .GT. N) DOWN = N
      IF (DOWN .LT. 1) DOWN = 1
      HOR = N - DOWN
C
C        LABELW IS WIDTH OF LABEL.
C
      LABELW = WORDS * LENGTH + 4
      I = HOR + 1
      K = I
      DO 3 J = 1, I
C
C        GET LINE WIDTH.
C
      LINEW = LABELW * (SIZE(K) + DOWN + 1)
      IF (LINEW .LT. WIDTH) GOTO 4
      K = K - 1
      DOWN = DOWN + 1
    3 CONTINUE
C
C        NOTE THAT LINE IS NOT WIDE ENOUGH.
C
      IFAULT = 2
      RETURN
    4 HOR = N - DOWN
C
C        SET FORMATS. CONVRT IS AN AUXILIARY
C        INTEGER TO ALPHANUMERIC ROUTINE.
C
      FMTT(2) = CONVRT((WIDTH - LENGTH * NT1) / 2 + 1)
      FMTT(4) = CONVRT(NT1)
      FMTT(6) = CONVRT(LENGTH)
      FMT(8) = FMTT(6)
      I = (WIDTH - LINEW) / 2 + 1
      TAB1 = CONVRT(I)
      TAB2 = CONVRT(I + LABELW * DOWN)
      XLAB1 = CONVRT(DOWN + 1)
      XLAB2 = CONVRT(1)
      FMT(6) = CONVRT(WORDS)
      FMT(12) = CONVRT(LABELW)
C
C        NOW ASCERTAIN VERTICAL SPACE REQUIREMENTS.
C
      SPACE = (COL + 1) * SIZE(N + 1) / SIZE(HOR + 1)
      ONEPAG = SPACE + 2 + HOR + NT2 + NT2 + SKIP
      IF (RESTOR .OR. LINE + ONEPAG .GT. PAGE) LINE = 0
      ICAR = 2
C
C        SEE IF CARRIAGE RESTORE NEEDED
C
      IF (LINE .NE. 0) GOTO 5
      ICAR = 3
      GOTO 7
C
C        SKIP APPROPRIATE NUMBER OF LINES.
C
    5 DO 6 I = 1, SKIP
    6 WRITE (UNIT, 100)
  100 FORMAT(1H )
    7 I = LINE + ONEPAG
      IF (I .LE. PAGE) LINE = I
C
```

```
C        SEE HOW MUCH FITS ON A PAGE.
C
      INDEX = N + 1
      K = PAGE - 2 - HOR - NT2 - NT2
      DO 8 I = 1, DOWN
      IF (SPACE .LE. K) GOTO 9
      INDEX = INDEX - 1
      SPACE = SPACE / DIM(INDEX)
    8 CONTINUE
C
C        RETURN WITH IFAULT = 2 IF TOO LITTLE SPACE AVAILABLE.
C
      IFAULT = 2
      RETURN
C
C        NOTCH IS UNITS PER PAGE.
C
    9 NOTCH = K / SPACE * SIZE(INDEX)
      INC = SIZE(HOR + 1)
      INC1 = INC - 1
      XLAB3 = CONVRT(INC + 1)
      FMT(10) = CONVRT(INC)
      NUM = (SIZE(N + 1) + NOTCH - 1) / NOTCH * NOTCH
      LL = HOR + 1
C
C        SET MARKER.
C
      MARK = 0
      INDEX = WORDS * DOWN
C
C        PRINT A PAGE.
C
      DO 18 I = NOTCH, NUM, NOTCH
      IC = ICAR
C
C        PRINT TITLE.
C
      DO 10 K = 1, NT2
      WRITE (UNIT, FMTT) CHAR(IC), (TITLE(J, K), J = 1, NT1)
      IC = 1
   10 CONTINUE
C
C        SKIP TWO LINES.
C
      WRITE (UNIT, 101)
  101 FORMAT(1H0)
C
C        PRINT HORIZONTAL LABELS.
C
      IF (HOR .EQ. 0) GOTO 12
      FMT(2) = TAB2
      FMT(4) = XLAB3
      DO 11 K = 1, HOR
      L = HOR - K + 1
      I1 = SIZE(L)
      I2 = SIZE(L + 1)
      I3 = DIM(L)
      WRITE (UNIT, FMT) CHAR(1), (VARNAM(I4, L), I4 = 1, WORDS),
     $  ((((CATNAM(I4, I5, L), I4 = 1, WORDS), I6 = 1, I1),
     $  I5 = 1, I3), I7 = I2, INC, I2)
   11 CONTINUE
```

```
C
C        VERTICAL LABELS.
C
   12 J = 0
      FMT(2) = TAB1
      FMT(4) = XLAB1
      M = N + 1
      DO 13 K = LL, N
      M = M - 1
      DO 13 I1 = 1, WORDS
      J = J + 1
      OUT(J) = VARNAM(I1, M)
   13 CONTINUE
      WRITE (UNIT, FMT) CHAR(1), (OUT(J), J = 1, INDEX)
C
C        NOW PRINT BODY OF TABLE.
C
      DO 18 I2 = INC, NOTCH, INC
      J = 0
      M = N + 1
      DO 14 K = LL, N
      M = M - 1
      I3 = COORD(M)
      DO 14 I1 = 1, WORDS
      J = J + 1
      OUT(J) = CATNAM(I1, I3, M)
   14 CONTINUE
      I3 = LOC(1) + MARK
      I4 = I3 + INC1
      FMT(2) = TAB1
      FMT(4) = XLAB1
      FMT(14) = CONVRT(DEC(1))
      WRITE (UNIT, FMT) CHAR(2), (OUT(J), J = 1, INDEX),
     $  (COLLAB(J, 1), J = 1, WORDS), (TABLE(I5), I5 = I3, I4)
      IF (COL .LE. 1) GOTO 16
      FMT(2) = TAB2
      FMT(4) = XLAB2
      DO 15 I6 = 2, COL
      I3 = LOC(I6) + MARK
      I4 = I3 + INC1
      FMT(14) = CONVRT(DEC(I6))
      WRITE (UNIT, FMT) CHAR(1), (COLLAB(I5, I6), I5 = 1, WORDS),
     $  (TABLE(I5), I5 = I3, I4)
   15 CONTINUE
   16 MARK = MARK + INC
C
C        RESET COORDINATES.
C
      DO 17 I1 = LL, N
      COORD(I1) = COORD(I1) + 1
      IF (COORD(I1) .LE. DIM(I1)) GOTO 18
      COORD(I1) = 1
   17 CONTINUE
      RETURN
   18 CONTINUE
C
C        ERROR RETURN.
C
   98 IFAULT = 1
      RETURN
      END
```

```
C
      FUNCTION CONVRT(I)
C
C        ALGORITHM AS 57.1  APPL. STATIST. (1973) VOL.22, P.118
C
C        CONVERT FROM INTEGER TO ALPHANUMERIC.
C        THE NORMAL RANGE FOR CONVRT IS FROM 0 TO 101.
C        ALL INTEGERS TO BE CONVERTED WILL BE WITHIN THIS
C        RANGE IF WIDTH DOES NOT EXCEED 131, LENGTH IS AT
C        LEAST 4, AND MAXVAR DOES NOT EXCEED 7.
C
      REAL C(102)
      DATA  C(1), C(2), C(3), C(4), C(5) / 1H0, 1H1, 1H2, 1H3, 1H4/
      DATA  C(6), C(7), C(8), C(9),C(10) / 1H5, 1H6, 1H7, 1H8, 1H9/
      DATA C(11),C(12),C(13),C(14),C(15) /2H10,2H11,2H12,2H13,2H14/
      DATA C(16),C(17),C(18),C(19),C(20) /2H15,2H16,2H17,2H18,2H19/
      DATA C(21),C(22),C(23),C(24),C(25) /2H20,2H21,2H22,2H23,2H24/
      DATA C(26),C(27),C(28),C(29),C(30) /2H25,2H26,2H27,2H28,2H29/
      DATA C(31),C(32),C(33),C(34),C(35) /2H30,2H31,2H32,2H33,2H34/
      DATA C(36),C(37),C(38),C(39),C(40) /2H35,2H36,2H37,2H38,2H39/
      DATA C(41),C(42),C(43),C(44),C(45) /2H40,2H41,2H42,2H43,2H44/
      DATA C(46),C(47),C(48),C(49),C(50) /2H45,2H46,2H47,2H48,2H49/
      DATA C(51),C(52),C(53),C(54),C(55) /2H50,2H51,2H52,2H53,2H54/
      DATA C(56),C(57),C(58),C(59),C(60) /2H55,2H56,2H57,2H58,2H59/
      DATA C(61),C(62),C(63),C(64),C(65) /2H60,2H61,2H62,2H63,2H64/
      DATA C(66),C(67),C(68),C(69),C(70) /2H65,2H66,2H67,2H68,2H69/
      DATA C(71),C(72),C(73),C(74),C(75) /2H70,2H71,2H72,2H73,2H74/
      DATA C(76),C(77),C(78),C(79),C(80) /2H75,2H76,2H77,2H78,2H79/
      DATA C(81),C(82),C(83),C(84),C(85) /2H80,2H81,2H82,2H83,2H84/
      DATA C(86),C(87),C(88),C(89),C(90) /2H85,2H86,2H87,2H88,2H89/
      DATA C(91),C(92),C(93),C(94),C(95) /2H90,2H91,2H92,2H93,2H94/
      DATA C(96),C(97),C(98),C(99),C(100)/2H95,2H96,2H97,2H98,2H99/
      DATA C(101),C(102)                 /3H100,3H101/
      J = I + 1
      IF (J .LT. 1) J = 1
      IF (J .GT. 102) J = 102
      CONVRT = C(J)
      RETURN
      END
CEND OF AS 57
```

```
      subroutine clustr (x,d,dev,b,f,e,i,j,n,nz,k)
      implicit double precision (a-h,o-z)
c
c     Algorithm AS 58  Appl. Statist. (1973) Vol.22, No.1
c
c     Given a matrix of i observations on j variables, the
c     observations are allocated to n clusters in such a way that the
c     within-cluster sum of squares is minimised.
c
      integer b(i),e(k)
      dimension x(i,j),d(k,j),dev(k),f(i)
      data zero/0.0d0/, one/1.0d0/, two/2.0d0/
      data big/1.0d10/
c
      do 10 ia=1,n
        e(ia)=0
   10 continue
c
c     For each observation, calculate the distance from each cluster
c     centre, and assign to the nearest
c
      do 40 ic=1,i
        f(ic)=zero
        da=big
        do 30 id=1,n
          db=zero
          do 20 ie=1,j
            dc=x(ic,ie)-d(id,ie)
            db=db+dc*dc
            if (db.ge.da) go to 30
   20     continue
          da=db
          b(ic)=id
   30   continue
        ig=b(ic)
        e(ig)=e(ig)+1
   40 continue
c
c     Calculate the mean and sum of squares for each cluster
c
      do 50 ix=1,n
        dev(ix)=zero
        do 50 iy=a1,j
          d(ix,iy)=zero
   50 continue
      do 60 ic=1,i
        ig=b(ic)
        do 60 ih=1,j
          d(ig,ih)=d(ig,ih)+x(ic,ih)
   60 continue
      do 80 ij=1,j
        do 70 ii=1,n
          d(ii,ij)=d(ii,ij)/dble(e(ii))
   70   continue
        do 80 ik=1,i
          il=b(ik)
          da=x(ik,ij)-d(il,ij)
          db=da*da
          f(ik)=f(ik)+db
          dev(il)=dev(il)+db
   80 continue
```

```
      do 85 ik=1,i
        il=b(ik)
        fl=e(il)
        if (fl.ge.two) f(ik)=f(ik)*fl/(fl-one)
   85 continue
c
c     Examine each observation in turn to see if it should be
c     reassigned to a different cluster
c
      if (nz.le.0) nz=1
   90 iw=0
      do 140 ik=1,i
      il=b(ik)
      ir=il
c
c     If the number of cluster points is less than or equal to the
c     specified minimum, nz, bypass this iteration
c
      if (e(il).le.nz) go to 140
      fl=e(il)
      dc=f(ik)
      do 100 in=1,n
        if (in.eq.il) goto 100
        fm=e(in)
        fm=fm/(fm+one)
        de=zero
        do 95 ip=1,j
          da=x(ik,ip)-d(in,ip)
          de=de + da *da * fm
          if (de.ge.dc) goto 100
   95 continue
      dc=de
      ir=in
  100 continue
      if (ir.eq.il) goto 140
c
c     Reassignment is made here if necessary
c
      fq=e(ir)
      dev(il)=dev(il)-f(ik)
      dev(ir)=dev(ir)+dc
      e(ir)=e(ir)+1
      e(il)=e(il)-1
      do 110 is=1,j
        d(il,is) = (d(il,is) * fl - x(ik,is)) / (fl-one)
        d(ir,is) = (d(ir,is) * fq + x(ik,is)) / (fq + one)
  110 continue
      b(ik) = ir
      do 130 it=1,i
        ij = b(it)
        if (ij.ne.il. and. ij.ne.ir) goto 130
        f(it)=zero
        do 120 iu=1,j
          da = x(it,iu) - d(ij,iu)
          f(it)=f(it) + da * da
  120   continue
        fl=e(ij)
        f(it) = f(it)*fl/(fl-one)
  130 continue
      iw=iw+1
  140 continue
```

```
c
c     Return to calling program if no reassignments were made during
c     this iteration
c
      if (iw.eq.o) return
      goto 90
      end
```

```
      SUBROUTINE TDIAG (N,TOL,A,D,E,Z,MAXDIM,IFAULT)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
      DIMENSION A(MAXDIM,MAXDIM), D(MAXDIM), E(MAXDIM), Z(MAXDIM,MAXDIM)
      DATA ZERO/0.0/,ONE/1.0/
      DATA ETA/1.0D-37/,EPS/1.0D-14/
C
C       Algorithm as 60.1 appl.statist. (1973) vol.22 no.2
C
C        reduces real symmetric matrix to tridiagonal form
C
C       tol is a machine dependent constant , tol = eta/eps , where
C       eta is the smallest positive number representable in the
C       computer and eps is the smallest positive number for which
C       1+eps.ne.1.
C
C         eta=eps*tol
C         eps=0.7105427358e-14
C         tol=0.3131513063e-293
C         precis=1.0e-14
C
C         nb
C           real constants must be le 15 decimal digits
C           the range of a real constant is from 1.0e-293 to 1.0e+322
C
      TOL=ETA/EPS
      IFAULT=1
      IF (N.LE.1) RETURN
      IFAULT=0
      DO 10 I=1,N
        DO 10 J=1,I
 10     Z(I,J)=A(I,J)
      I=N
      DO 110 I1=2,N
        L=I-2
        F=Z(I,I-1)
        G=ZERO
        IF (L.LT.1) GO TO 30
        DO 20 K=1,L
 20       G=G+Z(I,K)**2
 30     H=G+F*F
C
C       if g is too small for orthogonality to be guaranteed, the
C       transformation is skipped
C
        IF (G.GT.TOL) GO TO 40
        E(I)=F
        D(I)=ZERO
        GO TO 100
 40     L=L+1
        G=SQRT(H)
        IF (F.GE.ZERO) G=-G
        E(I)=G
        H=H-F*G
        Z(I,I-1)=F-G
        F=ZERO
        DO 80 J=1,L
          Z(J,I)=Z(I,J)/H
          G=ZERO
C
C       form element of a * u
```

```
C
          DO 50 K=1,J
 50         G=G+Z(J,K)*Z(I,K)
          IF (J.GE.L) GO TO 70
          J1=J+1
          DO 60 K=J1,L
 60         G=G+Z(K,J)*Z(I,K)
C
C       form element of p
C
 70       E(J)=G/H
          F=F+G*Z(J,I)
 80     CONTINUE
C
C       form k
C
        HH=F/(H+H)
C
C       form reduced a
C
        DO 90 J=1,L
          F=Z(I,J)
          G=E(J)-HH*F
          E(J)=G
          DO 90 K=1,J
          Z(J,K)=Z(J,K)-F*E(K)-G*Z(I,K)
 90     CONTINUE
        D(I)=H
 100    I=I-1
 110  CONTINUE
      D(1)=ZERO
      E(1)=ZERO
C
C       accumulation of transformation matrices
C
      DO 160 I=1,N
        L=I-1
        IF (D(I).EQ.ZERO.OR.L.EQ.0) GO TO 140
        DO 130 J=1,L
          G=ZERO
          DO 120 K=1,L
 120        G=G+Z(I,K)*Z(K,J)
          DO 130 K=1,L
          Z(K,J)=Z(K,J)-G*Z(K,I)
 130    CONTINUE
 140    D(I)=Z(I,I)
        Z(I,I)=ONE
        IF (L.EQ.0) GO TO 160
        DO 150 J=1,L
          Z(I,J)=ZERO
          Z(J,I)=ZERO
 150    CONTINUE
 160  CONTINUE
      RETURN
      END
C
C
      SUBROUTINE LRVT (N,PRECIS,D,E,Z,IFAULT,MAXDIM)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
C       algorithm as 60.2 appl.statist. (1973) vol.22, no.2
```

```
C
C        finds latent roots and vectors of tridiagonal matrix
C
      DIMENSION D(MAXDIM), E(MAXDIM), Z(MAXDIM,MAXDIM)
      DATA MITS/30/,ZERO/0.0/,ONE/1.0/,TWO/2.0/
C
      PRECIS=1.0D-14
      IFAULT=2
      IF (N.LE.1) RETURN
      IFAULT=1
      N1=N-1
      DO 10 I=2,N
 10     E(I-1)=E(I)
      E(N)=ZERO
      B=ZERO
      F=ZERO
      DO 100 L=1,N
        JJ=0
        H=PRECIS*(ABS(D(L))+ABS(E(L)))
        IF (B.LT.H) B=H
C
C       look for small sub-diagonal element
C
        DO 20 M1=L,N
          M=M1
          IF (ABS(E(M)).LE.B) GO TO 30
 20     CONTINUE
 30     IF (M.EQ.L) GO TO 90
 40     IF (JJ.EQ.MITS) RETURN
        JJ=JJ+1
C
C       form shift
C
        P=(D(L+1)-D(L))/(TWO*E(L))
        R=SQRT(P*P+ONE)
        PR=P+R
        IF (P.LT.ZERO) PR=P-R
        H=D(L)-E(L)/PR
        DO 50 I=L,N
 50       D(I)=D(I)-H
        F=F+H
C
C       ql transformation
C
        P=D(M)
        C=ONE
        S=ZERO
        M1=M-1
        I=M
        DO 80 I1=L,M1
          J=I
          I=I-1
          G=C*E(I)
          H=C*P
          IF (ABS(P).GE.ABS(E(I))) GO TO 60
          C=P/E(I)
          R=SQRT(C*C+ONE)
          E(J)=S*E(I)*R
          S=ONE/R
          C=C/R
          GO TO 70
```

```
 60       C=E(I)/P
          R=SQRT(C*C+ONE)
          E(J)=S*P*R
          S=C/R
          C=ONE/R
 70       P=C*D(I)-S*G
          D(J)=H+S*(C*G+S*D(I))
C
C       form vector
C
          DO 80 K=1,N
          H=Z(K,J)
          Z(K,J)=S*Z(K,I)+C*H
          Z(K,I)=C*Z(K,I)-S*H
 80     CONTINUE
        E(L)=S*P
        D(L)=C*P
        IF (ABS(E(L)).GT.B) GO TO 40
 90     D(L)=D(L)+F
 100  CONTINUE
C
C       order latent roots and vectors
C
      DO 130 I=1,N1
        K=I
        P=D(I)
        I1=I+1
        DO 110 J=I1,N
          IF (D(J).LE.P) GO TO 110
          K=J
          P=D(J)
 110    CONTINUE
        IF (K.EQ.I) GO TO 130
        D(K)=D(I)
        D(I)=P
        DO 120 J=1,N
          P=Z(J,I)
          Z(J,I)=Z(J,K)
          Z(J,K)=P
 120    CONTINUE
 130  CONTINUE
      IFAULT=0
      RETURN
      END
```

```
c AS 62 generates the frequencies for the Mann-Whitney U-statistic.
c Users are much more likely to need the distribution function.
c Code to return the distribution function has been added at the end
c of AS 62 by Alan Miller.   Remove the C's in column 1 to activate it.
c
      SUBROUTINE UDIST(M, N, FRQNCY, LFR, WORK, LWRK, IFAULT)
C
C     ALGORITHM AS 62  APPL. STATIST. (1973) VOL.22, NO.2
C
C     The distribution of the Mann-Whitney U-statistic is generated for
C     the two given sample sizes
C
      INTEGER M, N, LFR, LWRK, IFAULT
      REAL FRQNCY(LFR), WORK(LWRK)
C
C     Local variables
C
      INTEGER MINMN, MN1, MAXMN, N1, I, IN, L, K, J
      REAL ZERO, ONE, SUM
      DATA ZERO /0.0/, ONE /1.0/
C
C     Check smaller sample size
C
      IFAULT = 1
      MINMN = MIN(M, N)
      IF (MINMN .LT. 1) RETURN
C
C     Check size of results array
C
      IFAULT = 2
      MN1 = M * N + 1
      IF (LFR .LT. MN1) RETURN
C
C     Set up results for 1st cycle and return if MINMN = 1
C
      MAXMN = MAX(M, N)
      N1 = MAXMN + 1
      DO 1 I = 1, N1
    1 FRQNCY(I) = ONE
      IF (MINMN .EQ. 1) GO TO 4
C
C     Check length of work array
C
      IFAULT = 3
      IF (LWRK .LT. (MN1 + 1) / 2 + MINMN) RETURN
C
C     Clear rest of FREQNCY
C
      N1 = N1 + 1
      DO 2 I = N1, MN1
    2 FRQNCY(I) = ZERO
C
C     Generate successively higher order distributions
C
      WORK(1) = ZERO
      IN = MAXMN
      DO 3 I = 2, MINMN
        WORK(I) = ZERO
        IN = IN + MAXMN
        N1 = IN + 2
        L = 1 + IN / 2
```

```
        K = I
C
C     Generate complete distribution from outside inwards
C
        DO 3 J = 1, L
          K = K + 1
          N1 = N1 - 1
          SUM = FRQNCY(J) + WORK(J)
          FRQNCY(J) = SUM
          WORK(K) = SUM - FRQNCY(N1)
          FRQNCY(N1) = SUM
    3 CONTINUE
C
    4 IFAULT = 0
C
C     Code to overwrite the frequency function with the distribution
C     function.   N.B. The frequency in FRQNCY(1) is for U = 0, and
C     that in FRQNCY(I) is for U = I - 1.
C
C     SUM = ZERO
C     DO 10 I = 1, MN1
C       SUM = SUM + FRQNCY(I)
C       FRQNCY(I) = SUM
C  10 CONTINUE
C     DO 20 I = 1, MN1
C  20 FRQNCY(I) = FRQNCY(I) / SUM
C
      RETURN
      END
```

```
      double precision function betain(x, p, q, beta, ifault)
      implicit double precision (a-h, o-z)
c
c     algorithm as 63  appl. statist. (1973), vol.22, no.3
c
c     computes incomplete beta function ratio for arguments
c     x between zero and one, p and q positive.
c     log of complete beta function, beta, is assumed to be known
c
      logical indx
c
c     define accuracy and initialise
c
      data zero/0.0d0/, one/1.0d0/, acu/0.1d-14/
      betain=x
c
c     test for admissibility of arguments
c
      ifault=1
      if(p.le.zero .or. q.le.zero) return
      ifault=2
      if(x.lt.zero .or. x.gt.one) return
      ifault=0
      if(x.eq.zero .or. x.eq. one) return
c
c     change tail if necessary and determine s
c
      psq=p+q
      cx=one-x
      if(p.ge.psq*x) goto 1
      xx=cx
      cx=x
      pp=q
      qq=p
      indx=.true.
      goto 2
    1 xx=x
      pp=p
      qq=q
      indx=.false.
    2 term=one
      ai=one
      betain=one
      ns=qq+cx*psq
c
c     user soper's reduction formulae.
c
      rx=xx/cx
    3 temp=qq-ai
      if(ns.eq.0) rx=xx
    4 term=term*temp*rx/(pp+ai)
      betain=betain+term
      temp=abs(term)
      if(temp.le.acu .and. temp.le.acu*betain) goto 5
      ai=ai+one
      ns=ns-1
      if(ns.ge.0) goto 3
      temp=psq
      psq=psq+one
      goto 4
c
```

```
c     calculate result
c
    5 betain=betain*exp(pp*log(xx)+(qq-one)*log(cx)-beta)/pp
      if(indx) betain=one-betain
      return
      end
```

```
CSTART OF AS 64/109
      REAL FUNCTION XINBTA(P, Q, BETA, ALPHA, IFAULT)
C
C        ALGORITHM AS 109  APPL. STATIST. (1977) VOL.26, P.111
C        (REPLACING ALGORITHM AS 64  APPL. STATIST. (1973),
C        VOL.22, P.411)
C
C        COMPUTES INVERSE OF INCOMPLETE BETA FUNCTION
C        RATIO FOR GIVEN POSITIVE VALUES OF THE ARGUMENTS
C        P AND Q, ALPHA BETWEEN ZERO AND ONE.
C        LOG OF COMPLETE BETA FUNCTION, BETA, IS ASSUMED TO BE
C        KNOWN.
C
      LOGICAL INDEX
      REAL A, ALPHA, ADJ, BETA, G, H, P, PP, PREV, Q, QQ, R, S,
     $  SQ, T, TX, W, Y, YPREV, ZERO, HALF, ONE, TWO, THREE,
     $  FOUR, FIVE, SIX, NINE, ACU, LOWER, UPPER, CONST1,
     $  CONST2, CONST3, CONST4, BETAIN, ZEXP, ZLOG, ZSQRT
C
C        DEFINE ACCURACY AND INITIALIZE
C
      DATA ZERO, HALF, ONE, TWO, THREE, FOUR, FIVE, SIX, NINE
     $     /0.0,  0.5, 1.0, 2.0,   3.0,  4.0,  5.0, 6.0,  9.0/
      DATA   ACU,  LOWER,  UPPER,  CONST1,  CONST2,  CONST3,  CONST4
     $  /1.0E-14, 0.0001, 0.9999, 2.30753, 0.27061, 0.99229, 0.04481/
C
      ZEXP(A) = EXP(A)
      ZLOG(A) = ALOG(A)
      ZSQRT(A) = SQRT(A)
C
      XINBTA = ALPHA
C
C        TEST FOR ADMISSIBILITY OF PARAMETERS
C
      IFAULT = 1
      IF (P .LE. ZERO .OR. Q .LE. ZERO) RETURN
      IFAULT = 2
      IF (ALPHA .LT. ZERO .OR. ALPHA .GT. ONE) RETURN
      IFAULT = 0
      IF (ALPHA .EQ. ZERO .OR. ALPHA .EQ. ONE) RETURN
C
C        CHANGE TAIL IF NECESSARY
C
      IF (ALPHA .LE. HALF) GOTO 1
      A = ONE - ALPHA
      PP = Q
      QQ = P
      INDEX = .TRUE.
      GOTO 2
    1 A = ALPHA
      PP = P
      QQ = Q
      INDEX = .FALSE.
C
C        CALCULATE THE INITIAL APPROXIMATION
C
    2 R = ZSQRT(-ZLOG(A * A))
      Y = R - (CONST1 + CONST2 * R) / (ONE + (CONST3 + CONST4 *
     $  R) * R)
      IF (PP .GT. ONE .AND. QQ .GT. ONE) GOTO 5
      R = QQ + QQ
```

```
        T = ONE / (NINE * QQ)
        T = R * (ONE - T + Y * ZSQRT(T)) ** 3
        IF (T .LE. ZERO) GOTO 3
        T = (FOUR * PP + R - TWO) / T
        IF (T .LE. ONE) GOTO 4
        XINBTA = ONE - TWO / (T + ONE)
        GOTO 6
      3 XINBTA = ONE - ZEXP((ZLOG((ONE - A) * QQ) + BETA) / QQ)
        GOTO 6
      4 XINBTA = ZEXP((ZLOG(A * PP) + BETA) / PP)
        GOTO 6
      5 R = (Y * Y - THREE) / SIX
        S = ONE / (PP + PP - ONE)
        T = ONE / (QQ + QQ - ONE)
        H = TWO / (S + T)
        W = Y * ZSQRT(H + R) / H - (T - S) * (R + FIVE / SIX -
     $  TWO / (THREE * H))
        XINBTA = PP / (PP + QQ * ZEXP(W + W))
C
C          SOLVE FOR X BY A MODIFIED NEWTON-RAPHSON METHOD,
C          USING THE FUNCTION BETAIN
C
      6 R = ONE - PP
        T = ONE - QQ
        YPREV = ZERO
        SQ = ONE
        PREV = ONE
        IF (XINBTA .LT. LOWER) XINBTA = LOWER
        IF (XINBTA .GT. UPPER) XINBTA = UPPER
      7 Y = BETAIN(XINBTA, PP, QQ, BETA, IFAULT)
        IF (IFAULT .EQ. 0) GOTO 8
        IFAULT = 3
        RETURN
      8 Y = (Y - A) * ZEXP(BETA + R * ZLOG(XINBTA) + T *
     $  ZLOG(ONE - XINBTA))
        IF (Y * YPREV .LE. ZERO) PREV = SQ
        G = ONE
      9 ADJ = G * Y
        SQ = ADJ * ADJ
        IF (SQ .GE. PREV) GOTO 10
        TX = XINBTA - ADJ
        IF (TX .GE. ZERO .AND. TX .LE. ONE) GOTO 11
     10 G = G / THREE
        GOTO 9
     11 IF (PREV .LE. ACU) GOTO 12
        IF (Y * Y .LE. ACU) GOTO 12
        IF (TX .EQ. ZERO .OR. TX .EQ. ONE) GOTO 10
        IF (TX .EQ. XINBTA) GOTO 12
        XINBTA = TX
        YPREV = Y
        GOTO 7
     12 IF (INDEX) XINBTA = ONE - XINBTA
        RETURN
        END
CEND OF AS 64/109
```

```
      SUBROUTINE SFINT(ICODE, LCODE, KPOWER, IORDER, LIMIT,
     *  MAXLBP, LISTBP, LENBP, IFAULT)
C
C         AS R82 (REMARK ON AS65) APPL.STATIST. (1990), VOL.39, NO.1
C
C         Expands a structure formula into a list of binary integers
C
      INTEGER IFAULT,IORDER,KPOWER,LCODE,LENBP,LIMIT,MAXLBP
      INTEGER ICODE(LCODE),LISTBP(MAXLBP)
      INTEGER I,IBP,IBS,IJB,ILS,IORD,IORJ,IP,IPRONO,IREL,IRS,
     1        J,JB,JBP,JC,JLFAC,JLSUM,JLSYM,JMFAC,JMSUM,JMSYM,JOP,JP,
     2        JRFAC,JRSUM,JRSYM,JTFAC,JTSUM,JTSYM,K,LLSUM,M,NPOWER,NS
      LOGICAL LCHECK,RCHECK
C
C         Set NBI so that 2**NBI-1 is a large legal positive integer
C
      INTEGER NBI
      PARAMETER (NBI=31)
C
C         Set codes for K-symbols and L according to number of symbols
C         (PNSYM) and number N of precedence classes (PNPREC), also
C         codes for symbols
C
      INTEGER PDOT,PK0,PK2NP1,PK2NP2,PL,PLBRKT,PMINSL,PMINST,PMINUS,
     1        PNPREC,PNSYM,PPLUS,PRBRKT,PSLASH,PSTAR,PSTAR2
      PARAMETER (PSTAR2=1,PDOT=2,PSLASH=3,PSTAR=4,PPLUS=5,PMINUS=6,
     1 PMINSL=7,PMINST=8,PLBRKT=9,PRBRKT=10, PNPREC=5,PNSYM=10,
     2 PK0=PNSYM+1,PK2NP1=PK0+2*PNPREC+1,PK2NP2=PK2NP1+1,PL=PK2NP2+1)
C
C         Initialise compact precedence table and set auxiliary
C         quantities
C
      INTEGER KILLEG,KOPEN,KCON,KCLOSE
      PARAMETER (KILLEG=0,KOPEN=1,KCON=2,KCLOSE=3)
      INTEGER IPRTAB(5,5), IENTRY(PNSYM+2*PNPREC+4),
     1        IPREC(PNSYM+2*PNPREC+4)
C
      DATA IPRTAB /2*0,-1,-1,1, -3,3,3*0, 2*0,3*1, -3,3,3*0, 2*3,3*0/
      DATA IENTRY /8*4, 3, 2, 13*1, 5/
      DATA IPREC /2, 4, 6, 8, 4*10, 12, 11,
     1      0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13/
C
C     LITERAL =   **  .  /  *  +  - -/ -*   (  )    FACTOR
C     SYMBOL  =   O2 O4 O6 O8 10 10 10 10   (  )    K0 K1 ... K12(=K2N+2)   L
C     CODE    =    1  2  3  4  5  6  7  8   9 10    11 12 ...  23          24
C     IENTRY  =    4  4  4  4  4  4  4  4   3  2     1  1 ...   1           5
C     IPREC   =    2  4  6  8 10 10 10 10  12 11     0  1 ...  12          13
C
C-------------------INITIALISATION--------------------------------------
C         IORD=highest order of retained terms
C         JC=pointer to next input symbol in icode
C         JB=pointer to top of sum stack LISTBP(1...)
C         JP=pointer to top of symbol stack LISTBP(...MAXLBP) (top.down)
C         containing 3-integer cells (symbol code,pointer to SUM,FAC)
C
      IFAULT = 0
      LENBP = 0
      IORD = LIMIT
      IF (IORD.LE.0) IORD = NBI
      JC = 0
      IF (LCODE) 1001, 999, 10
```

```
   10 JB = 0
      JP = MAXLBP + 1
C
C------------------ SYNTAX ANALYSER ----------------------------------
C-------- Stack limit symbol L and/or next ICODE symbol
C         Stack -input code for factor
C
  100 I = PL
      IF (JC.LT.1.OR.JC.GE.LCODE) GOTO 115
  110 JC = JC + 1
      I = ICODE(JC)
      IF (I.EQ.0.OR.I.GT.PNSYM) GOTO 1004
  115 JP = JP - 3
      IF (JB.GE.JP) GOTO 1002
      LISTBP(JP + 2) = -I
      LISTBP(JP + 1) = 0
      IF (I.LT.0) I = PK0
      LISTBP(JP) = I
      IF (JC.EQ.0) GOTO 110
C
C-------- SEARCH FOR PRODUCTION
C         If closing relation between top two stacked symbols scan for
C         matching opening relation to delimit possible rightpart
C
  120 IP = JP
      IBS = 0
  130 IP = IP + 3
C
C         IREL=relation between leftsymbol at IP and rightsymbol at IP-3
C
      ILS = LISTBP(IP)
      IRS = LISTBP(IP - 3)
      IREL = IPRTAB(IENTRY(ILS), IENTRY(IRS))
      IF (IREL.GE.0) GOTO 140
      IREL = -IREL
      M = IPREC(ILS) - IPREC(IRS)
      IF (IREL.EQ.KOPEN) M = 1 - M
      IF (M.EQ.0) IREL = KCON
      IF (M.GT.0) IREL = KILLEG
C
C         Branch on IBS=0/1 for terminal/earlier relation
C
  140 IF (IBS.EQ.1) GOTO 170
C
C         Terminal relation
C
      IF (IREL.EQ.KILLEG) GOTO 1005
      IBS = 1
      IF (IREL - KCLOSE) 100, 130, 100
C
C         Earlier relation
C
  170 IF (IREL.NE.KOPEN) GOTO 130
C
C-------- IDENTIFY PRODUCTION
C         Leftpart  rightpart                  Semantic action  IPRONO
C         K1        K0                         load             1
C         K1        ( K2N+1 )                  copy             2
C         KI        KI OI KI-1 I=2,4,...,2N    dyadic operation 3
C         KI        KI-1       I=1,2,...,2N+1         last two productions
C         K2N+2   L K2N+1 L                           are shortcircuited
```

```
C         Identify production, if any, with rightpart matching the NS
C         stacked symbols between the opening and closing relations
C         Copy terminal + rightpart cells
C
      NS = (IP - JP - 3) /3
      IF (NS.NE.1.AND.NS.NE.3) GOTO 1005
      JTSYM = LISTBP(JP)
      JTSUM = LISTBP(JP + 1)
      JTFAC = LISTBP(JP + 2)
      JRSYM = LISTBP(JP + 3)
      JRSUM = LISTBP(JP + 4)
      JRFAC = LISTBP(JP + 5)
      IPRONO = 4
      IF (NS.EQ.3) GOTO 180
      IF (JRSYM.EQ.PK0) IPRONO = 1
      GOTO 190
  180 JMSYM = LISTBP(JP + 6)
      JMSUM = LISTBP(JP + 7)
      JMFAC = LISTBP(JP + 8)
      JLSYM = LISTBP(JP + 9)
      JLSUM = LISTBP(JP + 10)
      JLFAC = LISTBP(JP + 11)
      IF (JMSYM.EQ.PK2NP1.AND.JLSYM.EQ.PLBRKT.AND.JRSYM.EQ.PRBRKT)
     *  IPRONO = 2
      I = IPREC(JMSYM)
      IF (IENTRY(JMSYM).EQ.4.AND.
     *    JLSYM.EQ.PK0 + I.AND.JRSYM.EQ.PK0 + I - 1) IPRONO = 3
C
C         Branch to semantics section
C
  190 GOTO (500, 600, 700, 1005), IPRONO
C
C-------- UPDATE STACK
C         Set JRSYM=K-symbol in leftpart. anticipate and shortcircuit
C         chain of 1-symbol productions by promoting JRSYM until it
C         concatenates with a stacked neighbouring symbol
C
  200 JRSYM = PK0 + MIN(IPREC(LISTBP(IP)) - 1, IPREC(LISTBP(JP)))
C
C         If leftpart=K2N+2 (formula) set LENBP and return
C         else stack terminal + leftpart cells
C
      IF (JRSYM.LT.PK2NP2) GOTO 210
      LENBP = JB
      GOTO 999
  210 JP = IP - 6
      LISTBP(JP) = JTSYM
      LISTBP(JP + 1) = JTSUM
      LISTBP(JP + 2) = JTFAC
      LISTBP(JP + 3) = JRSYM
      LISTBP(JP + 4) = JRSUM
      LISTBP(JP + 5) = JRFAC
      GOTO 120
C
C------------------ SEMANTICS ----------------------------------------
C-------- LOAD
C
  500 IF (JRFAC.GT.KPOWER) THEN
        JRFAC = - MIN(IORD, JRFAC - KPOWER)
      ELSE
        IF (JRFAC.GT.NBI) GOTO 1003
```

```
          JRFAC = 2 ** (JRFAC - 1)
        ENDIF
        JRSUM = JB
        JB = JB + 1
        IF (JB.GE.JP) GOTO 1002
        LISTBP(JB) = JRFAC
        GOTO 200
C
C-------- COPY
  600 JRSYM = JMSYM
        JRSUM = JMSUM
        JRFAC = JMFAC
        GOTO 200
C
C-------- DYADIC OPERATION
C
  700 JOP = JMSYM
        IJB = JB
        LLSUM = JRSUM
        RCHECK = (JOP.EQ.PSTAR2).OR.(JOP.EQ.PSTAR)
        LCHECK = (JOP.EQ.PSLASH).OR.(JOP.EQ.PSTAR)
C
C            **    .     /     *     +     -     -/    -*
        GOTO (702, 730, 710, 730, 780, 720, 720, 720), JOP
  702 NPOWER = - LISTBP(JRSUM + 1)
        IF(NPOWER.LE.0) GOTO 1005
        JB = JRSUM
        JRFAC = JLFAC
        IF (NPOWER.EQ.1) GOTO 775
        JRSUM = JLSUM
        IJB = LLSUM
        GOTO 730
  710 JB = JRSUM
        IBP = JLFAC
        I = JRSUM
        GOTO 740
  720 JB = JLSUM
C
C         Outer loop over left operand
C
  730 I = JLSUM
  735 I = I + 1
        IF (I.GT.LLSUM) GOTO 770
        IBP = LISTBP(I)
        IF(IBP.LT.0) GOTO 1005
C
C         Inner loop over right operand
C
  740 J = JRSUM
  745 J = J + 1
        IF (J.GT.IJB) GOTO 760
        JBP = LISTBP(J)
        IF(JBP.LT.0) GOTO 1005
        IORJ = IOR(IBP, JBP)
        IF ((JOP.EQ.PMINUS).OR.(JOP.EQ.PMINSL).OR.(JOP.EQ.PMINST))GOTO 750
C
C         ** . / and * only
C
        IF ((LCHECK.AND.(IORJ.EQ.IBP)).OR.
       1    (RCHECK.AND.(IORJ.EQ.JBP)).OR.
       2    (ICNT(IORJ).GT.IORD)) GOTO 745
```

```
      JB = JB + 1
      IF (JB.GE.JP) GOTO 1002
      LISTBP(JB) = IORJ
      GOTO 745
C
C         - -/ and -* only
C
  750 IF (IORJ.NE.IBP) GOTO 745
      K = PMINUS
      IF (IBP.EQ.JBP) K = PMINSL
      IF (JOP - K) 735, 745, 735
C
C         End of inner loop
C         Copy integer not deleted in - -/ or -* operation
C
  760 IF ((JOP.NE.PMINUS).AND.(JOP.NE.PMINSL).AND.(JOP.NE.PMINST))
     1  GOTO 735
      JB = JB + 1
      LISTBP(JB) = IBP
      GOTO 735
C
C         End of outer loop
C         For .  /  and * reorder sum and copy omitting repeats
C
  770 IF ((JOP.EQ.PMINUS).OR.(JOP.EQ.PMINSL).OR.(JOP.EQ.PMINST))GOTO 790
      IF (JOP.NE.PSTAR2) GOTO 775
      NPOWER = NPOWER - 1
      IF(NPOWER.EQ.1) GO TO 775
      J = IJB
      GO TO 776
  775 IF (JOP.NE.PDOT) IJB = JLSUM
      J = JLSUM
  776 CALL BISORT(LISTBP, IJB, JB, IORDER)
      IBP = -1
      DO 779 I = IJB + 1, JB
      IF (LISTBP(I).EQ.IBP) GOTO 779
      IBP = LISTBP(I)
      J = J + 1
      LISTBP(J) = IBP
  779 CONTINUE
      JB = J
      IF (JOP.NE.PSTAR2) GO TO 790
      IF (NPOWER.EQ.1) GO TO 790
C         For ** form next power
      JRSUM = IJB
      IJB = JB
      GO TO 730
C
C         + merge right with left sum
C
  780 JB = JRSUM
      DO 784 J = JRSUM + 1, IJB
      JBP = LISTBP(J)
      DO 782 I = JLSUM + 1, LLSUM
      IF(JBP.EQ.LISTBP(I)) GO TO 784
  782 CONTINUE
      JB = JB + 1
      LISTBP(JB) = JBP
  784 CONTINUE
C
C         Set JRSUM, JRFAC
```

```
C
  790 JRSUM = JLSUM
      JRFAC = IOR(JLFAC, JRFAC)
      IF ((JOP.EQ.PMINUS).OR.(JOP.EQ.PMINSL).OR.(JOP.EQ.PMINST))
     1  JRFAC = JLFAC
      GOTO 200
C
C------------------ RETURN AND ERROR SETTING --------------------------
C
 1005 IFAULT = IFAULT + 1
 1004 IFAULT = IFAULT + 1
 1003 IFAULT = IFAULT + 1
 1002 IFAULT = IFAULT + 1
 1001 IFAULT = IFAULT + 1
      LENBP = JC
  999 RETURN
      END
```

```
c This file includes the Applied Statistics algorithm AS 66 for calculating
c the tail area under the normal curve, and two alternative routines which
c give higher accuracy.   The latter have been contributed by Alan Miller of
c CSIRO Division of Mathematics & Statistics, Clayton, Victoria.   Notice
c that each function or routine has different call arguments.
c
c
      double precision function alnorm(x,upper)
c
c         Algorithm AS66 Applied Statistics (1973) vol22 no.3
c
c       Evaluates the tail area of the standardised normal curve
c       from x to infinity if upper is .true. or
c       from minus infinity to x if upper is .false.
c
      double precision zero,one,half
      double precision con,z,y,x
      double precision p,q,r,a1,a2,a3,b1,b2,c1,c2,c3,c4,c5,c6
      double precision d1,d2,d3,d4,d5
      logical upper,up
c*** machine dependent constants
      double precision ltone,utzero
      data zero/0.0d0/, one/1.0d0/, half/0.5d0/
      data ltone/7.0d0/,utzero/18.66d0/
      data con/1.28d0/
      data p/0.398942280444d0/,q/0.39990348504d0/,r/0.398942280385d0/
      data a1/5.75885480458d0/,a2/2.62433121679d0/,a3/5.92885724438d0/
      data b1/-29.8213557807d0/,b2/48.6959930692d0/
      data c1/-3.8052d-8/,c2/3.98064794d-4/,c3/-0.151679116635d0/
      data c4/4.8385912808d0/,c5/0.742380924027d0/,c6/3.99019417011d0/
      data d1/1.00000615302d0/,d2/1.98615381364d0/,d3/5.29330324926d0/
      data d4/-15.1508972451d0/,d5/30.789933034d0/
c
      up=upper
      z=x
      if(z.ge.zero)goto 10
      up=.not.up
      z=-z
   10 if(z.le.ltone.or.up.and.z.le.utzero)goto 20
      alnorm=zero
      goto 40
   20 y=half*z*z
      if(z.gt.con) goto 30
c
      alnorm=half-z*(p-q*y/(y+a1+b1/(y+a2+b2/(y+a3))))
      goto 40
   30 alnorm=r*dexp(-y)/(z+c1+d1/(z+c2+d2/(z+c3+d3/(z+c4+d4/(z+c5+d5/
     2   (z+c6))))))
   40 if(.not.up)alnorm=one-alnorm
      return
      end
c
c
c
        SUBROUTINE NORMP(Z, P, Q, PDF)
C
C       Normal distribution probabilities accurate to 1.e-15.
C       Z = no. of standard deviations from the mean.
C       P, Q = probabilities to the left & right of Z.   P + Q = 1.
C       PDF = the probability density.
C
```

```
C       Based upon algorithm 5666 for the error function, from:
C       Hart, J.F. et al, 'Computer Approximations', Wiley 1968
C
C       Programmer: Alan Miller
C
C       Latest revision - 30 March 1986
C
        IMPLICIT DOUBLE PRECISION (A-H, O-Z)
        DATA P0, P1, P2, P3, P4, P5, P6/220.20 68679 12376 1D0,
     *    221.21 35961 69931 1D0, 112.07 92914 97870 9D0,
     *    33.912 86607 83830 0D0, 6.3739 62203 53165 0D0,
     *    .70038 30644 43688 1D0, .35262 49659 98910 9D-01/,
     *    Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7/440.41 37358 24752 2D0,
     *    793.82 65125 19948 4D0, 637.33 36333 78831 1D0,
     *    296.56 42487 79673 7D0, 86.780 73220 29460 8D0,
     *    16.064 17757 92069 5D0, 1.7556 67163 18264 2D0,
     *    .88388 34764 83184 4D-1/,
     *    CUTOFF/7.071D0/, ROOT2PI/2.5066 28274 63100 1D0/
C
        ZABS = ABS(Z)
C
C       |Z| > 37.
C
        IF (ZABS .GT. 37.D0) THEN
          PDF = 0.D0
          IF (Z .GT. 0.D0) THEN
            P = 1.D0
            Q = 0.D0
          ELSE
            P = 0.D0
            Q = 1.D0
          END IF
          RETURN
        END IF
C
C       |Z| <= 37.
C
        EXPNTL = EXP(-0.5D0*ZABS**2)
        PDF = EXPNTL/ROOT2PI
C
C       |Z| < CUTOFF = 10/sqrt(2).
C
        IF (ZABS .LT. CUTOFF) THEN
          P = EXPNTL*((((((P6*ZABS + P5)*ZABS + P4)*ZABS + P3)*ZABS +
     *          P2)*ZABS + P1)*ZABS + P0)/(((((((Q7*ZABS + Q6)*ZABS +
     *          Q5)*ZABS + Q4)*ZABS + Q3)*ZABS + Q2)*ZABS + Q1)*ZABS +
     *          Q0)
C
C       |Z| >= CUTOFF.
C
        ELSE
          P = PDF/(ZABS + 1.D0/(ZABS + 2.D0/(ZABS + 3.D0/(ZABS + 4.D0/
     *          (ZABS + 0.65D0)))))
        END IF
C
        IF (Z .LT. 0.D0) THEN
          Q = 1.D0 - P
        ELSE
          Q = P
          P = 1.D0 - Q
        END IF
```

```
      RETURN
      END
C
C
C
      SUBROUTINE NPROB(Z,P,Q,PDF)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C
C     P, Q = PROBABILITIES TO THE LEFT AND RIGHT OF Z
C     FOR THE STANDARD NORMAL DISTRIBUTION.
C     PDF  = THE PROBABILITY DENSITY FUNCTION
C
C     REFERENCE: ADAMS,A.G. AREAS UNDER THE NORMAL CURVE,
C     ALGORITHM 39, COMPUTER J., VOL. 12, 197-8, 1969.
C
C     LATEST REVISION - 23 JANUARY 1981
C
C*****************************************************************
C
      DATA A0,A1,A2,A3,A4,A5,A6,A7/0.5D0, 0.398942280444D0,
     1  0.399903438504D0, 5.75885480458D0, 29.8213557808D0,
     2  2.62433121679D0, 48.6959930692D0, 5.92885724438D0/,
     3  B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11/0.398942280385D0,
     4  3.8052D-8, 1.00000615302D0, 3.98064794D-4, 1.98615381364D0,
     5  0.151679116635D0, 5.29330324926D0, 4.8385912808D0,
     6  15.1508972451D0, 0.742380924027D0, 30.789933034D0,
     7  3.99019417011D0/
C
      ZABS = ABS(Z)
      IF(ZABS.GT.12.7D0) GO TO 20
      Y = A0*Z*Z
      PDF = EXP(-Y)*B0
      IF(ZABS.GT.1.28D0) GO TO 10
C
C     Z BETWEEN -1.28 AND +1.28
C
      Q = A0-ZABS*(A1-A2*Y/(Y+A3-A4/(Y+A5+A6/(Y+A7))))
      IF(Z.LT.0.D0) GO TO 30
      P = 1.D0-Q
      RETURN
C
C     ZABS BETWEEN 1.28 AND 12.7
C
   10 Q = PDF/(ZABS-B1+B2/(ZABS+B3+B4/(ZABS-B5+B6/(ZABS+B7-B8/
     1 (ZABS+B9+B10/(ZABS+B11))))))
      IF(Z.LT.0.D0) GO TO 30
      P = 1.D0-Q
      RETURN
C
C     Z FAR OUT IN TAIL
C
   20 Q = 0.D0
      PDF = 0.D0
      IF(Z.LT.0.D0) GO TO 30
      P = 1.D0
      RETURN
C
C     NEGATIVE Z, INTERCHANGE P AND Q
C
   30 P = Q
      Q = 1.D0-P
```

```
      RETURN
      END
```

```
      SUBROUTINE INCLUD (NP,NRBAR,WEIGHT,XROW,YELEM,D,RBAR,THETAB,SSERR,
     1 IFAULT)
C
C     Algorithm AS 75.1 Appl. Statist. (1974) Vol.23, No.3, p448
C
C     Calling this subroutine updates d, rbar, thetab and sserr
C     by the inclusion of xrow, yelem with specified weight.
C
      DOUBLE PRECISION XROW(NP),D(NP),RBAR(NRBAR),THETAB(NP),WEIGHT,
     1 SSERR,CBAR,DI,DPI,SBAR,W,XI,XK,Y,ZERO,YELEM
C
      DATA ZERO/0.0D0/
C
C        check input parameters
C
      IFAULT=1
      IF (NP.LT.1.OR.NRBAR.LE.NP*(NP-1)/2) RETURN
      IFAULT=0
C
      W=WEIGHT
      Y=YELEM
      DO 30 I=1,NP
C
C     Skip unnecessary transformations.  Test on exact zeros must
C     be used or stability can be destroyed.
C
        IF (W.EQ.ZERO) RETURN
        IF (XROW(I).EQ.ZERO) GO TO 30
        XI=XROW(I)
        DI=D(I)
        DPI=DI+W*XI*XI
        CBAR=DI/DPI
        SBAR=W*XI/DPI
        W=CBAR*W
        D(I)=DPI
        IF (I.EQ.NP) GO TO 20
        NEXTR=(I-1)*(NP+NP-I)/2+1
        IP=I+1
        DO 10 K=IP,NP
          XK=XROW(K)
          XROW(K)=XK-XI*RBAR(NEXTR)
          RBAR(NEXTR)=CBAR*RBAR(NEXTR)+SBAR*XK
          NEXTR=NEXTR+1
 10     CONTINUE
 20     XK=Y
        Y=XK-XI*THETAB(I)
        THETAB(I)=CBAR*THETAB(I)+SBAR*XK
 30   CONTINUE
      SSERR=SSERR+W*Y*Y
      RETURN
      END
C
C
      SUBROUTINE CONFND (NP,NRBAR,J,RBAR,CONTRA,IFAULT)
C
C     Algorithm AS 75.2 Appl. Statist. (1974) Vol. 23, No. 3, P448
C
C     Calling this subroutine obtains the contrast which could not
C     be estimated if D(j) were assumed to be zero, that is, obtains the
C     linear combination of the first j columns which would be zero.  Th
C     obtained by setting the first j-1 elements of contra to the soluti
```

```
C     of the triangular system formed by the first j-1 rows and columns
C     rbar with the first j-1 elements of the jth column as right hand
C     side, setting the jth element of contra to -1, and setting the
C     remaining elements of contra to zero.
C
      DOUBLE PRECISION RBAR(NRBAR),CONTRA(NP),ZERO,ONE
C
      DATA ZERO/0.0D0/,ONE/1.0D0/
C
C         check input parameters
C
      IFAULT=1
      IF (NP.LT.1.OR.NRBAR.LE.NP*(NP-1)/2) RETURN
      IFAULT=0
C
      JM=J-1
      IF (J.EQ.NP) GO TO 20
      JP=J+1
      DO 10 I=JP,NP
 10     CONTRA(I)=ZERO
 20   CONTRA(J)=-ONE
      IF (J.EQ.1) RETURN
      DO 40 IJ=1,JM
        I=J-IJ
        NEXTR=(I-1)*(NP+NP-I)/2+1
        K=NEXTR+J-I-1
        CONTRA(I)=RBAR(K)
        IF (I.EQ.JM) GO TO 40
        IP=I+1
        DO 30 K=IP,JM
          CONTRA(I)=CONTRA(I)-RBAR(NEXTR)*CONTRA(K)
          NEXTR=NEXTR+1
 30     CONTINUE
 40   CONTINUE
      RETURN
      END
C
C
      SUBROUTINE SSDCMP (NP,D,THETAB,SS,IFAULT)
C
C     Algorithm AS75.3  Appl. Statist. (1974) Vol.23, No. 3, P448
C
C     Calling this subroutine computes the np components of the sum
C     of squares decomposition from D and thetab.
C
      DOUBLE PRECISION D(NP),THETAB(NP),SS(NP)
C
C        check input parameters
C
      IFAULT=1
      IF (NP.LT.1) RETURN
      IFAULT=0
C
      DO 10 I=1,NP
 10     SS(I)=D(I)*THETAB(I)**2
      RETURN
      END
C
C
      SUBROUTINE REGRSS (NP,NRBAR,RBAR,THETAB,BETA,IFAULT)
C
```

```
C     Algorithm AS 75.4  Appl. Statist. (1974), Vol. 23, No. 3, p448
C
C     Calling this subroutine obtains beta by back-substitution in
C     the triangular system rbar and thetab.
C
      DOUBLE PRECISION RBAR(NRBAR),THETAB(NP),BETA(NP)
C
C        check input parameters
C
      IFAULT=1
      IF (NP.LT.1.OR.NRBAR.LE.NP*(NP-1)/2) RETURN
      IFAULT=0
C
      DO 20 J=1,NP
        I=NP-J+1
        BETA(I)=THETAB(I)
        NEXTR=(I-1)*(NP+NP-I)/2+1
        IP=I+1
        DO 10 K=IP,NP
          BETA(I)=BETA(I)-RBAR(NEXTR)*BETA(K)
          NEXTR=NEXTR+1
 10     CONTINUE
 20   CONTINUE
      RETURN
      END
C
C
      SUBROUTINE ANALYZ (MAXP,MAXR,NP,NRBAR,INPUT,OUTPUT,X,DD,THETA,R,
     1 IFAULT)
C
C        Algorithm AS 75.5  Appl. Statist. (1974) Vol. 23, P. 448
C
C        This subroutine reads data from channel input and
C        produces least squares variance component analyses on
C        channel output.  The input medium is assumed to
C        contain first a value for the integer np, the number of
C        independent variates, and then a sequence of records
C        containing the rows of x and y together with weights.
C        The number zero (read as a weight) will indicate the end of
C        each data set, and upon reading this an analysis will be
C        produced.  Successive analyses will be done until np,
C        the number of independent variates, is read as zero.
C
C        No need to check the ifault parameter after calling
C        auxiliary routines as parameters have already been
C        checked here.
C
      INTEGER OUTPUT
      LOGICAL FIRST
      DOUBLE PRECISION X(MAXP),DD(MAXP),THETA(MAXP),R(MAXR),ERROR,W,Y,
     1 ZERO,ONE,EPS,TOL
C
      DATA ZERO,ONE,EPS,TOL/0.0D0,1.0D0,1.0D-8,1.0D-4/
C
C        input formats
C
 10   FORMAT (I3)
 20   FORMAT (F10.6)
 30   FORMAT (F10.6)
 40   FORMAT (I3)
 50   FORMAT (F10.6)
```

```
C
C        output formats
C
 60   FORMAT (1H1,I5,18H observations read/19H diagonal matrix is/(1X,
     1 5G15.8))
 70   FORMAT (21H confounded contrasts)
 80   FORMAT (1X,5G15.8)
 90   FORMAT (29H sum of squares decomposition/(1X,5G15.8))
 100  FORMAT (22H sum of squares error ,G15.8)
 110  FORMAT (24H regression coefficients,(1X,5G15.8))
C
C        read np, check its value and initialise arrays
C
      IFAULT=0
 120  READ (INPUT,10) NP
      IF (NP.EQ.0) RETURN
      NRBAR=NP*(NP-1)/2
      IF (NP.LT.0.OR.NP.GT.MAXP.OR.NRBAR.GT.MAXR) GO TO 240
      DO 130 K=1,NP
        DD(K)=ZERO
        THETA(K)=ZERO
 130  CONTINUE
      DO 140 K=1,NRBAR
 140    R(K)=ZERO
      ERROR=ZERO
      N=0
C
C        read weight and check its value
C
 150  READ (INPUT,20) W
      IF (W.EQ.ZERO) GO TO 190
      IF (W.GT.ZERO) N=N+1
      IF (W.LT.ZERO) N=N-1
C
C        read the y-value and all corresponding x-values
C
      READ (INPUT,30) Y
      DO 160 K=1,NP
 160    X(K)=ZERO
 170  READ (INPUT,40) K
      IF (K.LT.0.OR.K.GT.NP) GO TO 250
      IF (K.EQ.0) GO TO 180
      READ (INPUT,50) X(K)
      GO TO 170
 180  CALL INCLUD (NP,NRBAR,W,X,Y,DD,R,THETA,ERROR,IFAIL)
      GO TO 150
C
C        begin output
C
 190  WRITE (OUTPUT,60) N,(DD(I),I=1,NP)
      FIRST=.TRUE.
      DO 230 J=1,NP
        IF (ABS(DD(J)).GE.EPS) GO TO 230
C
C         confounding discovered
C
        IF (.NOT.FIRST) GO TO 200
        FIRST=.FALSE.
        WRITE (OUTPUT,70)
 200    CALL CONFND (NP,NRBAR,J,R,X,IFAIL)
        WRITE (OUTPUT,80) (X(I),I=1,NP)
```

```
C
C        choose resolving constraint
C
         IF (J.EQ.1) GO TO 220
         JM=J-1
         DO 210 K=1,JM
           IF (ABS(X(K)).LE.TOL) GO TO 210
           X(K)=ZERO
           GO TO 220
 210     CONTINUE
 220     CALL INCLUD (NP,NRBAR,ONE,X,ZERO,DD,R,THETA,ERROR,IFAIL)
 230   CONTINUE
       CALL SSDCMP (NP,DD,THETA,X,IFAIL)
       WRITE (OUTPUT,90) (X(I),I=1,NP)
       WRITE (OUTPUT,100) ERROR
       CALL REGRSS (NP,NRBAR,R,THETA,X,IFAIL)
       WRITE (OUTPUT,110) (X(I),I=1,NP)
       GO TO 120
C
C        error returns
C
 240   IFAULT=2
       RETURN
 250   IFAULT=3
       RETURN
       END
```

```
C Two versions of algorithm AS 76 are given here; the original with one
C correction incorporated, and AS R55, also amended.   AS R55 requires
C AS 76.   N.B. The accuracy of AS 76 could be increased by using more
C Gaussian quadrature points, or better, by using Hermite integration.
C
      FUNCTION TFN(X, FX)
C
C     ALGORITHM AS 76  APPL. STATIST. (1974) VOL.23, NO.3
C
C     Calculates the T-function of Owen, using Gaussian quadrature.
C     Incorporates correction AS R30 (vol.28, no.1, 1979)
C
      REAL U(5), R(5)
C
      DATA U /0.0744372, 0.2166977, 0.3397048, 0.4325317, 0.4869533/
      DATA R /0.1477621, 0.1346334, 0.1095432, 0.0747257, 0.0333357/
      DATA NG,    TP,    TV1,     TV2,     TV3,     TV4
     *   /  5, 0.159155, 1.E-35,  15.0,    15.0,   1.E-5 /
      DATA ZERO, QUART, HALF, ONE,  TWO
     *   / 0.0,  0.25,  0.5,  1.0,  2.0 /
C
C     Test for X near zero
C
      IF (ABS(X) .GE. TV1) GO TO 5
      TFN = TP * ATAN(FX)
      RETURN
C
C     Test for large values of abs(X)
C
    5 IF (ABS(X) .GT. TV2) GO TO 10
C
C     Test for FX near zero
C
      IF (ABS(FX) .GE. TV1) GO TO 15
   10 TFN = ZERO
      RETURN
C
C     Test whether abs(FX) is so large that it must be truncated
C
   15 XS = -HALF * X * X
      X2 = FX
      FXS = FX * FX
      IF (LOG(ONE + FXS) - XS * FXS .LT. TV3) GO TO 25
C
C     Computation of truncation point by Newton iteration
C
      X1 = HALF * FX
      FXS = QUART * FXS
   20 RT = FXS + ONE
      X2 = X1 + (XS * FXS + TV3 - LOG(RT)) / (TWO * X1 * (ONE/RT - XS))
      FXS = X2 * X2
      IF (ABS(X2 - X1) .LT. TV4) GO TO 25
      X1 = X2
      GO TO 20
C
C     Gaussian quadrature
C
   25 RT = ZERO
      DO 30 I = 1, NG
        R1 = ONE + FXS * (HALF + U(I))**2
        R2 = ONE + FXS * (HALF - U(I))**2
```

```
          RT = RT + R(I) * (EXP(XS * R1) / R1 + EXP(XS * R2) / R2)
   30 CONTINUE
      TFN = RT * X2 * TP
C
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      REAL FUNCTION THA(H1, H2, A1, A2)
C
C     AS R55  APPL. STATIST. (1985) VOL.34, NO.1
C
C     A remark on AS 76
C     Incorporating improvements in AS R80 (Appl. Statist. (1989)
C     vol.38, no.3), and AS R89 (Appl. Statist. (1992) vol.41, no.2).
C
C     Computes T(H1/H2, A1/A2) for any real numbers H1, H2, A1 and A2
C
C     Auxiliary function required: ALNORM (= AS 66) and AS 76
C
      REAL A, ALNORM, A1, A2, G, H, H1, H2, TFN, ABSA, AH, GH, GAH,
     *  TWOPI, LAM, EX, C1, C2,
     *  AH, ZERO, ONE, TWO, PT3, SEVEN, HALF, SIX, QUART
C
      DATA TWOPI /6.2831853/, ZERO /0.0/, ONE /1.0/, TWO /2.0/,
     *   PT3 /0.3/, SEVEN /7.0/, HALF /0.5/, SIX /6.0/, QUART /0.25/
C
      IF (H2 .NE. ZERO) GO TO 1
      THA = ZERO
      RETURN
C
    1 H = H1 / H2
      IF (A2 .EQ. ZERO) GO TO 2
      A = A1 / A2
      IF ((ABS(H) .LT. PT3) .AND. (ABS(A) .GT. SEVEN)) GO TO 6
C
C     Correction AS R89
C
      ABSA = ABS(A)
      IF (ABSA .GT. ONE) GO TO 7
      THA = TFN(H, A)
      RETURN
    7 AH = ABSA * H
      GH = ALNORM(H, .FALSE.)
      GAH = ALNORM(AH, .FALSE.)
      THA = HALF * (GH + GAH) - GH * GAH - TFN(AH, ONE/ABSA)
      IF (A .LT. ZERO) THA = - THA
      RETURN
C
    2 G = ALNORM(H, .FALSE.)
      IF (H .GE. ZERO) GO TO 3
      THA = G / TWO
      GO TO 4
    3 THA = (ONE - G) / TWO
    4 IF (A1 .GE. ZERO) RETURN
      THA = -THA
      RETURN
C
    6 LAM = ABS(A * H)
      EX = EXP(-LAM * LAM / TWO)
```

```
      G = ALNORM(LAM, .FALSE.)
      C1 = (EX/LAM + SQRT(TWOPI) * (G - HALF)) / (TWOPI)
      C2 = ((LAM * LAM + TWO) * EX/LAM**3 + SQRT(TWOPI) * (G - HALF))
     *       / (SIX * TWOPI)
      AH = ABS(H)
      THA = QUART - C1 * AH + C2 * AH**3
      THA = SIGN(THA, A)
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION ROY TST(NN1, NN2, NNP, X, IFAULT)
C
C     ALGORITHM AS 77  APPL. STATIST. (1974) VOL.23, NO.3
C
C     A function for the exact null distribution of the largest root
C     of a beta matrix.
C
C     Restrictions:
C       The value of NN2 - NNP - 1 must be an even number.
C       Local storage is sufficient for min(NN1, NNP) at most 10.
C       The number of terms is limited to at most 150,000.
C
      INTEGER ND(10), NP(10), K(56), R, P, PMAX, TERMS
      DOUBLE PRECISION T(10), S(10), A, B, C, CX, CXS, HALF, HCX, ONE,
     +  P1, P2, TL, TWO, X, XM, XMM, XMN2, XN1, XN11, ZERO
      LOGICAL C1, C2
      DATA ZERO, HALF, ONE, TWO /0.D0, 0.5D0, 1.D0, 2.D0/
      DATA PMAX, TERMS /10, 150000/
C
C     Check for errors in the parameters.
C     Commence initialisation.
C
      N1 = MAX(NN1, NNP)
      N2 = MIN(NN2, NN1 + NN2 - NNP)
      P = MIN(NN1, NNP)
      IFAULT = 0
      ROY TST = ZERO
      IF (N1 .GT. 0 .AND. N2 .GT. 0 .AND. P .GT. 0 .AND. X .GE. ZERO
     +  .AND. X .LE. ONE) GO TO 10
      IFAULT = 1
      RETURN
   10 M = (N2 - P - 1) / 2
      IF (M .GE. 0 .AND. M + M .EQ. N2 - P - 1) GO TO 20
      IFAULT = 2
      RETURN
   20 P1 = ONE
      P2 = ZERO
      IF (M .EQ. 0) GO TO 160
      A = ONE
      B = M + P
      R = MIN(M, P)
      C = R
      DO 30 J = 1, R
        A = A * B / C
        B = B - ONE
        C = C - ONE
   30 CONTINUE
      R = A
      IF (P .LE. PMAX .AND. R .LE. TERMS) GO TO 40
      IFAULT = 3
      RETURN
C
C     Error checks complete; initialisation continues.
C
   40 XM = P
      XN11 = N1 - 1
      CX = ONE - X
      P2 = ONE
      DO 50 I = 1, P
        A = -I
        DO 50 J = 1, M
```

```
            A = A + TWO
            P2 = P2 * CX * (XN11 + A) / (XM + A)
     50 CONTINUE
        S(1) = P2
        MP = M * P
        MN = MP - 2
        IF (MN .LT. 0) GO TO 160
        R = 0
        L = 0
        N = 1
        LE = 1
        K(1) = M
        XMM = P + P + 1 - N1 - N2
        XMN2 = P - N2
        XN1 = N1
        HCX = HALF * CX
        CXS = ONE / (CX * CX)
        T(1) = ONE
        GO TO 80
C
C       Initialisation finished.   Main loop starts here.
C
C       Check if the current partition cannot have a unit part appended.
C
     60 C1 = (R .EQ. P)
C
C       Check if the current partition cannot have its last part
C       increased.
C
        C2 = (K(LE) .EQ. K(LE-1))
        IF (C1 .AND. C2) GO TO 150
        IF (C1) GO TO 100
        IF (C2) GO TO 90
        LB = LE - R + 1
        DO 70 I = LB, LE
          J = I + R
          K(J) = K(I)
     70 CONTINUE
        S(L+1) = S(L)
        T(L+1) = T(L)
C
C       Signal that a second partition is waiting in the list.
C
        ASSIGN 80 TO NEXT
        GO TO 100
     80 ASSIGN 140 TO NEXT
        L = L + 1
        LE = LE + R
     90 LE = LE + 1
        R = R + 1
        NP(L) = R
        K(LE) = 0
    100 K(LE) = K(LE) + 1
        ND(L) = N
C
C       Use the recurrence formulae for the terms of the current partition
C
        B = 2 * K(LE) - R - 1
        A = K(LE)
        TL = T(L)
        TL = TL * HCX * (XN1 + B) * (XM + B) / (A * (A + A - ONE))
```

```
      JP = R - 1
      IF (JP .EQ. 0) GO TO 130
      LB = LE - JP
      C = R + 2
      DO 120 J = 1, JP
        C = C - ONE
        TL = TL * (ONE - TWO / (TWO * (K(LB) - A) + C))
        LB = LB + 1
  120 CONTINUE
  130 P1 = P1 + TL
      T(L) = TL
C
C     Check if conjugate partition terms are necessary.
C
      IF (MN .EQ. 0) GO TO NEXT, (80, 140)
      B = B + ONE
      S(L) = S(L) * CXS * (XMN2 + B) * (XM + B) / ((XMM + B) *
     +                  (XN11 + B))
      P2 = P2 + TL * S(L)
C
C     Check if a second partition has been generated.
C
      GO TO NEXT, (80, 140)
  140 N = ND(L) + 1
      R = NP(L)
C
C     Check that the degree of the partition is within limits.
C
      MN = MP - N - N
      IF (MN .GE. 0) GO TO 60
  150 L = L - 1
      LE = LE - R
C
C     If L = 0 then list is empty and calculation complete.
C
      IF (L .GT. 0) GO TO 140
C
C     Multiply by the external factor, and return
C
  160 ROY TST = (P1 + P2) * X ** (HALF * (N1 * P))
      RETURN
      END
```

```
      SUBROUTINE MEDIAN(X, Y, N, IP, IT, IFAULT)
C
C     ALGORITHM AS 78  APPL. STATIST. (1974) VOL.23, NO.3
C
C     The mediancentre, generalising the median, is the point with
C     minimum total distance from a set of multivariate samples
C
      REAL X(N, IP), Y(IP)
      REAL Z(50), LAMBDA, LEPSI, LEPSR, LEPSD
C
      DOUBLE PRECISION C(50), COMP, D, DD, DELTA, EPSR, EPSD, SLAM,
     +  ZERO, ONE
C
      DATA LEPSD /0.0001/, LEPSR / 0.00001/, LEPSI /0.000001/
      DATA ICOUNT /100/, LCOUNT /50/
      DATA ZERO /0.D0/, ONE /1.D0/
C
C     Initial settings
C
      IFAULT = 0
      LL = 0
      II = 1
      IF (N .EQ. 1) GO TO 25
      IF (N .LE. 0 .OR. IP .LE. 0) GO TO 5
C
C     Calculate the diameter
C
      DIAM = 0.0
      DO 2 I = 2, N
        DO 2 J = 1, I-1
          S = 0.0
          DO 1 K = 1, IP
    1     S = S + (X(I,K) - X(J,K))**2
          DIAM = MAX(S, DIAM)
    2 CONTINUE
      DIAM = SQRT(DIAM)
      EPSR = LEPSR * DIAM
      EPSI = LEPSI * DIAM
      EPSD = LEPSD * DIAM
C
C     Initial median centre = the centroid
C
      U1 = 1.0 / FLOAT(N)
      DO 4 J = 1, IP
        S = 0.0
        DO 3 I = 1, N
    3   S = S + X(I,J)
        Y(J) = S * U1
    4 CONTINUE
      IT = ICOUNT
      IF (IP .LE. 50) GO TO 6
    5 IFAULT = 1
      IT = 0
      RETURN
C
C     Main iterative loop
C
    6 DO 23 L = 1, ICOUNT
C
C     Direction cosines and resultant
C
```

```
           CORNER = 0.0
           DO 7 J = 1, IP
       7   C(J) = ZERO
           LL = L
           DO 11 I = 1, N
             D = ZERO
             DO 8 J = 1, IP
       8     D = D + (X(I,J) - Y(J))**2
             DD = SQRT(D)
             IF (DD .GT. EPSD) GO TO 9
             CORNER = CORNER + 1.0
             II = I
             GO TO 11
       9     D = ONE / DD
             DO 10 J = 1, IP
      10     C(J) = C(J) + (X(I,J) - Y(J)) * D
      11   CONTINUE
           D = ZERO
           DO 12 J = 1, IP
      12   D = D + C(J)**2
           D = SQRT(D)
           DD = D
C
C     Tests for zero resultant or degenerate solution
C
           IF (CORNER .EQ. 0.0) GO TO 13
           IF (D .LE. CORNER) GO TO 25
           D = D - CORNER
      13   IF (D .LE. EPSR) GO TO 24
           DD = ONE / DD
           DO 14 J = 1, IP
      14   C(J) = C(J) * DD
C
C     Step by bisection to give zero component at lambda
C
           U1 = 0.0
           U2 = DIAM
           DO 20 LC = 1, LCOUNT
             COMP = ZERO
             LAMBDA = 0.5 * (U1 + U2)
             SLAM = LAMBDA * LAMBDA
             DO 15 J = 1, IP
      15     Z(J) = Y(J) + LAMBDA * C(J)
             DO 17 I = 1, N
               DELTA = ZERO
               D = SLAM
               DO 16 J = 1, IP
                 XX = X(I,J)
                 D = D - (XX - Y(J))**2
                 DELTA = DELTA + (XX - Z(J))**2
      16       CONTINUE
               DD = SQRT(DELTA)
               IF (DD .LT. EPSD) GO TO 21
               COMP = COMP - (D + DELTA) / DD
      17     CONTINUE
             IF (COMP .GT. ZERO) GO TO 18
             U2 = LAMBDA
             GO TO 19
      18     U1 = LAMBDA
      19     IF ((U2 - U1) .LE. EPSI) GO TO 21
      20   CONTINUE
```

```
C
   21   DO 22 J = 1, IP
   22   Y(J) = Y(J) + C(J) * LAMBDA
   23 CONTINUE
C
   24 IT = LL
      RETURN
   25 IT = -LL
      DO 26 J = 1, IP
   26 Y(J) = X(II,J)
      RETURN
      END
```

```
      subroutine fastf(xreal, ximag, isize, itype)
c
c       Algorithm AS 83.1 Appl. Statist. (1975) vol.24, no.1
c
c       Radix 4 complex discrete fast Fourier transform with
c       unscrambling of the transformed arrays.
c
      real    xreal(isize), ximag(isize)
c
c       Check for valid transform size.
c
      ii = 4
      do 2 k = 2, 20
        if (ii .eq. isize) go to 4
        if (ii .gt. isize) go to 3
        ii = ii * 2
    2 continue
c
c       If this point is reached a size error has occurred.
c
    3 return
c
c       Call FASTG to perform the transform.
c
    4 call fastg(xreal, ximag, isize, itype)
c
c       Call SCRAM to unscramble the results.
c
      call scram(xreal, ximag, isize, k)
c
      return
      end
c
c
c
      subroutine fastg(xreal, ximag, n, itype)
c
c       Algorithm AS 83.2 Appl. Statist. (1975) vol.24, no.1
c
c       Radix 4 complex discrete fast Fourier transform without
c       unscrambling, suitable for convolutions or other applications
c       which do not require unscrambling.   Called by subroutine
c       FASTF which also does the unscrambling.
c
      real    xreal(n), ximag(n)
      data    zero, half, one, one5, two, four
     +          /0.0,  0.5, 1.0,  1.5, 2.0,  4.0/
      pi = four * atan(one)
      ifaca = n / 4
      if (itype .eq. 0) return
      if (itype .gt. 0) go to 5
c
c       ITYPE < 0 indicates inverse transform required.
c       Calculate conjugate.
c
      do 4 k = 1, n
    4 ximag(k) = -ximag(k)
c
c       Following code is executed for IFACA = N/4, N/16, N/64, ...
c       until IFACA <= 1.
c
```

```
      5 ifcab = ifaca * 4
        z = pi / ifcab
        bcos = -two * sin(z)**2
        bsin = sin(two * z)
        cw1 = one
        sw1 = zero
        do 10 litla = 1, ifaca
          do 8 i0 = litla, n, ifcab
            i1 = i0 + ifaca
            i2 = i1 + ifaca
            i3 = i2 + ifaca
            xs0 = xreal(i0) + xreal(i2)
            xs1 = xreal(i0) - xreal(i2)
            ys0 = ximag(i0) + ximag(i2)
            ys1 = ximag(i0) - ximag(i2)
            xs2 = xreal(i1) + xreal(i3)
            xs3 = xreal(i1) - xreal(i3)
            ys2 = ximag(i1) + ximag(i3)
            ys3 = ximag(i1) - ximag(i3)
            xreal(i0) = xs0 + xs2
            ximag(i0) = ys0 + ys2
            x1 = xs1 + ys3
            y1 = ys1 - xs3
            x2 = xs0 - xs2
            y2 = ys0 - ys2
            x3 = xs1 - ys3
            y3 = ys1 + xs3
            if (litla .eq. 1) then
              xreal(i2) = x1
              ximag(i2) = y1
              xreal(i1) = x2
              ximag(i1) = y2
              xreal(i3) = x3
              ximag(i3) = y3
            else
              xreal(i2) = x1 * cw1 + y1 * sw1
              ximag(i2) = y1 * cw1 - x1 * sw1
              xreal(i1) = x2 * cw2 + y2 * sw2
              ximag(i1) = y2 * cw2 - x2 * sw2
              xreal(i3) = x3 * cw3 + y3 * sw3
              ximag(i3) = y3 * cw3 - x3 * sw3
            end if
      8   continue
c
c       Calculate a new set of twiddle factors.
c
          if (litla .lt. ifaca) then
            z = cw1 * bcos - sw1 * bsin + cw1
            sw1 = bcos * sw1 + bsin * cw1 + sw1
            tempr = one5 - half * (z * z + sw1 * sw1)
            cw1 = z * tempr
            sw1 = sw1 * tempr
            cw2 = cw1 * cw1 - sw1 * sw1
            sw2 = two * cw1 * sw1
            cw3 = cw1 * cw2 - sw1 * sw2
            sw3 = cw1 * sw2 + cw2 * sw1
          end if
     10 continue
        if (ifaca .le. 1) go to 14
c
c       Set up the transform split for the next stage.
```

```
c
      ifaca = ifaca / 4
      if (ifaca .gt. 0) go to 5
c
c       Radix 2 calculation, if needed.
c
      if (ifaca .lt. 0) return
      do 13 k = 1, n, 2
        tempr = xreal(k) + xreal(k+1)
        xreal(k+1) = xreal(k) - xreal(k+1)
        xreal(k) = tempr
        tempr = ximag(k) + ximag(k+1)
        ximag(k+1) = ximag(k) - ximag(k+1)
        ximag(k) = tempr
   13 continue
   14 if (itype .lt. 0) then
c
c       Inverse transform; conjugate the result.
c
        do 16 k = 1, n
   16   ximag(k) = -ximag(k)
        return
      end if
c
c       Forward transform
c
      z = one / n
      do 18 k = 1, n
        xreal(k) = xreal(k) * z
        ximag(k) = ximag(k) * z
   18 continue
c
      return
      end
c
c
c
      subroutine scram(xreal, ximag, n, ipow)
c
c       Algorithm AS 83.3 Appl. Statist. (1975) vol.24, no.1
c
c       Subroutine for unscrambling FFT data.
c
      real    xreal(n), ximag(n)
      integer l(19)
      equivalence (l1,l(1)), (l2,l(2)), (l3,l(3)), (l4,l(4)),
     +          (l5,l(5)), (l6,l(6)), (l7,l(7)), (l8,l(8)), (l9,l(9)),
     +          (l10,l(10)), (l11,l(11)), (l12,l(12)), (l13,l(13)),
     +          (l14,l(14)), (l15,l(15)), (l16,l(16)), (l17,l(17)),
     +          (l18,l(18)), (l19,l(19))
c
      ii = 1
      itop = 2 ** (ipow - 1)
      i = 20 - ipow
      do 5 k = 1, i
    5 l(k) = ii
      l0 = ii
      i = i + 1
      do 6 k = i, 19
        ii = ii * 2
        l(k) = ii
```

```
      6 continue
c
        ii = 0
        do 9 j1 = 1, l1, l0
          do 9 j2 = j1, l2, l3
            do 9 j3 = j2, l3, l2
              do 9 j4 = j3, l4, l3
                do 9 j5 = j4, l5, l4
                  do 9 j6 = j5, l6, l5
                    do 9 j7 = j6, l7, l6
                      do 9 j8 = j7, l8, l7
                        do 9 j9 = j8, l9, l8
                          do 9 j10 = j9, l10, l9
                            do 9 j11 = j10, l11, l10
                              do 9 j12 = j11, l12, l11
                                do 9 j13 = j12, l13, l12
                                  do 9 j14 = j13, l14, l13
                                    do 9 j15 = j14, l15, l14
                                      do 9 j16 = j15, l16, l15
                                        do 9 j17 = j16, l17, l16
                                          do 9 j18 = j17, l18, l17
                                            do 9 j19 = j18, l19, l18
                                              j20 = j19
                                              do 9 i = 1, 2
                                                ii = ii + 1
                                                if (ii .lt. j20) then
c
c       J20 is the bit-reverse of II pairwise interchange.
c
                                                  tempr = xreal(ii)
                                                  xreal(ii) = xreal(j20)
                                                  xreal(j20) = tempr
                                                  tempr = ximag(ii)
                                                  ximag(ii) = ximag(j20)
                                                  ximag(j20) = tempr
                                                end if
                                                j20 = j20 + itop
      9 continue
c
        return
        end
```

```
      subroutine skewkt(x, ip, n, b1, b2, work, ssq, nn, irank, ifault)
c
c       Algorithm AS 84  Appl. Statist. (1975) Vol.24, No. 2
c
c       For a given observation matrix X(ip, n), the subroutine
c       calculates measures of multivariate skewness and kurtosis
c       and stores them in B1 and B2
c
c       It is important that the value of irank be tested on exit as
c       the null distribution of the measures under multinormality
c       depends on the rank of the sample covariance matrix
c
c
c       Formal Parameters:-
c
c       X           Real Array (IP, N)     Input: Array containing the data,
c                                                 Each observn occupying
c                                                 one column
c                                          Output: Mean subtracted from data
c       IP          Integer                Input: Number of variables p
c       N           Integer                Input: Number of observations n
c       B1          Real                   Output: Skewness b1(1,p)
c       B2          Real                   Output: Kurtosis b(2,p)
c       WORK        Real Array (IP)        Workspace: Used by SYMINV
c       SSQ         Real Array (NN)        Output: Contains inverse A of
c                                                  corrected ssp matrix n*S
c                                                  stored in the sequence
c                                                  a11,a21,a22,a31,a32,a33,...
c       NN          Integer                Input: NN=IP*(IP+1)/2
c       IRANK       Integer                Output: Rank r of matrix S
c       IFAULT      Integer                Output: Failure Indicator:-
c                                          =0 None
c                                          =1 S is singular, g-inverse used
c                                          =2 Error returned from Syminv
c                                          =3 IP LE 1
c                                          =4 NN NE IP*(IP+1)/2
c                                          =5 N LT IP
c
c
      double precision x(ip,n), ssq(nn), work(ip), sumwts, xm, b1,
     #                 b2, qq, qr, xki, xkj, ssqm, ss, zero, two
      data zero, two /0.0d0, 2.0d0/
c
      ifault = 0
      if (ip .le. 1) ifault = 3
      if (nn .ne. ip * (ip + 1) / 2) ifault = 4
      if (n .lt. ip) ifault = 5
      if (ifault .ne. 0) return
      sumwts = n
      do 32 j = 1, ip
        xm = zero
        do 25 i = 1, n
   25   xm = xm + x(j, i)
        xm = xm / sumwts
c
c       xm at this stage contains the sample mean of the j-th variable,
c       where j is the control variable of the do 32 loop
c
        do 30 i = 1, n
   30   x(j, i) = x(j, i) - xm
   32 continue
```

```
      l = 0
      do 40 j = 1, ip
        do 40 k = 1, j
          l = l + 1
          xm = zero
          do 35 i = 1, n
   35     xm = xm + x(j, i) * x(k, i)
          ssq(l) = xm
   40 continue
c
c       ssq at this stage contains the sample matrix of corrected sums
c       of squares and products, stored as lower triangle
c
      call syminv (ssq, ip, nn, ssq, work, irank, ifault)
      if (ifault .ne. 0) return
      if (irank .ne. 0) ifault = 1
      irank = ip - irank
      b1 = zero
      b2 = zero
      do 100 i = 1, n
        xm = ssq(1) * x(1, i) ** 2
        m = 1
        do 70 k = 2, ip
          qq = zero
          kk = k - 1
          do 50 l = 1, kk
            m = m + 1
            qq = qq + ssq(m) * x(l, i)
   50     continue
          m = m + 1
          xki = x(k, i)
          xm = xm + (ssq(m) * xki + two * qq) * xki
   70   continue
        qq = xm ** 2
        b2 = b2 + qq
        b1 = b1 + qq * xm
  100 continue
      do 150 i = 2, n
        ii = i - 1
        ss = ssq(1) * x(1, i)
        do 150 j = 1, ii
          xm = ss * x(1, j)
          m = 1
          do 120 k = 2, ip
            xki = x(k, i)
            xkj = x(k, j)
            qq = zero
            qr = zero
            kk = k - 1
            do 110 l = 1, kk
              m = m + 1
              ssqm = ssq(m)
              qq = qq + ssqm * x(l, j)
              qr = qr + ssqm * x(l, i)
  110       continue
            qq = qq * xki + qr * xkj
            m = m + 1
            xm = xm + qq + ssq(m) * xki * xkj
  120     continue
          b1 = b1 + two * xm ** 3
  150 continue
```

```
        b1 = b1 * sumwts
        b2 = b2 * sumwts
        return
        end
```

```
      subroutine allnr(n, r, j, ifault)
c
c        Algorithm AS 88  Appl. Statist. (1975) Vol.24, No. 3
c
c        When called once, generates all possible combinations
c        from a group of N items.  Each combination (represented in j as
c        r ordered integers between 1 and n) is processed within allnr.
c
c        Parameters:-
c
c        n        integer             input:  The size of the group from which
c                                             the combinations are selected.
c
c        r        integer             input:  The size of each comination.
c
c        j        integer array(r)  workspace: Used by allnr to store
c                                              combinations.
c
c        ifault   integer            output:  Fault indicator, equal to:
c                                             0 if 1 le R le N;
c                                             1 otherwise.
c
      integer r, j(r)
c
      ifault = 1
      if (r .lt.1 .or. r .gt. n) return
      ifault = 0
      kount = 0
      nmr = n - r
c
c        Initialize J(1) to lower limit separately, since lower limit for
c        each index depends on lower limit for previous index
c
      i = 1
      j(1) = 1
c
c        Initialize indices for loops i=1,...,r to lower limits
c
    1 if (i .eq. r) goto 3
      ip1 = i + 1
      do 2 l = ip1, r
    2 j(l) = j(l - 1) + 1
c
c        Update the count (kount) of combinations and process the current
c        combination.  The call to Subroutine job may be replaced by
c        statements to process the current combination.
c
    3 kount = kount + 1
      call job(n, r, j, kount)
c
c        Increment the first possible index (of loop i) among indices of
c        loops R, R-1,...,1
c
      i = r
    4 if (j(i) .lt. nmr + i) goto 5
      i = i - 1
c
c        Return after all indices have achieved their upper limits
c
      if (i .le. 0) return
      goto 4
```

```
      5 j(i) = j(i) + 1
        goto 1
        end
c
c
c
        subroutine job (n, r, j, kount)
        integer r, j(r)
        return
        end
```

```
      double precision function prho(n, is, ifault)
c
c        Algorithm AS 89   Appl. Statist. (1975) Vol.24, No. 3, P377.
c
c        To evaluate the probability of obtaining a value greater than or
c        equal to is, where is=(n**3-n)*(1-r)/6, r=Spearman's rho and n
c        must be greater than 1
c
c     Auxiliary function required: ALNORM = algorithm AS66
c
      dimension l(6)
      double precision zero, one, two, b, x, y, z, u, six,
     $  c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12
      data zero, one, two, six /0.0d0, 1.0d0, 2.0d0, 6.0d0/
      data        c1,     c2,     c3,     c4,     c5,     c6,
     $          c7,     c8,     c9,    c10,    c11,    c12/
     $  0.2274d0, 0.2531d0, 0.1745d0, 0.0758d0, 0.1033d0, 0.3932d0,
     $  0.0879d0, 0.0151d0, 0.0072d0, 0.0831d0, 0.0131d0, 0.00046d0/
c
c        Test admissibility of arguments and initialize
c
      prho = one
      ifault = 1
      if (n .le. 1) return
      ifault = 0
      if (is .le. 0) return
      prho = zero
      if (is .gt. n * (n * n -1) / 3) return
      js = is
      if (js .ne. 2 * (js / 2)) js = js + 1
      if (n .gt. 6) goto 6
c
c        Exact evaluation of probability
c
      nfac = 1
      do 1 i = 1, n
        nfac = nfac * i
        l(i) = i
    1 continue
      prho = one / dble(nfac)
      if (js .eq. n * (n * n -1) / 3) return
      ifr = 0
      do 5 m = 1,nfac
        ise = 0
        do 2 i = 1, n
          ise = ise + (i - l(i)) ** 2
    2   continue
        if (js .le. ise) ifr = ifr + 1
        n1 = n
    3   mt = l(1)
        nn = n1 - 1
        do 4 i = 1, nn
          l(i) = l(i + 1)
    4   continue
        l(n1) = mt
        if (l(n1) .ne. n1 .or. n1 .eq. 2) goto 5
        n1 = n1 - 1
        if (m .ne. nfac) goto 3
    5 continue
      prho = dble(ifr) / dble(nfac)
      return
```

```
c
c        Evaluation by Edgeworth series expansion
c
    6 b = one / dble(n)
      x = (six * (dble(js) - one) * b / (one / (b * b) -one) -
     $  one) * sqrt(one / b - one)
      y = x * x
      u = x * b * (c1 + b * (c2 + c3 * b) + y * (-c4
     $  + b * (c5 + c6 * b) - y * b * (c7 + c8 * b
     $  - y * (c9 - c10 * b + y * b * (c11 - c12 * y)))))
c
c      Call to algorithm AS 66
c
      prho = u / exp(y / two) + alnorm(x, .true.)
      if (prho .lt. zero) prho = zero
      if (prho .gt. one) prho = one
      return
      end
```

```
      double precision function ppchi2(p, v, g, ifault)
c
c       Algorithm AS 91   Appl. Statist. (1975) Vol.24, P.35
c
c       To evaluate the percentage points of the chi-squared
c       probability distribution function.
c
c       p must lie in the range 0.000002 to 0.999998,
c       v must be positive,
c       g must be supplied and should be equal to
c         ln(gamma(v/2.0))
c
c     Incorporates the suggested changes in AS R85 (vol.40(1),
c     pp.233-5, 1991) which should eliminate the need for the limited
c     range for p above, though these limits have not been removed
c     from the routine.
c     If IFAULT = 4 is returned, the result is probably as accurate as
c     the machine will allow.
c
c     Auxiliary routines required: PPND = AS 111 (or AS 241) and
c     GAMMAD = AS 239.
c
      integer maxit
      parameter (maxit = 20)
      double precision p, v, g, gammad, ppnd, aa, e, zero, half, one,
     $   two, three, six, pmin, pmax, c1, c2, c3, c4, c5, c6, c7,
     $   c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19,
     $   c20, c21, c22, c23, c24, c25, c26, c27, c28, c29, c30,
     $   c31, c32, c33, c34, c35, c36, c37, c38, a, b, c, ch, p1, p2,
     $   q, s1, s2, s3, s4, s5, s6, t, x, xx
c
      data             aa,       e,    pmin,     pmax
     $     /0.6931471806d0, 0.5d-6, 0.000002d0, 0.999998d0/
      data zero, half, one, two, three, six
     $     /0.0d0, 0.5d0, 1.0d0, 2.0d0, 3.0d0, 6.0d0/
      data        c1,     c2,     c3,     c4,     c5,     c6,
     $            c7,     c8,     c9,    c10,    c11,    c12,
     $           c13,    c14,    c15,    c16,    c17,    c18,
     $           c19,    c20,    c21,    c22,    c23,    c24,
     $           c25,    c26,    c27,    c28,    c29,    c30,
     $           c31,    c32,    c33,    c34,    c35,    c36,
     $           c37,    c38/
     $         0.01d0, 0.222222d0, 0.32d0, 0.4d0, 1.24d0, 2.2d0,
     $         4.67d0, 6.66d0, 6.73d0, 13.32d0, 60.0d0, 70.0d0,
     $         84.0d0, 105.0d0, 120.0d0, 127.0d0, 140.0d0, 175.0d0,
     $        210.0d0, 252.0d0, 264.0d0, 294.0d0, 346.0d0, 420.0d0,
     $        462.0d0, 606.0d0, 672.0d0, 707.0d0, 735.0d0, 889.0d0,
     $        932.0d0, 966.0d0, 1141.0d0, 1182.0d0, 1278.0d0, 1740.0d0,
     $       2520.0d0, 5040.0d0/
c
c       test arguments and initialise
c
      ppchi2 = -one
      ifault = 1
      if (p .lt. pmin .or. p .gt. pmax) return
      ifault = 2
      if (v .le. zero) return
      ifault = 0
      xx = half * v
      c = xx - one
c
```

```
c        starting approximation for small chi-squared
c
      if (v .ge. -c5 * log(p)) goto 1
      ch = (p * xx * exp(g + xx * aa)) ** (one/xx)
      if (ch .lt. e) goto 6
      goto 4
c
c        starting approximation for v less than or equal to 0.32
c
    1 if (v .gt. c3) goto 3
      ch = c4
      a = log(one-p)
    2 q = ch
      p1 = one + ch * (c7+ch)
      p2 = ch * (c9 + ch * (c8 + ch))
      t = -half + (c7 + two * ch) / p1 - (c9 + ch * (c10 +
     $  three * ch)) / p2
      ch = ch - (one - exp(a + g + half * ch + c * aa) *
     $  p2 / p1) / t
      if (abs(q / ch - one) .gt. c1) goto 2
      goto 4
c
c        call to algorithm AS 111 - note that p has been tested above.
c        AS 241 could be used as an alternative.
c
    3 x = ppnd(p, if1)
c
c        starting approximation using Wilson and Hilferty estimate
c
      p1 = c2 / v
      ch = v * (x * sqrt(p1) + one - p1) ** 3
c
c        starting approximation for p tending to 1
c
      if (ch .gt. c6 * v + six)
     $   ch = -two * (log(one-p) - c * log(half * ch) + g)
c
c
c        call to algorithm AS 239 and calculation of seven term
c        Taylor series
c
    4 do 7 i = 1, maxit
      q = ch
      p1 = half * ch
      p2 = p - gammad(p1, xx, if1)
      if (if1 .eq. 0) goto 5
c
      ifault = 3
      return
    5 t = p2 * exp(xx * aa + g + p1 - c * log(ch))
      b = t / ch
      a = half * t - b * c
      s1 = (c19 + a * (c17 + a * (c14 + a * (c13 + a * (c12 +
     $  c11 * a))))) / c24
      s2 = (c24 + a * (c29 + a * (c32 + a * (c33 + c35 *
     $  a)))) / c37
      s3 = (c19 + a * (c25 + a * (c28 + c31 * a))) / c37
      s4 = (c20 + a * (c27 + c34 * a) + c * (c22 + a * (c30 +
     $  c36 * a))) / c38
      s5 = (c13 + c21 * a + c * (c18 + c26 * a)) / c37
      s6 = (c15 + c * (c23 + c16 * c)) / c38
```

```
        ch = ch + t * (one + half * t * s1 - b * c * (s1 - b *
     $  (s2 - b * (s3 - b * (s4 - b * (s5 - b * s6))))))
        if (abs(q / ch - one) .gt. e) goto 6
    7 continue
        ifault = 4
c
    6 ppchi2 = ch
        return
        end
```

```
c Routine AS 93 returns frequencies.   The following short routine calculates
c the distribution function from these frequencies (overwriting them).
c The calling arguments are as for AS 93.   The distribution function is
c returned in array A1.   The first element in A1 is F(ASTART).   N.B. ASTART
c is a real variable.
c
      subroutine wprob(test, other, astart, a1, l1, a2, a3, ifault)
      integer test, other, l1, ifault
      real astart, a1(l1), a2(l1), a3(l1)
c
c     Local variables
c
      real zero, sum
      data zero /0.0/
c
      call gscale(test, other, astart, a1, l1, a2, a3, ifault)
      if (ifault .ne. 0) return
c
c     Scale column of F
c
      nrows = 1 + (test * other)/2
      sum = zero
      do 10 i = 1, nrows
        sum = sum + a1(i)
        a1(i) = sum
   10 continue
      do 20 i = 1, nrows
   20 a1(i) = a1(i) / sum
c
      return
      end

c----------------------------------------------------------------------

        SUBROUTINE GSCALE(TEST, OTHER, ASTART, A1, L1, A2, A3, IFAULT)
C
C         ALGORITHM AS 93 APPL. STATIST. (1976) VOL.25, NO.1
C
C         FROM THE SIZES OF TWO SAMPLES THE DISTRIBUTION OF THE
C         ANSARI-BRADLEY TEST FOR SCALE IS GENERATED IN ARRAY A1.
C
        REAL ASTART, A1(L1), A2(L1), A3(L1), AI, ONE, FPOINT
        INTEGER TEST, OTHER
        LOGICAL SYMM
        DATA ONE /1.0/
C
C         TYPE CONVERSION (EFFECT DEPENDS ON TYPE STATEMENT ABOVE).
C
        FPOINT(I) = I
C
C         CHECK PROBLEM SIZE AND DEFINE BASE VALUE OF THE DISTRIBUTION.
C
        M = MIN0(TEST, OTHER)
        IFAULT = 2
        IF (M. LT. 0) RETURN
        ASTART = FPOINT((TEST + 1) / 2) * FPOINT(1 + TEST / 2)
        N = MAX0(TEST, OTHER)
C
C         CHECK SIZE OF RESULT ARRAY.
C
        IFAULT = 1
```

```
      LRES = 1 + (M * N) / 2
      IF (L1 .LT. LRES) RETURN
      SYMM = MOD(M + N, 2) .EQ. 0
C
C       TREAT SMALL SAMPLES SEPARATELY.
C
      MM1 = M - 1
      IF (M .GT. 2) GOTO 5
C
C       START-UP PROCEDURES ONLY NEEDED.
C
      IF (MM1) 1, 2, 3
C
C       ONE SAMPLE ONLY.
C
1     A1(1) = ONE
      GOTO 15
C
C       SMALLER SAMPLE SIZE = 1.
C
2      CALL START1(N, A1, L1, LN1)
      GOTO 4
C
C       SMALLER SAMPLE SIZE = 2.
C
3     CALL START2(N, A1, L1, LN1)
C
C     RETURN IF A1 IS NOT IN REVERSE ORDER.
C
4     IF (SYMM .OR. (OTHER .GT. TEST)) GOTO 15
      GOTO 13
C
C       FULL GENERATOR NEEDED
C       SET UP INITIAL CONDITIONS (DEPENDS ON MOD(N, 2)).
C
5     NM1 = N - 1
      NM2 = N - 2
      MNOW = 3
      NC = 3
      IF (MOD(N, 2) .EQ. 1) GOTO 6
C       SET UP FOR EVEN N.
C
      N2B1 = 3
      N2B2 = 2
      CALL START2(N, A1, L1, LN1)
      CALL START2(NM2, A3, L1, LN3)
      CALL START1(NM1, A2, L1, LN2)
      GOTO 8
C
C       SET UP FOR ODD N.
C
6     N2B1 = 2
      N2B2 = 3
      CALL START1(N, A1, L1, LN1)
      CALL START2(NM1, A2, L1, LN2)
C
C       INCREASE ORDER OF DISTRIBUTION IN A1 BY 2
C       (USING A2 AND IMPLYING A3).
C
7     CALL FRQADD(A1, LN1, L1OUT, L1, A2, LN2, N2B1)
      LN1 = LN1 + N
```

```
          CALL IMPLY(A1, L1OUT, LN1, A3, LN3, L1, NC)
          NC = NC + 1
          IF (MNOW .EQ. M) GOTO 9
          MNOW = MNOW + 1
C
C           INCREASE ORDER OF DISTRIBUTION IN A2 BY 2 (USING A3).
C
8         CALL FRQADD(A2, LN2, L2OUT, L1, A3, LN3, N2B2)
          LN2 = LN2 + NM1
          CALL IMPLY(A2, L2OUT, LN2, A3, J, L1, NC)
          NC = NC + 1
          IF (MNOW .EQ. M) GOTO 9
          MNOW = MNOW + 1
          GOTO 7
C
C           IF SYMMETRICAL, RESULTS IN A1 ARE COMPLETE.
C
9         IF (SYMM) GOTO 15
C
C           FOR A SKEW RESULT ADD A2 (OFFSET) INTO A1.
C
          KS = (M + 3) / 2
          J = 1
          DO 12 I = KS, LRES
          IF (I .GT. LN1) GOTO 10
          A1(I) = A1(I) + A2(J)
          GOTO 11
10        A1(I) = A2(J)
11        J = J + 1
12        CONTINUE
C
C           DISTRIBUTION IN A1 POSSIBLY IN REVERSE ORDER.
C
          IF (OTHER .LT. TEST) GOTO 15
C
C           REVERSE THE RESULTS IN A1.
C
13        J = LRES
          NDO = LRES / 2
          DO 14 I = 1, NDO
          AI = A1(I)
          A1(I) =A1(J)
          A1(J) = AI
          J = J - 1
14        CONTINUE
C
C           FINAL RESULTS NOW IN A1.
C
15        IFAULT = 0
          RETURN
          END

          SUBROUTINE START1(N, F, L, LOUT)
C
C           ALGORITHM AS 93.1 APPL. STATIST. (1976) VOL.25, NO.1
C
C           GENERATES A 1,N ANSARI-BRADLEY DISTRIBUTION IN F.
C
          REAL F(L), ONE, TWO
          DATA ONE, TWO /1.0, 2.0/
          LOUT = 1 + N / 2
```

```
      DO 1 I = 1, LOUT
1     F(I) = TWO
      IF (MOD(N, 2) .EQ. 0) F(LOUT) = ONE
      RETURN
      END
C
      SUBROUTINE START2(N, F, L, LOUT)
C
C       ALGORITHM AS 93.2 APPL. STATIST. (1976) VOL.25, NO.1
C
C       GENERATES A 2,N ANSARI-BRADLEY DISTRIBUTION IN F.
C
      REAL F(L), ONE, TWO, THREE, FOUR
      DATA ONE, TWO, THREE, FOUR /1.0, 2.0, 3.0, 4.0/
C
C       DERIVE F FOR 2, NU, WHERE NU IS HIGHEST EVEN INTEGER
C       LESS THAN OR EQUAL TO N.
C       DEFINE NU AND ARRAY LIMITS.
C
      NU = N - MOD(N, 2)
      J = NU + 1
      LOUT = J
      LT1 = LOUT + 1
      NDO = LT1 / 2
      A = ONE
      B = THREE
C
C       GENERATE THE SYMMETRICAL 2,NU DISTRIBUTION.
C
      DO 1 I = 1, NDO
      F(I) = A
      F(J) = A
      J = J - 1
      A = A + B
      B = FOUR - B
1     CONTINUE
      IF (NU .EQ. N) RETURN
C
C       ADD AN OFFSET 1,N DISTRIBUTION INTO F TO GIVE 2,N RESULT.
C
      NU = NDO + 1
      DO 2 I = NU, LOUT
2     F(I) = F(I) + TWO
      F(LT1) = TWO
      LOUT = LT1
      RETURN
      END
C
      SUBROUTINE FRQADD(F1, L1IN, L1OUT, L1, F2, L2, NSTART)
C
C       ALGORITHM AS 93.3 APPL. STATIST. (1976) VOL.25, NO.1
C
C       ARRAY F1 HAS TWICE THE CONTENTS OF ARRAY F2 ADDED INTO IT
C       STARTING WITH ELEMENTS NSTART AND 1 IN F1 AND F2 RESPECTIVELY.
C
      REAL F1(L1), F2(L2), MUL2
      DATA MUL2 /2.0/
      I2 = 1
      DO 1 I1 = NSTART, L1IN
      F1(I1) = F1(I1) + MUL2 * F2(I2)
      I2 = I2 + 1
```

```
1       CONTINUE
        NXT = L1IN + 1
        L1OUT = L2 + NSTART - 1
        DO 2 I1 = NXT, L1OUT
        F1(I1) = MUL2 * F2(I2)
        I2 = I2 + 1
2       CONTINUE
        NSTART = NSTART + 1
        RETURN
        END
C
        SUBROUTINE IMPLY(F1, L1IN, L1OUT, F2, L2, L2MAX, NOFF)
C
C         ALGORITHM AS 93.4 APPL. STATIST. (1976) VOL.25, NO.1
C
C         GIVEN L1IN ELEMENTS OF AN ARRAY F1, A SYMMETRICAL
C         ARRAY F2 IS DERIVED AND ADDED ONTO F1, LEAVING THE
C         FIRST NOFF ELEMENTS OF F1 UNCHANGED AND GIVING A
C         SYMMETRICAL RESULT OF L1OUT ELEMENTS IN F1.
C
        REAL F1(L1OUT), F2(L2MAX), SUM, DIFF
C
C         SET-UP SUBSCRIPTS AND LOOP COUNTER.
C
        I2 = 1 - NOFF
        J1 = L1OUT
        J2 = L1OUT - NOFF
        L2 = J2
        J2MIN = (J2 + 1) / 2
        NDO = (L1OUT + 1) / 2
C
C         DERIVE AND IMPLY NEW VALUES FROM OUTSIDE INWARDS.
C
        DO 6 I1 = 1, NDO
C
C         GET NEW F1 VALUE FROM SUM OF L/H ELEMENTS OF
C         F1 + F2 (IF F2 IS IN RANGE).
C
        IF (I2 .GT. 0) GOTO 1
        SUM = F1(I1)
        GOTO 2
1       SUM = F1(I1) + F2(I2)
C
C         REVISE LEFT ELEMENT OF F1.
C
        F1(I1) = SUM
C
C         IF F2 NOT COMPLETE IMPLY AND ASSIGN F2 VALUES
C         AND REVISE SUBSCRIPTS.
C
2       I2 = I2 + 1
        IF (J2 .LT. J2MIN) GOTO 5
        IF (J1 .LE. L1IN) GOTO 3
        DIFF = SUM
        GOTO 4
3       DIFF = SUM - F1(J1)
4       F2(I1) = DIFF
        F2(J2) = DIFF
        J2 = J2 - 1
C
C         ASSIGN R/H ELEMENT OF F1 AND REVISE SUBSCRIPT.
```

```
C
5       F1(J1) = SUM
        J1 = J1 - 1
6       CONTINUE
        RETURN
        END
```

```
      SUBROUTINE CURVE (P, X, N0, N, EPS, MAXITR, MU, SIGMA, ITER,
     1   SEMU, SESIG, COVAR, E0, EX, CHISQ, IFAULT)
C
C       ALGORITHM AS 95 APPL. STATIST. (1976) VOL.25, NO.1
C
C       ESTIMATES MU AND SIGMA OF DISTRIBUTION FUNCTION
C       F( (X-MU)/SIGMA ) FROM A GROUPED SAMPLE OF X VALUES.
C       NOTE ON ARRAY SIZES
C       THE ARRAYS IN THE SECOND DIMENSION STATEMENT MUST HAVE
C       MINIMUM SIZE P.  IF P IS TO EXCEED 20, A SUITABLE SIZE
C       MUST BE SET FOR THEM, AND THE IF STATEMENT WHICH CHECKS
C       THE VALUE OF P MUST BE AMENDED.
C
C     Auxiliary routines required: FUNC & DEVIAT (both user-supplied)
C
      INTEGER P
      REAL NN, NI, NP, MU, ONE, ZERO
      DIMENSION X(P), N(P), EX(P)
      DIMENSION F(20), F1(20), XN(20)
      DATA RR/1.0E-10/, ONE/1.0/, ZERO/0.0/
C
C       ERROR EXIT IF P TOO SMALL OR TOO LARGE
C
      IF (P.LT.2 .OR. P.GT.20) GO TO 80
      IFAULT = 0
C
C       SET FREQUENCIES IN FLOATING POINT
C
      XN0 = N0
      NSUM = N0
      DO 10 I = 1, P
        XN(I) = N(I)
        NSUM = NSUM + N(I)
 10   CONTINUE
      K = P - 1
      XNSUM = FLOAT(NSUM)
      NP = XN(P)
C
C       ITERATIVE APPROXIMATION
C
      DO 40 ITER = 1, MAXITR
C
C       COMPUTE VALUES OF DISTRIBUTION AND DENSITY FUNCTIONS,
C       USING CURRENT VALUES OF MU, SIGMA
C
        DO 20 I = 1, P
 20       CALL FUNC ((X(I) - MU)/SIGMA, F(I), F1(I))
        DM = ONE - F(P)
C
C
C       TEST FOR SMALL DIVISOR TO AVOID OVERFLOW
C
        IF (ABS(DM).LT.RR) GO TO 90
        F1P = F1(P)
        IF (ABS(F(1)).LT.RR) GO TO 90
        XI1 = X(1) - MU
        XP = X(P) - MU
        R = F1(1)/F(1)
        S = F1P/DM
        T = -XN0*R
        U = NP*S
```

```
          A = T + U
          B = XI1*T + XP*U
          R = F1(1)*R
          S = F1P*S
          C = R + S
          R = XI1*S
          S = XP*S
          D = R + S
          E = XI1*R + XP*S
          DO 30 I = 1, K
            FI = F(I)
            FI1 = F(I + 1)
            F1I1 = F1(I + 1)
            F1I = F1(I)
            XI = XI1
            XI1 = X(I + 1) - MU
            NI = XN(I)
            R = FI1 - FI
            IF (ABS(R).LT.RR) GO TO 90
            S = F1I1 - F1I
            U = XI1*F1I1 - XI*F1I
            SR = S/R
            UR = U/R
            A = A - NI*SR
            B = B - NI*UR
            C = C + S*SR
            D = D + S*UR
            E = E + U*UR
 30       CONTINUE
          DENOM = (C*E - D*D)*XNSUM
C
C         COMPUTE ADJUSTMENTS TO MU, SIGMA
C
          SIGDEN = SIGMA/DENOM
          DMU = (E*A - D*B)*SIGDEN
          DSIGMA = (C*B - D*A)*SIGMA*SIGDEN
          MU = MU + DMU
          SIGMA = SIGMA + DSIGMA
          ERR = ABS(DMU) + ABS(DSIGMA)
C
C         TEST FOR CONVERGENCE
C
          IF (ERR.LT.EPS) GO TO 50
 40     CONTINUE
C
C         SET FAULT IF LIMIT FOR NUMBER OF ITERATIONS IS
C         REACHED, THEN PROCEED
C
        IFAULT = 4
        ITER = MAXITR
 50     DO 60 I = 1, P
 60       CALL FUNC ((X(I) - MU)/SIGMA, F(I), DUM)
C
C         COMPUTE VARIANCES AND COVARIANCE OF ESTIMATES
C
        SIGDEN = SIGMA*SIGMA/DENOM
        VARMU = E*SIGDEN
        SIGDEN = SIGMA*SIGDEN
        COVAR = -D*SIGDEN
        VARSIG = C*SIGMA*SIGDEN
        IF (VARMU.LT.ZERO .OR. VARSIG.LT.ZERO) GO TO 100
```

```
      SEMU = SQRT(VARMU)
      SESIG = SQRT(VARSIG)
C
C       COMPUTE EXPECTED FREQUENCIES AND CHI SQUARE
C
      E0 = XNSUM*F(1)
      EP = XNSUM*(ONE - F(P))
      EX(P) = EP
      CHISQ = ((XN0 - E0)**2)/E0 + ((NP - EP)**2)/EP
      DO 70 I = 1, K
        NN = XNSUM*(F(I+1) - F(I))
        CHISQ = CHISQ + ((NN - XN(I))**2)/NN
        EX(I) = NN
 70   CONTINUE
      RETURN
C
C       ERROR EXITS
C
 80   IFAULT = 1
      RETURN
 90   IFAULT = 2
      RETURN
 100  IFAULT = 3
      RETURN
      END
C
C
      SUBROUTINE INITL (P, X, N0, N, MU, SIGMA, IFAULT)
C
C       ALGORITHM AS 95.1 APPL. STATIST. (1976) VOL.25, NO.1
C
C       COMPUTES ROUGH LEAST SQUARES ESTIMATES OF MU AND SIGMA
C       ( FOR DEFINITION OF MU AND SIGMA SEE SUBROUTINE CURVE ).
C
      INTEGER P
      REAL MU, ONE, ZERO
      DIMENSION X(P), N(P)
      DATA ONE/1.0/, ZERO/0.0/
C
C       ERROR EXIT IF P TOO SMALL
C
      IFAULT = 1
      IF (P.LT.2) RETURN
      IFAULT = 0
C
C       COMPUTE AND FLOAT SUM OF FREQUENCIES
C
      NSUM = N0
      DO 10 I = 1, P
 10     NSUM = NSUM + N(I)
      XNSUM = FLOAT(NSUM)
C
C       ZERO ACCUMULATORS
C
      NPAR = N0
      XBAR = ZERO
      YBAR = ZERO
      SXX = ZERO
      SXY = ZERO
      SW = ZERO
C
```

```
C       COMPUTE WEIGHTED MEANS XAR, YBAR, AND CORRECTED SUMS
C       OF X*X AND X*Y.
C
      DO 30 I = 1, P
C
C       NULL FREQUENCIES AT EITHER END OF THE RANGE ARE
C       ZERO WEIGHTED
C
        IF (NPAR.EQ.0 .OR. NPAR.EQ.NSUM) GO TO 20
        PROB = FLOAT(NPAR)/XNSUM
        Y = DEVIAT(PROB)
        CALL FUNC (Y, DUMMY, DFY)
        DX = X(I) - XBAR
        DY = Y - YBAR
        W = DFY*DFY/(PROB*(ONE - PROB))
        SW = SW + W
        FAC = W/SW
        XBAR = XBAR + FAC*DX
        YBAR = YBAR + FAC*DY
        FAC = W*DX*(ONE - FAC)
        SXX = SXX + FAC*DX
        SXY = SXY + FAC*DY
 20     NPAR = NPAR + N(I)
 30   CONTINUE
      SIGMA = SXX/SXY
      MU = XBAR - SIGMA*YBAR
      RETURN
      END
```

```
      SUBROUTINE SCALE(FMN, FMX, N, VALMIN, STEP, VALMAX, IFAULT)
C
C     ALGORITHM AS 96  APPL. STATIST. (1976) VOL.25, NO.1
C
C     Given extreme values FMN, FMX, and the need for a scale with N
C     marks, calculates value for the lowest scale mark (VALMIN) and
C     step length (STEP) and highest scale mark (VALMAX).
C
      REAL FMN, FMX, VALMIN, STEP, VALMAX
      INTEGER N, IFAULT
C
C     Units for step lengths
C
      REAL UNIT(11)
C
C     Local variables
C
      INTEGER NUNIT, I, J
      REAL TOL, ZERO, HALF, ONE, TEN, BIAS, FMAX, FMIN, RN, X, S, RANGE
C
C     Array length unit()
C
      DATA NUNIT /11/
C
C     Local constant, defining effective equality of values.
C
      DATA TOL /5.0E-6/
      DATA ZERO /0.0/, HALF /0.5/, ONE /1.0/, TEN /10.0/
      DATA BIAS /1.0E-4/
      DATA UNIT /1.0, 1.2, 1.6, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0, 8.0,
     *        10.0/
C
      FMAX = FMX
      FMIN = FMN
      IFAULT = 1
C
C     Test for valid parameter values
C
      IF (FMAX .LT. FMIN .OR. N .LE. 1) RETURN
      IFAULT = 0
      RN = N - 1
      X = ABS(FMAX)
      IF (X .EQ. ZERO) X = ONE
      IF ((FMAX - FMIN) / X .GT. TOL) GO TO 20
C
C     All values effectively equal
C
      IF (FMAX .LT. ZERO) THEN
        FMAX = ZERO
      ELSE IF (FMAX .EQ. ZERO) THEN
        FMAX = ONE
      ELSE
        FMIN = ZERO
      END IF
C
   20 STEP = (FMAX - FMIN) / RN
      S = STEP
C
C     Find power of 10
C
   25 IF (S .GE. ONE) GO TO 30
```

```
         S = S * TEN
         GO TO 25
      30 IF (S .LT. TEN) GO TO 35
         S = S / TEN
         GO TO 30
C
C     Calculate STEP
C
      35 X = S - BIAS
         DO 40 I = 1, NUNIT
           IF (X .LE. UNIT(I)) GO TO 45
      40 CONTINUE
      45 STEP = STEP * UNIT(I) / S
         RANGE = STEP * RN
C
C     Make first estimate of VALMIN
C
         X = HALF * (ONE + (FMIN + FMAX - RANGE) / STEP)
         J = X - BIAS
         IF (X .LT. ZERO) J = J - 1
         VALMIN = STEP * FLOAT(J)
C
C     Test if VALMIN could be zero
C
         IF (FMIN .GE. ZERO .AND. RANGE .GE. FMAX) VALMIN = ZERO
         VALMAX = VALMIN + RANGE
C
C     Test if VALMAX could be zero
C
         IF (FMAX .GT. ZERO .OR. RANGE .LT. -FMIN) RETURN
         VALMAX = ZERO
         VALMIN = -RANGE
         RETURN
         END
```

```
      SUBROUTINE FORRT(X, M)
C
C     ALGORITHM AS 97  APPL. STATIST. (1976) VOL.25, NO. 2
C
C     Forward discrete Fourier transform in one dimension of real
C     data using complex transform subroutine FASTG.
C
C     X = array of real input data, type real, dimension M.
C     M = length of the transform, must be a power of 2.
C     The minimum length is 8, maximum 2**21.
C
C     The result is placed in X as described in the text of the paper.
C
C     Auxiliary routines required: SCRAG (or SCRAM) & FASTG from AS 83,
C     but with SCRAG modified as described on page 168 of the paper for
C     this algorithm.
C
      REAL X(M)
      DATA ZERO/0.0/, QUART/0.25/, HALF/0.5/, ONE/1.0/, ONE5/1.5/,
     *  TWO/2.0/, FOUR/4.0/
C
C     Check for valid transform size.
C
      II = 8
      DO 2 K = 3, 21
        IPOW = K
        IF (II .EQ. M) GO TO 3
        II = II * 2
    2 CONTINUE
C
C     If this point is reached, an illegal size was specified.
C
      RETURN
    3 PIE = FOUR * ATAN(ONE)
C
C     Separate odd and even parts into two halves.
C     First bit reverse the whole array of length M.
C
      CALL SCRAG(X, M, IPOW)
C
C     Next bit reverse the half arrays separately.
C
      N = M / 2
      JPOW = IPOW - 1
      CALL SCRAG(X, N, JPOW)
      CALL SCRAG(X(N+1), N, JPOW)
C
C     Faster alternative to the two lines above to SCRAM.
C       CALL SCRAM(X, X(N+1), N, JPOW)
C
C     Now do the transform.
C
      CALL FASTG(X, X(N+1), N, 1)
C
C     Unscramble the transform results.
C
      CALL SCRAG(X, N, JPOW)
      CALL SCRAG(X(N+1), N, JPOW)
C
C     Faster alternative to the two lines above to SCRAM.
C       CALL SCRAM(X, X(N+1), N, JPOW)
```

```
C
      NN = N / 2
C
C     Now unravel the result; first the special cases.
C
      Z = HALF * (X(1) + X(N+1))
      X(N+1) = HALF * (X(1) - X(N+1))
      X(1) = Z
      NN1 = NN + 1
      NN2 = NN1 + N
      X(NN1) = HALF * X(NN1)
      X(NN2) = -HALF * X(NN2)
      Z = PIE / N
      BCOS = -TWO * (SIN(Z / TWO) **2)
      BSIN = SIN(Z)
      UN = ONE
      VN = ZERO
      DO 4 K = 2, NN
        Z = UN * BCOS + VN * BSIN + UN
        VN = VN * BCOS - UN * BSIN + VN
        SAVE1 = ONE5 - HALF * (Z * Z + VN * VN)
        UN = Z * SAVE1
        VN = VN * SAVE1
        KI = N + K
        L = N + 2 - K
        LI = N + L
        AN = QUART * (X(K) + X(L))
        BN = QUART * (X(KI) - X(LI))
        CN = QUART * (X(KI) + X(LI))
        DN = QUART * (X(L) - X(K))
        XN = UN * CN - VN * DN
        YN = UN * DN + VN * CN
        X(K) = AN + XN
        X(KI) = BN + YN
        X(L) = AN - XN
        X(LI) = YN - BN
    4 CONTINUE
      RETURN
      END
C

      SUBROUTINE REVRT(X, M)
C
C     ALGORITHM AS 97.1  APPL. STATIST. (1976) VOL.25, NO. 2
C
C     Inverse discrete Fourier transform in one dimension of real
C     data using complex transform subroutine FASTG.
C
C     X = array of Fourier components as output from subroutine FORRT,
C         type real, dimension M.
C     M = length of the inverse transform, must be a power of 2.
C     The minimum length is 8, maximum 2**21.
C
C     Auxiliary routines required: SCRAG & FASTG from AS 83, but
C     with SCRAG modified as described on page 168 of the paper for
C     this algorithm.
C
      REAL X(M)
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/, ONE5/1.5/,
     *  TWO/2.0/, FOUR/4.0/
C
```

```
C     Check for valid transform size.
C
      II = 8
      DO 2 K = 3, 21
        IPOW = K
        IF (II .EQ. M) GO TO 3
        II = II * 2
    2 CONTINUE
C
C     If this point is reached, an illegal size was specified.
C
      RETURN
    3 PIE = FOUR * ATAN(ONE)
      N = M / 2
      NN = N / 2
C
C     Undo the spectrum into that of two interleaved series.
C     First, the special cases.
C
      Z = X(1) + X(N+1)
      X(N+1) = X(1) - X(N+1)
      X(1) = Z
      NN1 = NN + 1
      NN2 = NN1 + N
      X(NN1) = TWO * X(NN1)
      X(NN2) = -TWO * X(NN2)
      Z = PIE / N
      BCOS = -TWO * (SIN(Z / TWO) **2)
      BSIN = SIN(Z)
      UN = ONE
      VN = ZERO
      DO 4 K = 2, NN
        Z = UN * BCOS + VN * BSIN + UN
        VN = VN * BCOS - UN * BSIN + VN
        SAVE1 = ONE5 - HALF * (Z * Z + VN * VN)
        UN = Z * SAVE1
        VN = VN * SAVE1
        KI = N + K
        L = N + 2 - K
        LI = N + L
        AN = X(K) + X(L)
        BN = X(KI) - X(LI)
        PN = X(K) - X(L)
        QN = X(KI) + X(LI)
        CN = UN * PN + VN * QN
        DN = UN * QN - VN * PN
        X(K) = AN - DN
        X(KI) = BN + CN
        X(L) = AN + DN
        X(LI) = CN - BN
    4 CONTINUE
C
C     Now do the inverse transform
C
      CALL FASTG(X, X(N+1), N, -1)
C
C     Now undo the order - the half arrays are already bit reversed;
C     bit reverse the whole array.
C
      CALL SCRAG(X, M, IPOW)
      RETURN
```

```
      END
C
      subroutine fastg(xreal, ximag, n, itype)
c
c     Algorithm AS 83.2 Appl. Statist. (1975) vol.24, no.1
c
c     Radix 4 complex discrete fast Fourier transform without
c     unscrambling, suitable for convolutions or other applications
c     which do not require unscrambling.   Called by subroutine
c     FASTF which also does the unscrambling.
c
      real    xreal(n), ximag(n)
      data    zero, half, one, one5, two, four
     +        /0.0,  0.5, 1.0,  1.5, 2.0,  4.0/
      pi = four * atan(one)
      ifaca = n / 4
      if (itype .eq. 0) return
      if (itype .gt. 0) go to 5
c
c     ITYPE < 0 indicates inverse transform required.
c     Calculate conjugate.
c
      do 4 k = 1, n
    4 ximag(k) = -ximag(k)
c
c     Following code is executed for IFACA = N/4, N/16, N/64, ...
c     until IFACA <= 1.
c
    5 ifcab = ifaca * 4
      z = pi / ifcab
      bcos = -two * sin(z)**2
      bsin = sin(two * z)
      cw1 = one
      sw1 = zero
      do 10 litla = 1, ifaca
        do 8 i0 = litla, n, ifcab
          i1 = i0 + ifaca
          i2 = i1 + ifaca
          i3 = i2 + ifaca
          xs0 = xreal(i0) + xreal(i2)
          xs1 = xreal(i0) - xreal(i2)
          ys0 = ximag(i0) + ximag(i2)
          ys1 = ximag(i0) - ximag(i2)
          xs2 = xreal(i1) + xreal(i3)
          xs3 = xreal(i1) - xreal(i3)
          ys2 = ximag(i1) + ximag(i3)
          ys3 = ximag(i1) - ximag(i3)
          xreal(i0) = xs0 + xs2
          ximag(i0) = ys0 + ys2
          x1 = xs1 + ys3
          y1 = ys1 - xs3
          x2 = xs0 - xs2
          y2 = ys0 - ys2
          x3 = xs1 - ys3
          y3 = ys1 + xs3
          if (litla .eq. 1) then
            xreal(i2) = x1
            ximag(i2) = y1
            xreal(i1) = x2
            ximag(i1) = y2
            xreal(i3) = x3
```

```
              ximag(i3) = y3
            else
              xreal(i2) = x1 * cw1 + y1 * sw1
              ximag(i2) = y1 * cw1 - x1 * sw1
              xreal(i1) = x2 * cw2 + y2 * sw2
              ximag(i1) = y2 * cw2 - x2 * sw2
              xreal(i3) = x3 * cw3 + y3 * sw3
              ximag(i3) = y3 * cw3 - x3 * sw3
            end if
     8    continue
c
c       Calculate a new set of twiddle factors.
c
          if (litla .lt. ifaca) then
            z = cw1 * bcos - sw1 * bsin + cw1
            sw1 = bcos * sw1 + bsin * cw1 + sw1
            tempr = one5 - half * (z * z + sw1 * sw1)
            cw1 = z * tempr
            sw1 = sw1 * tempr
            cw2 = cw1 * cw1 - sw1 * sw1
            sw2 = two * cw1 * sw1
            cw3 = cw1 * cw2 - sw1 * sw2
            sw3 = cw1 * sw2 + cw2 * sw1
          end if
    10    continue
          if (ifaca .le. 1) go to 14
c
c       Set up the transform split for the next stage.
c
          ifaca = ifaca / 4
          if (ifaca .gt. 0) go to 5
c
c       Radix 2 calculation, if needed.
c
          if (ifaca .lt. 0) return
          do 13 k = 1, n, 2
            tempr = xreal(k) + xreal(k+1)
            xreal(k+1) = xreal(k) - xreal(k+1)
            xreal(k) = tempr
            tempr = ximag(k) + ximag(k+1)
            ximag(k+1) = ximag(k) - ximag(k+1)
            ximag(k) = tempr
    13    continue
    14    if (itype .lt. 0) then
c
c       Inverse transform; conjugate the result.
c
            do 16 k = 1, n
    16      ximag(k) = -ximag(k)
            return
          end if
c
c       Forward transform
c
          z = one / n
          do 18 k = 1, n
            xreal(k) = xreal(k) * z
            ximag(k) = ximag(k) * z
    18    continue
c
          return
```

```
      end
c
c
c
      subroutine scrag(xreal, n, ipow)
c
c     Algorithm AS 83.3 Appl. Statist. (1975) vol.24, no.1
c ***   MODIFIED FOR USE WITH AS 97 ***
c
c     Subroutine for unscrambling FFT data.
c
      real    xreal(n)
      integer l(19)
      equivalence (l1,l(1)), (l2,l(2)), (l3,l(3)), (l4,l(4)),
     +          (l5,l(5)), (l6,l(6)), (l7,l(7)), (l8,l(8)), (l9,l(9)),
     +          (l10,l(10)), (l11,l(11)), (l12,l(12)), (l13,l(13)),
     +          (l14,l(14)), (l15,l(15)), (l16,l(16)), (l17,l(17)),
     +          (l18,l(18)), (l19,l(19))
c
      ii = 1
      itop = 2 ** (ipow - 1)
      i = 20 - ipow
      do 5 k = 1, i
    5 l(k) = ii
      l0 = ii
      i = i + 1
      do 6 k = i, 19
        ii = ii * 2
        l(k) = ii
    6 continue
c
      ii = 0
      do 9 j1 = 1, l1, l0
        do 9 j2 = j1, l2, l1
          do 9 j3 = j2, l3, l2
            do 9 j4 = j3, l4, l3
              do 9 j5 = j4, l5, l4
                do 9 j6 = j5, l6, l5
                  do 9 j7 = j6, l7, l6
                    do 9 j8 = j7, l8, l7
                      do 9 j9 = j8, l9, l8
                        do 9 j10 = j9, l10, l9
                          do 9 j11 = j10, l11, l10
                            do 9 j12 = j11, l12, l11
                              do 9 j13 = j12, l13, l12
                                do 9 j14 = j13, l14, l13
                                  do 9 j15 = j14, l15, l14
                                    do 9 j16 = j15, l16, l15
                                      do 9 j17 = j16, l17, l16
                                        do 9 j18 = j17, l18, l17
                                          do 9 j19 = j18, l19, l18
                                            j20 = j19
                                            do 9 i = 1, 2
                                              ii = ii + 1
                                              if (ii .lt. j20) then
c
c     J20 is the bit-reverse of II pairwise interchange.
c
                                                tempr = xreal(ii)
                                                xreal(ii) = xreal(j20)
                                                xreal(j20) = tempr
```

```
                                                      end if
                                                      j20 = j20 + itop
      9   continue
c
          return
          end
```

```
CSTART OF AS 99
      SUBROUTINE JNSN(XBAR, SD, RB1, BB2, ITYPE, GAMMA, DELTA,
     $  XLAM, XI, IFAULT)
C
C        ALGORITHM AS 99  APPL. STATIST. (1976) VOL.25, P.180
C
C        FINDS TYPE AND PARAMETERS OF A JOHNSON CURVE
C        WITH GIVEN FIRST FOUR MOMENTS
C
      REAL XBAR, SD, RB1, BB2, GAMMA, DELTA, XLAM, XI, TOL,
     $  B1, B2, Y, X, U, W, ZERO, ONE, TWO, THREE, FOUR, HALF,
     $  QUART, ZABS, ZEXP, ZLOG, ZSIGN, ZSQRT
      LOGICAL FAULT
C
      DATA TOL /0.01/
      DATA ZERO, QUART, HALF, ONE, TWO, THREE, FOUR
     $     /0.0,  0.25,  0.5, 1.0, 2.0,   3.0,  4.0/
C
      ZABS(X) = ABS(X)
      ZEXP(X) = EXP(X)
      ZLOG(X) = ALOG(X)
      ZSIGN(X, Y) = SIGN(X, Y)
      ZSQRT(X) = SQRT(X)
C
      IFAULT = 1
      IF (SD .LT. ZERO) RETURN
      IFAULT = 0
      XI = ZERO
      XLAM = ZERO
      GAMMA = ZERO
      DELTA = ZERO
      IF (SD .GT. ZERO) GOTO 10
      ITYPE = 5
      XI = XBAR
      RETURN
   10 B1 = RB1 * RB1
      B2 = BB2
      FAULT = .FALSE.
C
C        TEST WHETHER LOGNORMAL (OR NORMAL) REQUESTED
C
      IF (B2 .GE. ZERO) GOTO 30
   20 IF (ZABS(RB1) .LE. TOL) GOTO 70
      GOTO 80
C
C        TEST FOR POSITION RELATIVE TO BOUNDARY LINE
C
   30 IF (B2 .GT. B1 + TOL + ONE) GOTO 60
      IF (B2 .LT. B1 + ONE) GOTO 50
C
C        ST DISTRIBUTION
C
   40 ITYPE = 5
      Y = HALF + HALF * ZSQRT(ONE - FOUR / (B1 + FOUR))
      IF (RB1 .GT. ZERO) Y = ONE - Y
      X = SD / ZSQRT(Y * (ONE - Y))
      XI = XBAR - Y * X
      XLAM = XI + X
      DELTA = Y
      RETURN
   50 IFAULT = 2
```

```
      RETURN
   60 IF (ZABS(RB1) .GT. TOL .OR. ZABS(B2 - THREE) .GT. TOL) GOTO 80
C
C        NORMAL DISTRIBUTION
C
   70 ITYPE = 4
      DELTA = ONE / SD
      GAMMA = -XBAR / SD
      RETURN
C
C        TEST FOR POSITION RELATIVE TO LOGNORMAL LINE
C
   80 X = HALF * B1 + ONE
      Y = ZABS(RB1) * ZSQRT(QUART * B1 + ONE)
      U = (X + Y) ** (ONE / THREE)
      W = U + ONE / U - ONE
      U = W * W * (THREE + W * (TWO + W)) - THREE
      IF (B2 .LT. ZERO .OR. FAULT) B2 = U
      X = U - B2
      IF (ZABS(X) .GT. TOL) GOTO 90
C
C        LOGNORMAL (SL) DISTRIBUTION
C
      ITYPE = 1
      XLAM = ZSIGN(ONE, RB1)
      U = XLAM * XBAR
      X = ONE / ZSQRT(ZLOG(W))
      DELTA = X
      Y = HALF * X * ZLOG(W * (W - ONE) / (SD * SD))
      GAMMA = Y
      XI = XLAM * (U - ZEXP((HALF / X - Y) / X))
      RETURN
C
C        SB OR SU DISTRIBUTION
C
   90 IF (X .GT. ZERO) GOTO 100
      ITYPE = 2
      CALL SUFIT(XBAR, SD, RB1, B2, GAMMA, DELTA, XLAM, XI)
      RETURN
  100 ITYPE = 3
      CALL SBFIT(XBAR, SD, RB1, B2, GAMMA, DELTA, XLAM, XI, FAULT)
      IF (.NOT. FAULT) RETURN
C
C        FAILURE - TRY TO FIT APPROXIMATE RESULT
C
      IFAULT = 3
      IF (B2 .GT. B1 + TWO) GOTO 20
      GOTO 40
      END
C
      SUBROUTINE SUFIT (XBAR, SD, RB1, B2, GAMMA, DELTA, XLAM, XI)
C
C        ALGORITHM AS 99.1  APPL. STATIST. (1976) VOL.25, P.180
C
C        FINDS PARAMETERS OF JOHNSON SU CURVE WITH
C        GIVEN FIRST FOUR MOMENTS
C
      REAL XBAR, SD, RB1, B2, GAMMA, DELTA, XLAM, XI, TOL, B1,
     $  B3, W, Y, W1, WM1, Z, V, A, B, X, ZERO, ONE, TWO, THREE,
     $  FOUR, SIX, SEVEN, EIGHT, NINE, TEN, HALF, ONE5, TWO8,
     $  SIXTEN, ZABS, ZEXP, ZLOG, ZSIGN, ZSQRT
```

```
C
      DATA TOL /0.01/
      DATA ZERO,  ONE,  TWO,  THREE, FOUR,  SIX, SEVEN,
     $    EIGHT, NINE,  TEN, SIXTEN, HALF, ONE5,  TWO8
     $     /0.0,  1.0,  2.0,    3.0,  4.0,  6.0,   7.0,
     $      8.0,  9.0, 10.0,   16.0,  0.5,  1.5,   2.8/
C
      ZABS(X) = ABS(X)
      ZEXP(X) = EXP(X)
      ZLOG(X) = ALOG(X)
      ZSIGN(X, Y) = SIGN(X, Y)
      ZSQRT(X) = SQRT(X)
C
      B1 = RB1 * RB1
      B3 = B2 - THREE
C
C        W IS FIRST ESTIMATE OF EXP(DELTA ** (-2))
C
      W = ZSQRT(TWO * B2 - TWO8 * B1 - TWO)
      W = ZSQRT(W-ONE)
      IF (ZABS(RB1) .GT. TOL) GOTO 10
C
C        SYMMETRICAL CASE - RESULTS ARE KNOWN
C
      Y = ZERO
      GOTO 20
C
C        JOHNSON ITERATION (USING Y FOR HIS M)
C
   10 W1 = W + ONE
      WM1 = W - ONE
      Z = W1 * B3
      V = W * (SIX + W * (THREE + W))
      A = EIGHT * (WM1 * (THREE + W * (SEVEN + V)) - Z)
      B = SIXTEN * (WM1 * (SIX + V) - B3)
      Y = (ZSQRT(A * A - TWO * B * (WM1 * (THREE + W *
     $  (NINE + W * (TEN + V))) - TWO * W1 * Z)) - A) / B
      Z = Y * WM1 * (FOUR * (W + TWO) * Y + THREE * W1 * W1) ** 2 /
     $  (TWO * (TWO * Y + W1) ** 3)
      V = W * W
      W = ZSQRT(ONE - TWO * (ONE5 - B2 + (B1 *
     $  (B2 - ONE5 - V * (ONE + HALF * V))) / Z))
      W = ZSQRT(W-ONE)
      IF (ZABS(B1 - Z) .GT. TOL) GOTO 10
C
C        END OF ITERATION
C
      Y = Y / W
      Y = ZLOG(ZSQRT(Y) + ZSQRT(Y + ONE))
      IF (RB1 .GT. ZERO) Y = -Y
   20 X = ZSQRT(ONE / ZLOG(W))
      DELTA = X
      GAMMA = Y * X
      Y = ZEXP(Y)
      Z = Y * Y
      X = SD / ZSQRT(HALF * (W - ONE) * (HALF * W *
     $  (Z + ONE / Z) + ONE))
      XLAM = X
      XI = (HALF * ZSQRT(W) * (Y - ONE / Y)) * X + XBAR
      RETURN
      END
```

```
      SUBROUTINE SBFIT(XBAR, SIGMA, RTB1, B2, GAMMA, DELTA, XLAM,
     $  XI, FAULT)
C
C        ALGORITHM AS 99.2  APPL. STATIST. (1976) VOL.25, P.180
C
C        FINDS PARAMETERS OF JOHNSON SB CURVE WITH
C        GIVEN FIRST FOUR MOMENTS
C
      REAL HMU(6), DERIV(4), DD(4), XBAR, SIGMA, RTB1, B2, GAMMA,
     $  DELTA, XLAM, XI, TT, TOL, RB1, B1, E, U, X, Y, W, F, D,
     $  G, S, H2, T, H2A, H2B, H3, H4, RBET, BET2, ZERO, ONE,
     $  TWO, THREE, FOUR, SIX, HALF, QUART, ONE5, A1, A2, A3,
     $  A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15,
     $  A16, A17, A18, A19, A20, A21, A22, ZABS, ZLOG, ZSQRT
      LOGICAL NEG, FAULT
C
      DATA TT, TOL, LIMIT /1.0E-4, 0.01, 50/
      DATA ZERO, ONE, TWO, THREE, FOUR, SIX, HALF, QUART, ONE5
     $     /0.0, 1.0, 2.0,   3.0,  4.0, 6.0,  0.5,  0.25,  1.5/
      DATA     A1,     A2,     A3,     A4,     A5,     A6,
     $         A7,     A8,     A9,    A10,    A11,    A12,
     $        A13,    A14,    A15,    A16,    A17,    A18,
     $        A19,    A20,    A21,    A22
     $    /0.0124, 0.0623, 0.4043,  0.408,  0.479,  0.485,
     $     0.5291, 0.5955,  0.626,   0.64, 0.7077, 0.7466,
     $        0.8, 0.9281, 1.0614,   1.25, 1.7973,    1.8,
     $      2.163,    2.5, 8.5245, 11.346/
C
      ZABS(X) = ABS(X)
      ZLOG(X) = ALOG(X)
      ZSQRT(X) = SQRT(X)
C
      RB1 = ZABS(RTB1)
      B1 = RB1 * RB1
      NEG = RTB1 .LT. ZERO
C
C        GET D AS FIRST ESTIMATE OF DELTA
C
      E = B1 + ONE
      X = HALF * B1 + ONE
      Y = ZABS(RB1) * ZSQRT(QUART * B1 + ONE)
      U = (X + Y) ** (ONE / THREE)
      W = U + ONE / U - ONE
      F = W * W * (THREE + W * (TWO + W)) - THREE
      E = (B2 - E) / (F - E)
      IF (ZABS(RB1) .GT. TOL) GOTO 5
      F = TWO
      GOTO 20
    5 D = ONE / ZSQRT(ZLOG(W))
      IF (D .LT. A10) GOTO 10
      F = TWO - A21 / (D * (D * (D - A19) + A22))
      GOTO 20
   10 F = A16 * D
   20 F = E * F + ONE
      IF (F .LT. A18) GOTO 25
      D = (A9 * F - A4) * (THREE - F) ** (-A5)
      GOTO 30
   25 D = A13 * (F - ONE)
C
C        GET G AS FIRST ESTIMATE OF GAMMA
C
```

```
      30 G = ZERO
         IF (B1 .LT. TT) GOTO 70
         IF (D .GT. ONE) GOTO 40
         G = (A12 * D ** A17 + A8) * B1 ** A6
         GOTO 70
      40 IF (D .LE. A20) GOTO 50
         U = A1
         Y = A7
         GOTO 60
      50 U = A2
         Y = A3
      60 G = B1 ** (U * D + Y) * (A14 + D * (A15 * D - A11))
      70 M = 0
C
C        MAIN ITERATION STARTS HERE
C
      80 M = M + 1
         FAULT = M .GT. LIMIT
         IF (FAULT) RETURN
C
C        GET FIRST SIX MOMENTS FOR LATEST G AND D VALUES
C
         CALL MOM(G, D, HMU, FAULT)
         IF (FAULT) RETURN
         S = HMU(1) * HMU(1)
         H2 = HMU(2) - S
         FAULT = H2 .LE. ZERO
         IF (FAULT) RETURN
         T = ZSQRT(H2)
         H2A = T * H2
         H2B = H2 * H2
         H3 = HMU(3) - HMU(1) * (THREE * HMU(2) - TWO * S)
         RBET = H3 / H2A
         H4 = HMU(4) - HMU(1) * (FOUR * HMU(3) - HMU(1) *
        $  (SIX * HMU(2) - THREE * S))
         BET2 = H4 / H2B
         W = G * D
         U = D * D
C
C        GET DERIVATIVES
C
         DO 120 J = 1, 2
         DO 110 K = 1, 4
         T = K
         IF (J .EQ. 1) GOTO 90
         S = ((W - T) * (HMU(K) - HMU(K + 1)) + (T + ONE) *
        $  (HMU(K + 1) - HMU(K + 2))) / U
         GOTO 100
      90 S = HMU(K + 1) - HMU(K)
     100 DD(K) = T * S / D
     110 CONTINUE
         T = TWO * HMU(1) * DD(1)
         S = HMU(1) * DD(2)
         Y = DD(2) - T
         DERIV(J) = (DD(3) - THREE * (S + HMU(2) * DD(1) - T * HMU(1))
        $  - ONE5 * H3 * Y / H2) / H2A
         DERIV(J + 2) = (DD(4) - FOUR * (DD(3) * HMU(1) + DD(1) * HMU(3))
        $  + SIX * (HMU(2) * T + HMU(1) * (S - T * HMU(1)))
        $  - TWO * H4 * Y / H2) / H2B
     120 CONTINUE
         T = ONE / (DERIV(1) * DERIV(4) - DERIV(2) * DERIV(3))
```

```
      U = (DERIV(4) * (RBET - RB1) - DERIV(2) * (BET2 - B2)) * T
      Y = (DERIV(1) * (BET2 - B2) - DERIV(3) * (RBET - RB1)) * T
C
C        FORM NEW ESTIMATES OF G AND D
C
      G = G - U
      IF (B1 .EQ. ZERO .OR. G .LT. ZERO) G = ZERO
      D = D - Y
      IF (ZABS(U) .GT. TT .OR. ZABS(Y) .GT. TT) GOTO 80
C
C        END OF ITERATION
C
      DELTA = D
      XLAM = SIGMA / ZSQRT(H2)
      IF (NEG) GOTO 130
      GAMMA = G
      GOTO 140
  130 GAMMA = -G
      HMU(1) = ONE - HMU(1)
  140 XI = XBAR - XLAM * HMU(1)
      RETURN
      END
C
      SUBROUTINE MOM(G, D, A, FAULT)
C
C        ALGORITHM AS 99.3  APPL. STATIST. (1976) VOL.25, P.180
C
C        EVALUATES FIRST SIX MOMENTS OF A JOHNSON
C        SB DISTRIBUTION, USING GOODWIN METHOD
C
      REAL A(6), B(6), C(6), G, D, ZZ, VV, RTTWO, RRTPI, W, E, R,
     $  H, T, U, Y, X, V, F, Z, S, P, Q, AA, AB, EXPA, EXPB,
     $  ZERO, QUART, HALF, P75, ONE, TWO, THREE, ZABS, ZEXP
      LOGICAL L, FAULT
C
      DATA ZZ, VV, LIMIT /1.0E-5, 1.0E-8, 500/
C
C        RTTWO IS SQRT(2.0)
C        RRTPI IS RECIPROCAL OF SQRT(PI)
C        EXPA IS A VALUE SUCH THAT EXP(EXPA) DOES NOT QUITE
C          CAUSE OVERFLOW
C        EXPB IS A VALUE SUCH THAT 1.0 + EXP(-EXPB) MAY BE
C          TAKEN TO BE 1.0
C
      DATA     RTTWO,         RRTPI, EXPA, EXPB
     $  /1.414213562, 0.5641895835, 80.0, 23.7/
      DATA ZERO, QUART, HALF,  P75, ONE, TWO, THREE
     $     /0.0,  0.25,  0.5, 0.75, 1.0, 2.0,   3.0/
C
      ZABS(X) = ABS(X)
      ZEXP(X) = EXP(X)
C
      FAULT = .FALSE.
      DO 10 I = 1, 6
   10 C(I) = ZERO
      W = G / D
C
C        TRIAL VALUE OF H
C
      IF (W .GT. EXPA) GOTO 140
      E = ZEXP(W) + ONE
```

```
      R = RTTWO / D
      H = P75
      IF (D .LT. THREE) H = QUART * D
      K = 1
      GOTO 40
C
C        START OF OUTER LOOP
C
   20 K = K + 1
      IF (K .GT. LIMIT) GOTO 140
      DO 30 I = 1, 6
   30 C(I) = A(I)
C
C        NO CONVERGENCE YET - TRY SMALLER H
C
      H = HALF * H
   40 T = W
      U = T
      Y = H * H
      X = TWO * Y
      A(1) = ONE / E
      DO 50 I = 2, 6
   50 A(I) = A(I - 1) / E
      V = Y
      F = R * H
      M = 0
C
C        START OF INNER LOOP
C        TO EVALUATE INFINITE SERIES
C
   60 M = M + 1
      IF (M .GT. LIMIT) GOTO 140
      DO 70 I = 1, 6
   70 B(I) = A(I)
      U = U - F
      Z = ONE
      IF (U .GT. -EXPB) Z = ZEXP(U) + Z
      T = T + F
      L = T .GT. EXPB
      IF (.NOT. L) S = ZEXP(T) + ONE
      P = ZEXP(-V)
      Q = P
      DO 90 I = 1, 6
      AA = A(I)
      P = P / Z
      AB = AA
      AA = AA + P
      IF (AA .EQ. AB) GOTO 100
      IF (L) GOTO 80
      Q = Q / S
      AB = AA
      AA = AA + Q
      L = AA .EQ. AB
   80 A(I) = AA
   90 CONTINUE
  100 Y = Y + X
      V = V + Y
      DO 110 I = 1, 6
      IF (A(I) .EQ. ZERO) GOTO 140
      IF (ZABS((A(I) - B(I)) / A(I)) .GT. VV) GOTO 60
  110 CONTINUE
```

```
C
C        END OF INNER LOOP
C
      V = RRTPI * H
      DO 120 I = 1, 6
  120 A(I) = V * A(I)
      DO 130 I = 1, 6
      IF (A(I) .EQ. ZERO) GOTO 140
      IF (ZABS((A(I) - C(I)) / A(I)) .GT. ZZ) GOTO 20
  130 CONTINUE
C
C        END OF OUTER LOOP
C
      RETURN
  140 FAULT =.TRUE.
      RETURN
      END
CEND OF AS 99
```

```
CSTART OF AS 100
      REAL FUNCTION AJV(SNV, ITYPE, GAMMA, DELTA, XLAM, XI, IFAULT)
C
C        ALGORITHM AS 100.1  APPL. STATIST. (1976) VOL.25, P.190
C
C        CONVERTS A STANDARD NORMAL VARIATE (SNV) TO A
C        JOHNSON VARIATE (AJV)
C
      REAL SNV, GAMMA, DELTA, XLAM, XI, V, W, ZERO, HALF, ONE,
     $  ZABS, ZEXP, ZSIGN
C
      DATA ZERO, HALF, ONE /0.0, 0.5, 1.0/
C
      ZABS(W) = ABS(W)
      ZEXP(W) = EXP(W)
      ZSIGN(W, V) = SIGN(W, V)
C
      AJV = ZERO
      IFAULT = 1
      IF (ITYPE .LT. 1 .OR. ITYPE .GT. 4) RETURN
      IFAULT = 0
      GOTO (10, 20, 30, 40), ITYPE
C
C        SL DISTRIBUTION
C
   10 AJV = XLAM * ZEXP((XLAM * SNV - GAMMA) / DELTA) + XI
      RETURN
C
C        SU DISTRIBUTION
C
   20 W = ZEXP((SNV - GAMMA) / DELTA)
      W = HALF * (W - ONE / W)
      AJV = XLAM * W + XI
      RETURN
C
C        SB DISTRIBUTION
C
   30 W = (SNV - GAMMA) / DELTA
      V = ZEXP(-ZABS(W))
      V = (ONE - V) / (ONE + V)
      AJV = HALF * XLAM * (ZSIGN(V, W) + ONE) + XI
      RETURN
C
C        NORMAL DISTRIBUTION
C
   40 AJV = (SNV - GAMMA) / DELTA
      RETURN
      END
C
      REAL FUNCTION SNV(AJV, ITYPE, GAMMA, DELTA, XLAM, XI, IFAULT)
C
C        ALGORITHM AS 100.2  APPL. STATIST. (1976) VOL.25, P.190
C
C        CONVERTS A JOHNSON VARIATE (AJV) TO A
C        STANDARD NORMAL VARIATE (SNV)
C
      REAL AJV, GAMMA, DELTA, XLAM, XI, V, W, C, ZERO, HALF, ONE,
     $  ZLOG, ZSQRT
C
      DATA ZERO, HALF, ONE, C /0.0, 0.5, 1.0, -63.0/
C
```

```
      ZLOG(W) = ALOG(W)
      ZSQRT(W) = SQRT(W)
C
      SNV = ZERO
      IFAULT = 1
      IF (ITYPE .LT. 1 .OR. ITYPE .GT. 4) RETURN
      IFAULT = 0
      GOTO (10, 20, 30, 40), ITYPE
C
C        SL DISTRIBUTION
C
   10 W = XLAM * (AJV - XI)
      IF (W .LE. ZERO) GOTO 15
      SNV = XLAM * (ZLOG(W) * DELTA + GAMMA)
      RETURN
   15 IFAULT = 2
      RETURN
C
C        SU DISTRIBUTION
C
   20 W = (AJV - XI) / XLAM
      IF (W .GT. C) GOTO 23
      W = -HALF / W
      GOTO 27
   23 W = ZSQRT(W * W + ONE) + W
   27 SNV = ZLOG(W) * DELTA + GAMMA
      RETURN
C
C        SB DISTRIBUTION
C
   30 W = AJV - XI
      V = XLAM - W
      IF (W .LE. ZERO .OR. V .LE. ZERO) GOTO 35
      SNV = ZLOG(W / V) * DELTA + GAMMA
      RETURN
   35 IFAULT = 2
      RETURN
C
C        NORMAL DISTRIBUTION
C
   40 SNV = DELTA * AJV + GAMMA
      RETURN
      END
CEND OF AS 100
```

```
      REAL FUNCTION DIGAMA(X, IFAULT)
C
C     ALGORITHM AS 103  APPL. STATIST. (1976) VOL.25, NO.3
C
C     Calculates DIGAMMA(X) = D( LOG( GAMMA(X))) / DX
C
      REAL ZERO, HALF, ONE
C
C     Set constants, SN = Nth Stirling coefficient, D1 = DIGAMMA(1.0)
C
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/
      DATA S, C, S3, S4, S5, D1 /1.E-05, 8.5, 8.333333333E-02,
     *    8.3333333333E-03, 3.96825 3968E-03, -0.57721 56649/
C
C     Check argument is positive
C
      DIGAMA = ZERO
      Y = X
      IFAULT = 1
      IF (Y .LE. ZERO) RETURN
      IFAULT = 0
C
C     Use approximation if argument <= S
C
      IF (Y .LE. S) THEN
        DIGAMA = D1 - ONE / Y
        RETURN
      END IF
C
C     Reduce to DIGAMA(X + N) where (X + N) >= C
C
    1 IF (Y .GE. C) GO TO 2
      DIGAMA = DIGAMA - ONE/Y
      Y = Y + ONE
      GO TO 1
C
C     Use Stirling's (actually de Moivre's) expansion if argument > C
C
    2 R = ONE / Y
      DIGAMA = DIGAMA + LOG(Y) - HALF*R
      R = R * R
      DIGAMA = DIGAMA - R*(S3 - R*(S4 - R*S5))
      RETURN
      END
```

```
      SUBROUTINE SEQUAL(BDRY, LBDRY, VEC, LVEC, NSTRD, LNSTRD, NDECNS,
     *    NSTOP, PROBS, IPROB1, IPROB2, TSTEPS, INCS, UNITT, RESULT,
     *    IFAULT)
C
C     ALGORITHM AS 107  APPL.STATIST. (1977), VOL.26, NO.1
C
C     Calculates the first (M-1) operating characteristics and the
C     average sampling number, given the parameter, for a general closed
C     sequential sampling plan with M terminal decisions.
C
      INTEGER LBDRY, LVEC, LNSTRD, NSTRD(LNSTRD), NDECNS, NSTOP, IPROB1,
     *    IPROB2, INCS(IPROB2), IFAULT
      REAL BDRY(LBDRY), VEC(LVEC), PROBS(IPROB2), TSTEPS(IPROB2), UNITT,
     *    RESULT(NDECNS)
C
C     Local variables
C
      INTEGER NL, NPAIRS, NBDRYS, JJ, N, IBASEC, IBASEL, NPC, NPL, KPC,
     *    L1, L2, J, IT, ITLIM, IBC, KKPC, KNPC, IR, IB, KKPL, KPL,
     *    KNPL, L, K, NS, LL, JPOS
      LOGICAL FLAG
      REAL U2, Z, T, TLIM, PN, PXGTH
      REAL SMALL, ZERO, HALF, ONE
C
C     SMALL is a machine-dependent constant used to test that TSTEPS
C     values divided by UNITT equal INCS values.
C
      DATA SMALL /1.E-06/, ZERO /0.0/, HALF /0.5/, ONE /1.0/
C
      NL = (LVEC / 2) / NDECNS
      NPAIRS = NDECNS - 1
      NBDRYS = 2 * NPAIRS
C
C     Test for admissibility of parameters.
C
      IFAULT = 1
      IF (NDECNS .LE. 1) RETURN
      IFAULT = 2
      IF (NSTOP .LE. 0) RETURN
      IFAULT = 3
      IF (IPROB1 .LE. 0) RETURN
      IFAULT = 4
      IF (IPROB2 .LE. IPROB1) RETURN
      IFAULT = 5
      IF (LVEC .LE. 0) RETURN
      IFAULT = 6
      IF (LNSTRD .LT. NBDRYS) RETURN
      IFAULT = 7
      IF (LBDRY .LT. NBDRYS * NSTOP) RETURN
C
      U2 = HALF * UNITT
      Z = UNITT * U2
      JJ = 2
C
C     Test that sampling plan is closed at observation count NSTOP.
C
      IFAULT = 10
      DO 10 N = 1, NPAIRS
        IF (BDRY(JJ) - BDRY(JJ-1) .GT. Z) GO TO 190
        JJ = JJ + 2
   10 CONTINUE
```

```
C
C     Test that TSTEPS values divided by UNITT equal INCS values.
C
      IFAULT = 100
      DO 20 N = IPROB1, IPROB2
        T = INCS(N)
        IF (ABS(TSTEPS(N) / UNITT - T) .GT. SMALL) GO TO 190
   20 CONTINUE
C
      IFAULT = 0
      IBASEC = 0
      IBASEL = 0
      NPC = 0
      NPL = NPAIRS
      N = NSTOP - 1
      FLAG = .FALSE.
      KPC = 0
      L1 = NPL + 1
      L2 = NPL + NPAIRS
      DO 30 J = L1, L2
   30 NSTRD(J) = 0
C
C     Enter backward instruction cycle.
C
   40 KPL = KPC
      IF (N .GT. 0) GO TO 50
      FLAG = .TRUE.
      T = ZERO
      GO TO 80
   50 KPC = KPC + NBDRYS
      IT = IBASEC
      ITLIM = IT + NL
      IBC = IBASEC
      J = 1
      KKPC = KPC
      KNPC = NPC
C
C     Enter loop through boundary pairs for sample size N.
C
   60 KKPC = KKPC + 2
      TLIM = BDRY(KKPC) - U2
      T = BDRY(KKPC-1)
C
C     Enter cycle to compute results for each T in continuation region
C     at sample size N.
C
   70 T = T + UNITT
      IF (T .GT. TLIM) GO TO 170
      IT = IT + 1
C
C     Test that number of continuation states does not exceed the limit.
C
      IF (IT .GT. ITLIM) GO TO 180
C
   80 DO 90 IR = 1, NPAIRS
   90 RESULT(IR) = ZERO
      RESULT(NDECNS) = ONE
      IB = IBASEL
      KKPL = KPL
      KNPL = NPL
      L1 = IPROB1
```

```
C
C     Loop through reachable continuation and stopping regions at
C     sample size (N+1).
C
      DO 140 K = 1, NPAIRS
        KKPL = KKPL + 2
        KNPL = KNPL + 1
        NS = NSTRD(KNPL)
        Z = T - BDRY(KKPL-1)
        JJ = Z / UNITT + SIGN(HALF, Z - U2)
        JJ = JJ + IB - 1
        Z = U2 - Z
        PN = ZERO
C
C     Loop through stop region for decision K.
C
        DO 100 L = L1, IPROB2
          IF (TSTEPS(L) .GT. Z) GO TO 110
          PN = PN + PROBS(L)
  100   CONTINUE
        RESULT(K) = RESULT(K) + PN
        GO TO 150
  110   L1 = L
        RESULT(K) = RESULT(K) + PN
        IF (NS .EQ. 0) GO TO 140
        Z = BDRY(KKPL) - T - U2
C
C     Loop through continuation region between decisions K and (K+1).
C
        DO 120 LL = L1, IPROB2
          IF (TSTEPS(LL) .GT. Z) GO TO 130
          JPOS = (JJ + INCS(LL)) * NDECNS
          PXGTH = PROBS(LL)
          DO 120 IR = 1, NDECNS
            JPOS = JPOS + 1
            RESULT(IR) = RESULT(IR) + PXGTH * VEC(JPOS)
  120   CONTINUE
        GO TO 150
  130   IB = IB + NS
        L1 = LL
  140 CONTINUE
C
C     Test for completion of calculations.
C
  150 IF (FLAG) RETURN
C
C     Store results for (N, T).
C
      JPOS = (IT - 1) * NDECNS
      DO 160 IR = 1, NDECNS
        JPOS = JPOS + 1
        VEC(JPOS) = RESULT(IR)
  160 CONTINUE
      GO TO 70
C
  170 KNPC = KNPC + 1
      NSTRD(KNPC) = IT - IBC
      IBC = IT
      J = J + 1
      IF (J .LE. NPAIRS) GO TO 60
C
```

```
C     End of calculations for sample size N.
C
      IBASEL = IBASEC
      IBASEC = NL - IBASEC
      NPL = NPC
      N = N - 1
      GO TO 40
C
  180 IFAULT = 1000
  190 IFAULT = IFAULT + N
      RETURN
      END
```

```
      SUBROUTINE MSAE(K, N, KPN, IMAX, JMAX, X, XBAR, Y, YBAR, ALPHA, B,
     +  LSAE, IDEP, IFAULT, A, C, IS, NB)
C
C     ALGORITHM AS 108  APPL. STATIST. (1977), VOL. 26, NO.1
C
C     Computes the minimum sum of absolute errors (MSAE) estimates of
C     unknown parameters ALPHA, B(1), B(2), ... , B(K) in the linear
C     regression equation:
C
C     Y(I) = ALPHA + B(1)X(1) + B(2)X(2) + ... + B(K)X(K)
C
C     The regression line passes through the point
C     ( XBAR(1), XBAR(2), ... , XBAR(K), YBAR ) ,
C     where these are the mean values of these variables,
C     though the values of XBAR() and YBAR are input by the user, and
C     could be any point through which the line is to be forced.
C     Users should read the criticism of this algorithm on page 378 of
C     volume 27 of Applied Statistics (1978).
C     One way to use this algorithm would be to fix the XBAR's, say
C     equal to the means, but to vary the value of YBAR until the minimum
C     value of LSAE is found.
C
C     An alternative source of Fortran algorithms for this task is:
C        Gonin, R. and Money, A.H. (1989) Nonlinear Lp-norm estimation,
C        Dekker: New York.
C
C     N.B. The user should check the output values of both IFAULT & IDEP.
C
      REAL A(IMAX, JMAX), B(K), C(KPN), X(N, K), XBAR(K), Y(N)
      INTEGER IS(IMAX), NB(JMAX)
      LOGICAL DONE
      REAL SMALL, TOL, ZERO, ONE, TWO
      REAL LSAE
      DATA SMALL/-1.E+10/, TOL/1.E-08/, ZERO/0.0/, ONE/1.0/, TWO/2.0/
C
C     Check for parameter consistency
C
      IFAULT = 0
      KP1 = K + 1
      KN = K + N
      NP1 = N + 1
      IF (K .GT. N) IFAULT = 1
      IF (IMAX .NE. NP1) IFAULT = 2
      IF (JMAX .NE. KP1) IFAULT = 3
      IF (KPN .NE. KN) IFAULT = 4
      IF (IFAULT .GT. 0) RETURN
C
C     Set up the initial tableau.
C
      LCOL = JMAX - 1
      DO 1 J = 1, JMAX
    1 A(1, J) = ZERO
      DO 10 I = 1, K
        B(I) = ZERO
        C(I) = ZERO
        NB(I) = I
   10 CONTINUE
      DO 20 I = KP1, KPN
   20 C(I) = TWO
      DO 30 J = 1, K
        DO 30 I = 1, N
```

```
            A(I+1, J) = X(I, J) - XBAR(J)
   30 CONTINUE
      DO 40 I = 1, N
        IS(I+1) = I + K
        A(I+1, JMAX) = Y(I) - YBAR
   40 CONTINUE
C
C     Determine the variable to leave the basis.
C
   50 H = -TOL
      ICAND = 0
      DONE = .TRUE.
      DO 80 I = 2, IMAX
        AA = A(I, JMAX)
        IF (AA .GE. H) GO TO 80
        DONE = .FALSE.
        H = AA
        ICAND = I
   80 CONTINUE
      IF (DONE) GO TO 200
C
C     Determine the variable to enter the basis.
C
      JCAND = 0
      RATIO = SMALL
      DO 110 J = 1, LCOL
        IONE = 1
        AA = A(ICAND, J)
        IF (ABS(AA) .LT. TOL) GO TO 110
        RCOST = A(1, J)
        IF (AA .LT. -TOL) GO TO 90
        IONE = -1
        IF (ABS(NB(J)) .GT. K) RCOST = RCOST - TWO
   90   R = RCOST / AA
        IF (R .LE. RATIO) GO TO 110
        JCAND = J * IONE
        RATIO = R
        RSAVE = RCOST
  110 CONTINUE
C
C     Determine if an ordinary simplex pivot is unnecessary.
C
      IT = IS(ICAND)
      II = ABS(IT)
      CJ = C(II)
      IF (RATIO .GT. -CJ) GO TO 140
      IS(ICAND) = -IS(ICAND)
      DO 130 J = 1, JMAX
        A(1, J) = A(1, J) + CJ * A(ICAND, J)
        A(ICAND, J) = -A(ICAND, J)
  130 CONTINUE
      GO TO 50
C
C     Perform ordinary simplex pivot.
C
  140 WUN = ONE
      IF (JCAND .GT. 0) GO TO 160
      JCAND = -JCAND
      NB(JCAND) = -NB(JCAND)
      WUN = -ONE
      A(1, JCAND) = RSAVE
```

```
  160 PIVOT = A(ICAND, JCAND) * WUN
      DO 170 J = 1, JMAX
  170 A(ICAND, J) = A(ICAND, J) / PIVOT
      DO 190 I = 1, IMAX
        IF (I .EQ. ICAND) GO TO 190
        AIJ = A(I, JCAND) * WUN
        IF (AIJ .EQ. ZERO) GO TO 190
        DO 180 J = 1, JMAX
          A(I, J) = A(I, J) - A(ICAND, J) * AIJ
  180   CONTINUE
        A(I, JCAND) = -AIJ / PIVOT
  190 CONTINUE
      A(ICAND, JCAND) = ONE / PIVOT
      IS(ICAND) = NB(JCAND)
      NB(JCAND) = IT
      GO TO 50
C
C     Compute ALPHA and the vector B containing the slopes.
C
  200 ALPHA = YBAR
      DO 220 I = 2, IMAX
        WUN = ONE
        II = IS(I)
        IF (ABS(II) .GT. K) GO TO 220
        IF (II .GT. 0) GO TO 210
        II = -II
        WUN = -ONE
  210   B(II) = WUN * A(I, JMAX)
        ALPHA = ALPHA - XBAR(II) * B(II)
  220 CONTINUE
      LSAE = -A(1, JMAX)
C
C     Inspect the final solution for dependencies amongst the predictor
C     variables.
C
      IDEP = 1
      DO 300 J = 1, LCOL
        IF (ABS(NB(J)) .GT. K) GO TO 300
        IDEP = 0
        RETURN
  300 CONTINUE
C
      RETURN
      END
```

```
      double precision function xinbta(p,q,beta,alpha,ifault)
      implicit double precision (a-h,o-z)
c
c     algorithm as 109 appl. statist. (1977), vol.26, no.1
c     (replacing algorithm as 64  appl. statist. (1973),
c     vol.22, no.3)
c
c     Remark AS R83 and the correction in vol40(1) p.236 have been
c     incorporated in this version.
c
c     Computes inverse of the incomplete beta function
c     ratio for given positive values of the arguments
c     p and q, alpha between zero and one.
c     log of complete beta function, beta, is assumed to be known.
c
c     Auxiliary function required: BETAIN = algorithm AS63
c
      logical indx
c
c     Define accuracy and initialise.
c     SAE below is the most negative decimal exponent which does not
c     cause an underflow; a value of -308 or thereabouts will often be
c     OK in double precision.
c
c     data acu/1.0d-14/
      data SAE/-37.D0/
      data zero/0.0d0/, one/1.0d0/, two/2.0d0/
      data three/3.0d0/, four/4.0d0/, five/5.0d0/, six/6.0d0/
c
      fpu = 10.d0 ** sae
      xinbta = alpha
c
c     test for admissibility of parameters
c
      ifault = 1
      if (p.le.zero .or. q.le.zero) return
      ifault = 2
      if (alpha.lt.zero .or. alpha.gt.one) return
      ifault = 0
      if (alpha.eq.zero .or. alpha.eq.one) return
c
c     change tail if necessary
c
      if (alpha.le.0.5d0) goto 1
      a = one-alpha
      pp = q
      qq = p
      indx = .true.
      goto 2
    1 a = alpha
      pp = p
      qq = q
      indx = .false.
c
c     calculate the initial approximation
c
    2 r = dsqrt(-dlog(a*a))
      y = r-(2.30753d0+0.27061d0*r)/(one+(0.99229d0+0.04481d0*r)*r)
      if(pp.gt.one .and. qq.gt.one) goto 5
      r = qq+qq
      t = one/(9.0d0*qq)
```

```
        t = r*(one-t+y*dsqrt(t))**3
        if(t.le.zero) goto 3
        t = (four*pp+r-two)/t
        if(t.le.one) goto 4
        xinbta = one-two/(t+one)
        goto 6
      3 xinbta = one-dexp((dlog((one-a)*qq)+beta)/qq)
        goto 6
      4 xinbta = dexp((dlog(a*pp)+beta)/pp)
        goto 6
      5 r = (y*y-three)/six
        s = one/(pp+pp-one)
        t = one/(qq+qq-one)
        h = two/(s+t)
        w = y*dsqrt(h+r)/h-(t-s)*(r+five/six-two/(three*h))
        xinbta = pp/(pp+qq*dexp(w+w))
c
c     solve for x by a modified newton-raphson method,
c     using the function betain
c
      6 r = one-pp
        t = one-qq
        yprev = zero
        sq = one
        prev = one
        if(xinbta.lt.0.0001d0) xinbta = 0.0001d0
        if(xinbta.gt.0.9999d0) xinbta = 0.9999d0
        IEX = MAX(-5.D0/PP**2 - 1.D0/A**.2 - 13.D0, SAE)
        ACU = 10.D0 ** IEX
      7 y = betain(xinbta,pp,qq,beta,ifault)
        if(ifault.eq.0) goto 8
        ifault = 3
        return
      8 continue
        xin = xinbta
        y = (y-a)*exp(beta+r*log(xin)+t*log(one-xin))
        if(y*yprev.le.zero) prev = max(sq, fpu)
        g = one
      9 adj = g*y
        sq = adj*adj
        if(sq.ge.prev) goto 10
        tx = xinbta-adj
        if(tx.ge.zero .and. tx.le.one) goto 11
     10 g = g/three
        goto 9
     11 if(prev.le.acu) goto 12
        if(y*y.le.acu) goto 12
        if(tx.eq.zero .or. tx.eq.one) goto 10
        if(tx.eq.xinbta) goto 12
        xinbta = tx
        yprev = y
        goto 7
     12 if (indx) xinbta = one-xinbta
        return
        end
```

```
      SUBROUTINE LPEST(N, P, X, Y, MAXIT, A, B, SD, R, RATE, IT, NPO,
     * IFAULT)
C
C FORMAL PARAMETERS
C    N     INTEGER        input : the number of points
C    P     REAL           input : p in the Lp norm
C    X     REAL ARRAY (N) input : the observed values x(i)
C    Y     REAL ARRAY (N) input : the observed values y(i)
C    MAXIT INTEGER        input : maximum allowable number of iterations
C    A     REAL           output : the estimate of alfa
C    B     REAL           output : the estimate of beta
C    SD    REAL           output : the Lp norm
C    R     REAL ARRAY (N) output : the signed residuals
C    RATE  REAL           output : abs(S(k+1)-S(k))/S(k+1), where S(k) is
C                                  the Lp norm of the fit on the kth iteration
C    IT    INTEGER        output : the number of iterations
C    NPO   INTEGER        output : the number of the points on the line
C   IFAULT INTEGER        output :
C                              IFAULT = 0 if the routine converged.
C                                     = 1 if return was due to an increase
C                                       in the norm.
C                                     = 2 if the maximum iterations
C                                       specified was less than 2.
C                                     = 3 if the weighted sample variance of
C                                       x is 0.
C                                     = 4 if N given less than 2.
C                                     = 5 if convergence has not be achieved
within
C                                       the maximum number of iterations
specified.
C
C  ALGORITHM AS 110 APPL. STATIST. (1977), VOL.26, NO.1
C
C  LP-NORM FIT OF STRAIGHT LINE BY EXTENSION OF SCHLOSSMACHER
C
      implicit INTEGER (h-n)
      implicit REAL (a-g)
      implicit REAL (o-z)
      REAL X(N), Y(N), R(N)
      DATA EPS /1.0E-6/
C
      IF(MAXIT .LT. 2) GOTO 9
      IF(N .LT. 2) GOTO 10
      IFAULT = 0
      WP = P - 2.0
      EPS2 = 2.0 * EPS
      SD = 0.0
      DO 1 I = 1,N
1     R(I) = 1.0
      DO 6 IT = 1, MAXIT
        NPO = 0
C
C          CALCULATE A AND B BY LEAST SQUARES ON WEIGHTED DATA,
C          USING THE HERRAMAN ALGORITHM.
C          OMIT OBSERVATIONS WITH SMALL RESIDUALS.
C
        SW = 0.0
        XMEAN = 0.0
        YMEAN = 0.0
        SSX = 0.0
        SPXY = 0.0
```

```
          DO 3 I = 1, N
            ABSRI = DABS(R(I))
            IF(ABSRI .LE. EPS) GOTO 2
            W = ABSRI ** WP
            SW = SW + W
            DIV = W / SW
            XI = X(I) - XMEAN
            YI = Y(I) - YMEAN
            XIW = XI * W
            DX = XI * XIW
            DXY = YI * XIW
            SSX = SSX + DX - DX * DIV
            SPXY = SPXY + DXY - DXY * DIV
            XMEAN = XMEAN + XI * DIV
            YMEAN = YMEAN + YI * DIV
            GOTO 3
2           NPO = NPO + 1
3         CONTINUE
          IF(SSX .LT. EPS) GOTO 11
          B = SPXY / SSX
          A = YMEAN - B * XMEAN
C
C            FORM RESIDUALS AND TEST CONVERGENCE
C
          SD2 = 0.0
          ISW = 0
          DO 4 I = 1, N
            RES = Y(I) - A - B * X(I)
            ABSRI = DABS(RES)
            IF(DABS(ABSRI - DABS(R(I))) .GT. EPS2) ISW = 1
            SD2 = SD2 + ABSRI ** P
            R(I) = RES
4         CONTINUE
          RATE = DABS(SD2 - SD) / SD2
          IF(ISW .EQ. 0) RETURN
          IF(IT .EQ. 1) GOTO 5
C
C           TEST FOR INCREASE IN NORM
C
          IF(SD2 .GT. SD) GOTO 7
5         SD = SD2
          A2 = A
          B2 = B
6       CONTINUE
C       FAILED TO CONVERGE IN MAXIT ITERATIONS
C
        IFAULT = 5
        RETURN
C
C        NORM INCREASED, RESTORE A, B, AND R, THEN RETURN
C
7       IFAULT = 1
        A = A2
        B = B2
        DO 8 I = 1, N
8       R(I) = Y(I) - A - B * X(I)
        RETURN
C
C        MAXIT SPECIFIED LESS THAN 2
C
9       IFAULT = 2
```

```
      RETURN
C
C      N LESS THAN 2
C
10     IFAULT = 4
       RETURN
C
C      VARIANCE OF WEIGHTED X IS ZERO
C
11    IFAULT = 3
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION PPND(P,IER)
C
C       ALGORITHM AS 111, APPL.STATIST., VOL.26, 118-121, 1977.
C
C       PRODUCES NORMAL DEVIATE CORRESPONDING TO LOWER TAIL AREA = P.
C
C       See also AS 241 which contains alternative routines accurate to
C       about 7 and 16 decimal digits.
C
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DATA SPLIT/0.42D0/
        DATA A0,A1,A2,A3/2.50662823884D0,-18.61500062529D0,
     1  41.39119773534D0,-25.44106049637D0/, B1,B2,B3,B4/
     2  -8.47351093090D0,23.08336743743D0,-21.06224101826D0,
     3  3.13082909833D0/, C0,C1,C2,C3/-2.78718931138D0,-2.29796479134D0,
     4  4.85014127135D0,2.32121276858D0/, D1,D2/3.54388924762D0,
     5  1.63706781897D0/
        DATA ZERO/0.D0/, ONE/1.D0/, HALF/0.5D0/
C
        IER = 0
        Q = P-HALF
        IF (ABS(Q).GT.SPLIT) GO TO 10
C
C       0.08 < P < 0.92
C
        R = Q*Q
        PPND = Q*(((A3*R + A2)*R + A1)*R + A0)/((((B4*R + B3)*R + B2)*R
     1          + B1)*R + ONE)
        RETURN
C
C       P < 0.08 OR P > 0.92, SET R = MIN(P,1-P)
C
   10   R = P
        IF (Q.GT.ZERO) R = ONE-P
        IF (R.LE.ZERO) GO TO 20
        R = SQRT(-LOG(R))
        PPND = (((C3*R + C2)*R + C1)*R + C0)/((D2*R + D1)*R + ONE)
        IF (Q.LT.ZERO) PPND = -PPND
        RETURN
   20   IER = 1
        PPND = ZERO
        RETURN
        END
```

```
      SUBROUTINE PANDQ(S, F, R, C, ROWSUM, COLSUM, N, IFAULT)
C
C     ALGORITHM AS 114  APPL. STATIST. (1977) VOL. 26, NO. 2
C
C     Finds the numerator S required by several measures of association.
C
      INTEGER S, R, C, F(R,C), ROWSUM(R), COLSUM(C), N, IFAULT
C
C     Local variables
C
      INTEGER P, Q, FI1, FIJ, I, J, NC, ND
C
      IFAULT = 1
      IF (R .LT. 2 .OR. C .LT. 2) RETURN
      IFAULT = 0
C
      ROWSUM(1) = 0
      DO 10 J = 1, C
        COLSUM(J) = F(1,J)
        ROWSUM(1) = ROWSUM(1) + F(1,J)
   10 CONTINUE
      NC = 0
      P = 0
      Q = 0
      DO 20 I = 2, R
        FI1 = F(I,1)
        ND = NC + ROWSUM(I-1) - COLSUM(1)
        Q = Q + FI1 * ND
        NC = COLSUM(1)
        COLSUM(1) = COLSUM(1) + FI1
        ROWSUM(I) = FI1
        DO 15 J = 2, C
          FIJ = F(I,J)
          ND = ND - COLSUM(J)
          Q = Q + FIJ * ND
          P = P + FIJ * NC
          NC = NC + COLSUM(J)
          COLSUM(J) = COLSUM(J) + FIJ
          ROWSUM(I) = ROWSUM(I) + FIJ
   15   CONTINUE
   20 CONTINUE
      N = NC + ROWSUM(R)
      S = P - Q
      RETURN
      END
```

```
      SUBROUTINE TETRA(A, B, C, D, R, SDR, SDZERO, ITYPE, IFAULT)
C
C        ALGORITHM  AS 116 APPL. STATIST. (1977) VOL.26, NO.3
C
C        TO COMPUTE THE TETRACHORIC CORRELATION (R) AND ITS STANDARD
C        ERRORS (SDR AND SDZERO) FROM THE FREQUENCIES OF A 2*2 TABLE
C        (A, B, C AND D)
C        X AND W ARE CONSTANTS USED IN GAUSSIAN QUADRATURE
C
      DIMENSION  X(16), W(16)
      DATA X(1) /0.9972638618/, X(2) /0.9856115115/, X(3) /0.9647622556/,
     1     X(4) /0.9349060759/, X(5) /0.8963211558/, X(6) /0.8493676137/,
     2     X(7) /0.7944837960/, X(8) /0.7321821187/, X(9) /0.6630442669/,
     3     X(10)/0.5877157572/, X(11)/0.5068999089/, X(12)/0.4213512761/,
     4     X(13)/0.3318686023/, X(14)/0.2392873623/, X(15)/0.1444719616/,
     5     X(16)/0.0483076657/
      DATA W(1) /0.0070186100/, W(2) /0.0162743947/, W(3) /0.0253920653/,
     1     W(4) /0.0342738629/, W(5) /0.0428358980/, W(6) /0.0509980593/,
     2     W(7) /0.0586840935/, W(8) /0.0658222228/, W(9) /0.0723457941/,
     3     W(10)/0.0781938958/, W(11)/0.0833119242/, W(12)/0.0876520930/,
     4     W(13)/0.0911738787/, W(14)/0.0938443991/, W(15)/0.0956387201/,
     5     W(16)/0.0965400885/
      DATA ZERO /0.0/, ONE/1.0/, TWO /2.0/, FOUR /4.0/, SIX /6.0/
      DATA HALF /0.5/, TWOPI/6.28318531/, SQT2PI /2.50662827/
      DATA RLIMIT /0.9999/, RCUT /0.95/, UPLIM /5.0/
      DATA CONST /1E-36/, CHALF /1E-18/
      DATA CONV /1E-8/, CITER /1E-6/, NITER /25/
C
C        INITIALIZATION
C
      R = ZERO
      SDZERO = ZERO
      SDR = ZERO
      ITYPE = 0
      IFAULT = 0
C
C       CHECK IF ANY CELL FREQUENCY IS NEGATIVE
C
      IF (A .LT. ZERO .OR. B .LT. ZERO .OR. C .LT. ZERO
     *   .OR. D .LT. ZERO) GOTO 92
C
C       CHECK IF ANY FREQUENCY IS ZERO AND SET KDELTA
C
      KDELTA = 1
      DELTA  = ZERO
      IF (A .EQ. ZERO .OR. D .EQ. ZERO) KDELTA  = 2
      IF (B .EQ. ZERO .OR. C .EQ. ZERO) KDELTA  = KDELTA + 2
C
C        KDELTA=4 MEANS TABLE HAS ZERO ROW OR COLUMN, RUN IS TERMINATED
C
      GOTO (4, 1, 2 , 92), KDELTA
C
C        DELTA IS 0.0, 0.5 OR -0.5 ACCORDING TO WHICH CELL IS ZERO
C
    1 DELTA  = HALF
      IF (A .EQ. ZERO .AND. D .EQ. ZERO) R = -ONE
      GOTO 4
    2 DELTA = -HALF
      IF (B .EQ. ZERO .AND. C .EQ. ZERO) R = ONE
    4 IF (R .NE. ZERO) ITYPE = 3
C
```

```
C        STORE FREQUENCIES IN  AA, BB, CC AND DD
C
      AA = A + DELTA
      BB = B - DELTA
      CC = C - DELTA
      DD = D + DELTA
      TOT = AA + BB + CC + DD
C
C        CHECK IF CORRELATION IS NEGATIVE, ZERO, POSITIVE
C
      IF (AA * DD - BB * CC) 7, 5, 6
    5 ITYPE = 4
C
C        COMPUTE PROBABILITIES OF QUADRANT AND OF MARGINALS
C        PROBAA AND PROBAC CHOSEN SO THAT CORRELATION IS POSITIVE.
C        KSIGN INDICATES WHETHER QUADRANTS HAVE BEEN SWITCHED
C
    6 PROBAA = AA / TOT
      PROBAC = (AA + CC) / TOT
      KSIGN = 1
      GOTO 8
    7 PROBAA = BB / TOT
      PROBAC = (BB + DD) / TOT
      KSIGN = 2
    8 PROBAB = (AA + BB) / TOT
C
C        COMPUTE NORMAL DEVIATES FOR THE MARGINAL FREQUENCIES
C        SINCE NO MARGINAL CAN BE ZERO, IE IS NOT CHECKED
C
      ZAC = PPND(PROBAC, IE)
      ZAB = PPND(PROBAB, IE)
      SS = EXP(-HALF * (ZAC ** 2 + ZAB ** 2)) / TWOPI
C
C        WHEN R IS 0.0, 1.0 OR -1.0, TRANSFER TO COMPUTE SDZERO
C
      IF (R .NE. ZERO .OR. ITYPE .GT. 0) GOTO 85
C
C        WHEN MARGINALS ARE EQUAL, COSINE EVALUATION IS USED
C
      IF (A .EQ. D .AND. B .EQ. C) GOTO 60
C
C        INITIAL ESTIMATE OF CORRELATION IS YULES Y
C
      RR = (SQRT(AA * DD) - SQRT(BB * CC)) ** 2 / ABS(AA * DD - BB * CC)
      ITER = 0
C
C        IF RR EXCEEDS RCUT, GAUSSIAN QUADRATURE IS USED
C
   10 IF (RR .GT. RCUT) GOTO 40
C
C        TETRACHORIC SERIES IS COMPUTED
C
C        INITIALIZATION
C
      VA = ONE
      VB = ZAC
      WA = ONE
      WB = ZAB
      TERM = ONE
      ITERM = 0
      SUM = PROBAB * PROBAC
```

```
      DERIV = ZERO
      SR = SS
   15 IF (ABS(SR) .GT. CONST) GOTO 20
C
C        RESCALE TERMS TO AVOID OVERFLOWS AND UNDERFLOWS
C
      SR = SR / CONST
      VA = VA * CHALF
      VB = VB * CHALF
      WA = WA * CHALF
      WB = WB * CHALF
C
C        FORM SUM AND DERIVATIVE OF SERIES
C
   20 DR = SR * VA * WA
      SR = SR * RR / TERM
      COF = SR * VA * WA
C
C        ITERM COUNTS NO. OF CONSECUTIVE TERMS .LT. CONV
C
      ITERM = ITERM + 1
      IF (ABS(COF) .GT. CONV) ITERM = 0
      SUM = SUM + COF
      DERIV = DERIV + DR
      VAA = VA
      WAA = WA
      VA = VB
      WA = WB
      VB = ZAC * VA - TERM * VAA
      WB = ZAB * WA - TERM * WAA
      TERM = TERM + ONE
      IF (ITERM .LT. 2 .OR. TERM .LT. SIX) GOTO 15
C
C        CHECK IF ITERATION CONVERGED
C
      IF (ABS(SUM-PROBAA) .GT. CITER) GOTO 25
C
C        ITERATION HAS CONVERGED, SET ITYPE
C
      ITYPE = TERM
      GOTO 70
C
C        CALCULATE NEXT ESTIMATE OF CORRELATION
C
   25 ITER = ITER + 1
C
C        IF TOO MANY ITERATlONS, RUN IS TERMINATED
C
      IF (ITER .GE. NITER) GOTO 93
      DELTA = (SUM - PROBAA) / DERIV
      RRPREV = RR
      RR = RR - DELTA
      IF (ITER .EQ. 1) RR = RR + HALF * DELTA
      IF (RR .GT. RLIMIT) RR = RLIMIT
      IF (RR .LT. ZERO) RR = ZERO
      GOTO 10
C
C       GAUSSIAN QUADRATURE
C
   40 IF (ITER .GT. 0) GOTO 41
C
```

```
C       INITIALIZATION, IF THIS IS FIRST ITERATION
C
      SUM = PROBAB * PROBAC
      RRPREV = ZERO
C
C        INITIALIZATION
C
   41 SUMPRV = PROBAB - SUM
      PROB = BB / TOT
      IF (KSIGN .EQ. 2) PROB = AA / TOT
      ITYPE = 1
C
C        LOOP TO FIND ESTIMATE OF CORRELATION
C        COMPUTATION OF INTEGRAL (SUM) BY QUADRATURE
C
   42 RRSQ = SQRT(ONE - RR ** 2)
      AMID = HALF * (UPLIM + ZAC)
      XLEN = UPLIM - AMID
      SUM = ZERO
      DO 44 IQUAD = 1, 16
      XLA = AMID + X(IQUAD) * XLEN
      XLB = AMID - X(IQUAD) * XLEN
C
C       TO AVOID UNDERFLOWS, TEMPA AND TEMPB ARE USED
C
      TEMPA = (ZAB - RR * XLA) / RRSQ
      IF (TEMP A .G E. -SIX) SUM = SUM + W(IQUAD) *
     *  EXP(-HALF  * XLA ** 2) * ALNORM(TEMPA, .FALSE.)
      TEMPB = (ZAB - RR * XLB) / RRSQ
      IF (TEMPB .GE. -SIX) SUM = SUM + W(IQUAD) *
     *  EXP(-HALF * XLB ** 2) * ALNORM (TEMPB, .FALSE.)
   44 CONTINUE
      SUM = SUM * XLEN / SQT2PI
C
C        CHECK IF ITERATION HAS CONVERGED
C
      IF (ABS(PROB - SUM) .LE. CITER) GOTO 70
      ITER = ITER + 1
C
C         IF TOO MANY ITERATIONS, RUN IS TERMINATED
C
      IF (ITER .GE. NITER) GO TO 93
C
C         ESTIMATE CORRELATION FOR NEXT ITERATION BY LINEAR INTERPOLATION
C
      RREST = ((PROB -  SUM) * RRPREV - (PROB - SUMPRV) * RR)
     *   / (SUMPRV - SUM)
C
C        IS ESTIMATE POSITIVE AND LESS THAN UPPER LIMIT
C
      IF (RREST .GT . RLIMIT) RREST = RLIMIT
      IF (RREST .LT. ZERO) RREST = ZERO
      RRPREV = RR
      RR = RREST
      SUMPRV = SUM
C
C        IF ESTIMATE HAS SAME VALUE ON TWO ITERATIONS, STOP ITERATION
C
      IF (RR .EQ. RRPREV) GOTO 70
      GOTO 42
C
```

```
C        WHEN ALL MARGINALS ARE EQUAL THE COSINE FUNCTION IS USED
C
   60 RR = -COS(TWOPI * PROBAA)
      ITYPE = 2
C
C        COMPUTE SDR
C
   70 R = RR
      RRSQ = SQRT(ONE - R ** 2)
      IF (KDELTA .GT. 1) ITYPE = -ITYPE
      IF (KSIGN .EQ. 1) GOTO 71
      R = -R
      ZAC = -ZAC
   71 PDF = EXP(-HALF * (ZAC ** 2 - TWO * R * ZAC * ZAB + ZAB ** 2)
     *   / RRSQ ** 2) / (TWOPI * RRSQ)
      PAC = ALNORM((ZAC - R * ZAB) / RRSQ, .FALSE.) - HALF
      PAB = ALNORM((ZAB - R * ZAC) / RRSQ, .FALSE.) - HALF
      SDR = (AA + DD) * (BB + CC) / FOUR + PAB ** 2 * (AA + CC)
     1 * (BB + DD) + PAC ** 2 * (AA + BB) * (CC + DD) + TWO * PAB *
     2  PAC * (AA * DD - BB * CC) - PAB * (AA * BB - CC * DD) -
     3  PAC * (AA * CC - BB * DD)
      IF (SDR .LT. ZERO) SDR = ZERO
      SDR = SQRT(SDR) / (TOT * PDF * SQRT(TOT))
C
C        COMPUTE SDZERO
C
   85 SDZERO = SQRT((AA + BB) * (AA + CC) * (BB + DD) * (CC + DD) / TOT)
     * / (TOT ** 2 * SS)
      IF (R .EQ. ZERO) SDR = SDZERO
      GOTO 99
C
C        ERROR TERMINATIONS
C
   92 IFAULT = 1
   93 IFAULT = IFAULT + 1
C
   99 RETURN
      END
```

```
      subroutine chrft(x, y, wr, wi, isize, lwork, itype, ifault)
c
c     Algorithm AS 117.1 Appl. Statist. (1977) vol.26, no.3
c
c     Code for complex discrete Fourier transform of a sequence of
c     general length by the Chirp-Z transform
c
      real    x(lwork), y(lwork), wr(lwork), wi(lwork)
      data    zero, one/0.0, 1.0/
c
c     Check that ISIZE and LWORK are valid
c
      ifault = 0
      if (isize .lt. 3) ifault = 1
      ii = 4
      do 1 k = 3, 20
        ii = ii * 2
        if (ii .eq. lwork) go to 3
        if (ii .gt. lwork) go to 2
    1 continue

    2 ifault = ifault + 2
    3 if (lwork .lt. 2 * isize) ifault = ifault + 4
      if (itype .eq. 0) ifault = ifault + 8
      if (ifault .ne. 0) return
c
c     Multiply by the Chirp function in the time domain
c
      call chrp(x, y, one, isize, lwork, itype)
      np1 = isize + 1
      do 4 nn = np1, lwork
        x(nn) = zero
        y(nn) = zero
    4 continue
c
c     Fourier transform the chirped series
c
      call fastf(x, y, lwork, 1)
c
c     Convolve by frequency domain multiplication with (wr, wi)
c
      do 5 nn = 1, lwork
        xwi = wi(nn)
        if (itype .lt. 0) xwi = -xwi
        z = x(nn) * wr(nn) - y(nn) * xwi
        y(nn) = x(nn) * xwi + y(nn) * wr(nn)
        x(nn) = z
    5 continue
c
c     Inverse Fourier transform
c
      call fastf(x, y, lwork, -1)
c
c     Multiply by Chirp function & gain correction
c
      gc = lwork
      if (itype .gt. 0) gc = gc/isize
      call chrp(x, y, gc, isize, lwork, itype)

      return
      end
```

```
      subroutine chfor(xray, wr, wi, jsize, lwork, mwork, ifault)
c
c     Algorithm AS 117.2 Appl. Statist. (1977) vol.26, no.3
c
c     Forward Chirp DFT for real even length input sequence
c
      real    xray(lwork), wr(mwork), wi(mwork)
      data    zero, quart, half, one, one5, two, four
     +        /0.0,  0.25,  0.5, 1.0,  1.5, 2.0,  4.0/
c
c     Check for valid JSIZE, LWORK & MWORK
c
      ifault = 0
      ii = 8
      do 1 k = 4, 21
        ii = ii * 2
        if (ii .eq. lwork) go to 3
        if (ii .gt. lwork) go to 2
    1 continue
    2 ifault = 2
    3 if (lwork .lt. 2 * jsize) ifault = ifault + 4
      if (lwork .ne. 2 * mwork) ifault = ifault + 16
      nn = jsize / 2
      if (2 * nn .ne. jsize) ifault = ifault + 32
      if (nn .lt. 3) ifault = ifault + 1
      if (ifault .ne. 0) return
      ll = mwork
c
c     Split XRAY into even and odd sequences of length n/2 placing
c     the even terms in the bottom half of XRAY as dummy real terms
c     and odd terms in the top half as dummy imaginary terms.
c
      do 4 i = 1, nn
        ii = 2 * i
        m = ll + i
        xray(m) = xray(ii)
        xray(i) = xray(ii-1)
    4 continue
c
c     Perform forward Chirp DFT on even and odd sequences.
c     IFA is not tested as possible faults have already been checked.
c
      call chrft(xray, xray(ll+1), wr, wi, nn, ll, 1, ifa)
c
c     Reorder the results according to output format.
c     First, the unique real terms.
c
      z = half * (xray(1) + xray(ll+1))
      xray(nn+1) = half * (xray(1) - xray(ll+1))
      xray(1) = z
c
c     If nn is even, terms at nn/2 + 1 and 3*nn/2 can be calculated
c
      nnn = nn / 2
      if (nn .ne. 2 * nnn) go to 5
      m = nnn + 1
      xray(m) = half * xray(m)
      mm = m + ll
```

```
          m = m + nn
          xray(m) = -half * xray(mm)
          go to 6
      5   nnn = nnn + 1
c
c         Set up trig functions for calculations of remaining
c         non-unique terms
c
      6   z = four * atan(one) / nn
          bcos = -two * (sin(z/two) ** 2)
          bsin = sin(z)
          un = one
          vn = zero
c
c         Calculate & place remaining terms in correct locations.
c
          do 7 i = 2, nnn
            z = un * bcos + un + vn * bsin
            vn = vn * bcos + vn - un * bsin
            save1 = one5 - half * (z * z + vn * vn)
            un = z * save1
            vn = vn * save1
            ii = nn + 2 - i
            m = ll + i
            mm = ll + ii
            an = quart * (xray(i) + xray(ii))
            bn = quart * (xray(m) - xray(mm))
            cn = quart * (xray(m) + xray(mm))
            dn = quart * (xray(i) - xray(ii))
            xray(i) = an + un * cn + vn * dn
            xray(ii) = two * an - xray(i)
            j = nn + i
            jj = nn + ii
            xray(j) = bn - un * dn + vn * cn
            xray(jj) = xray(j) - two * bn
      7   continue

          return
          end



          subroutine chrev(xray, wr, wi, jsize, lwork, mwork, ifault)
c
c         Algorithm AS 117.3 Appl. Statist. (1977) vol.26, no.3
c
c         Inverse Chirp DFT to give real even length sequence
c
          real    xray(lwork), wr(mwork), wi(mwork)
          data    zero, half, one, one5, two, four
     +           /0.0,  0.5, 1.0,  1.5, 2.0,  4.0/
c
c         Check for valid JSIZE, LWORK & MWORK
c
          ifault = 0
          ii = 8
          do 1 k = 4, 21
            ii = ii * 2
            if (ii .eq. lwork) go to 3
            if (ii .gt. lwork) go to 2
      1   continue
```

```
      2   ifault = 2
      3   if (lwork .lt. 2 * jsize) ifault = ifault + 4
          if (lwork .ne. 2 * mwork) ifault = ifault + 16
          nn = jsize / 2
          if (2 * nn .ne. jsize) ifault = ifault + 32
          if (nn .lt. 3) ifault = ifault + 1
          if (ifault .ne. 0) return
          ll = mwork
c
c         Reorder the spectrum; first the unique terms.
c
          z = xray(1) + xray(nn+1)
          xray(ll+1) = xray(1) - xray(nn+1)
          xray(1) = z
c
c         If nn is even then terms nn/2 + 1 and 3*nn/2 + 1 must be
c         reordered.
c
          nnn = nn / 2
          if (nn .ne. 2 * nnn) go to 4
          m = nnn + 1
          xray(m) = two * xray(m)
          mm = m + ll
          m = m + nn
          xray(mm) = -two * xray(m)
          go to 5
      4   nnn = nnn + 1
c
c         Set up trig functions for manipulation of remaining terms.
c
      5   z = four * atan(one) / nn
          bcos = -two * (sin(z/two) ** 2)
          bsin = sin(z)
          un = one
          vn = zero
c
c         Perform manipulation and reordering of remaining non-unique
c         terms.
c
          do 6 i = 2, nnn
            z = un * bcos + un + vn * bsin
            vn = vn * bcos + vn - un * bsin
            save1 = one5 - half * (z * z + vn * vn)
            un = z * save1
            vn = vn * save1
            ii = nn + 2 - i
            j = nn + i
            jj = nn + ii
            an = xray(i) + xray(ii)
            bn = xray(j) - xray(jj)
            pn = xray(i) - xray(ii)
            qn = xray(j) + xray(jj)
            cn = un * pn + vn * qn
            dn = vn * pn - un * qn
            xray(i) = an + dn
            xray(ii) = an - dn
            m = ll + i
            mm = ll + ii
            xray(m) = cn + bn
            xray(mm) = cn - bn
      6   continue
```

```
c
c       Do inverse Chirp DFT to give even and odd sequences.
c       IFA is not tested as possible faults have already been checked
c
        call chrft(xray, xray(ll+1), wr, wi, nn, ll, -1, ifa)
c
c       Interlace the results to produce the required output sequence.
c
        nnn = nn + 1
        ii = jsize + 2
        do 7 i = 1, nn
          j = nnn - i
          jj = ll + j
          m = ii - 2 * i
          mm = m - 1
          xray(m) = xray(jj)
          xray(mm) = xray(j)
    7   continue

        return
        end



      subroutine setwt(wr, wi, ksize, kwork, ifault)
c
c       Algorithm AS 117.4 Appl. Statist. (1977) vol.26, no.3
c
c       Subroutine to set up Fourier transformed Chirp function for
c       use by subroutine CHRFT.
c
      real    wr(kwork), wi(kwork)
      data    zero, one, four /0.0, 1.0, 4.0/
c
c       Check that KSIZE & KWORK are valid
c
      ifault = 0
      if (ksize .lt. 3) ifault = 1
      ii = 4
      do 1 k = 3, 20
        ii = ii * 2
        if (ii .eq. kwork) go to 3
        if (ii .gt. kwork) go to 2
    1   continue
    2   ifault = 2
    3   if (kwork .lt. 2 * ksize) ifault = ifault + 4
      if (ifault .ne. 0) return
      tc = four * atan(one) / ksize
c
c       Set up bottom segment of Chirp function
c
      do 4 nn = 1, ksize
        z = nn - 1
        z = z * z * tc
        wr(nn) = cos(z)
        wi(nn) = sin(z)
    4 continue
c
c       Clear the rest
c
      do 5 nn = ksize+1, kwork
```

```
          wr(nn) = zero
          wi(nn) = zero
      5 continue
c
c       Copy to the top segment
c
        do 6 nn = kwork-ksize+2, kwork
          ll = kwork - nn + 2
          wr(nn) = wr(ll)
          wi(nn) = wi(ll)
      6 continue
c
c       Fourier transform the Chirp function
c
        call fastf(wr, wi, kwork, 1)

        return
        end


        subroutine chrp(x, y, gc, isize, lwork, itype)
c
c       Algorithm AS 117.5 Appl. Statist. (1977) vol.26, no.3
c
c       Subroutine to multiply time series by Chirp function
c
        dimension x(lwork), y(lwork)
        data    one, four /1.0, 4.0/

        tc = four * atan(one) / isize
        do 3 nn = 1, isize
          z = nn - 1
          z = z * z * tc
          xwr = cos(z)
          xwi = -sin(z)
          if (itype .lt. 0) xwi = -xwi
          z = x(nn) * xwr - y(nn) * xwi
          y(nn) = (x(nn) * xwi + y(nn) * xwr) * gc
          x(nn) = z * gc
      3 continue

        return
        end
```

```
      double precision function trigam(x, ifault)
      implicit double precision (a-h,o-z)
c
c        algorithm as121   Appl. Statist. (1978) vol 27, no. 1
c
c        calculates trigamma(x) = d**2(log(gamma(x))) / dx**2
c
      double precision a, b, one, half, b2, b4, b6,b8, x, y, z, zero
      data a, b, one, half /1.0d-4, 5.0d0, 1.0d0, 0.5d0/
      data zero /0.0d0/
c
c        b2, b4, b6 and b8 are Bernoulli numbers
c
      data b2, b4, b6,b8
     */0.1666666667d0, -0.03333333333d0, 0.02380952381, -0.03333333333/
c
c        check for positive value of x
c
      trigam = zero
      ifault = 1
      if (x.le.zero) return
      ifault = 0
      z = x
c
c        use small value approximation if x .le. a
c
      if (z .gt. a) goto 10
      trigam = one / (z * z)
      return
c
c        increase argument to (x+i) .ge. b
c
   10 if (z .ge. b) goto 20
      trigam = trigam + one / (z * z)
      z = z + one
      goto 10
c
c        apply asymptotic formula if argument .ge. b
c
   20 y = one / (z * z)
      trigam = trigam + half * y +
     * (one + y * (b2 + y * (b4 + y * (b6 + y * b8)))) / z
      return
      end
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      FUNCTION BMIX(X, N, L, Q, ITYPE, IFAULT)
C
C        Algorithm AS 123 (Applied Statistics, 27 (1978), pp. 104-109);
C
C        CALCULATES C.D.F.S OF MIXTURES OF BETA DISTRIBUTIONS.
C
      DIMENSION Q(101)
      LOGICAL SEC, ALT
      PARAMETER (C1=0.636619772368, C2=76.7625)
C
C        C1 = 2.0/PI.  C2 MUST NOT EXCEED LOG (BASE 10) OF LARGEST
C        POSSIBLE FLOATING POINT NUMBER.
C
      BMIX = 0.0
C
C        CHECK FOR ERRORS AND INITIALISE.
C
      IFAULT = 1
      I = L + ITYPE
      IF (N .LE. 0 .OR. N .GE. 101 .OR. I .LE. 1) RETURN
      IFAULT = 2
      XF1 = 1.0 - X
      IF (X .LT. 0.0 .OR. XF1 .LE. 0.0) RETURN
      RX = SQRT(X)
      RXF1 = SQRT(XF1)
      IFAULT = 3
      IF (ITYPE .EQ. 2) GO TO 40
      IF (ITYPE .NE. 1) RETURN
      M = L - 2
      D = M
      E = 1.0
      F = L
      C = (F - 1.0)*XF1
      Z = X
      GO TO 50
   40 I = L + N
      M = I - 3
      D = I
      E = -1.0
      F = I - 2
      C = F
      Z = X/XF1
   50 IFAULT = 4
      NN = N
      ALT = .FALSE.
      K = -2
      J = N + 1
C
C        CHECK FOR POSSIBILITY OF OVERFLOW.
C
   60 A = -F*ALOG10(RXF1)
      IF (A .LT. C2) GO TO 90
      IF (ITYPE .EQ. 1 .OR. ALT) RETURN
      IF (I .LT. 101) GO TO 70
      IFAULT = 5
      RETURN
C
C        IF ITYPE = 2 TRY ALTERNATIVE APPROACH.
C
```

```
   70 ALT = .TRUE.
      NN = I
      Z = XF1/X
      XF1 = X
      A = RX
      RX = RXF1
      RXF1 = A
      IF (L .EQ. 0) GO TO 85
      J = N + 2
      K = I + 1
      DO 80 I = J, K
   80 Q(I) = 0.0
   85 J = 1
      K = 2
      GO TO 60
C
C        MAIN SECTION OF ALGORITHM BEGINS HERE.
C
   90 IFAULT = 0
      SEC = .FALSE.
      I = NN
  100 A = 0.0
      B = 0.0
C
C        INNER LOOP.
C
  110 B = B + Q(J)
      IF (I .LE. 1) GO TO 120
      FI = I
      A = A*Z*(D/FI + E) - B
      I = I - 2
      J = J + K
      GO TO 110
  120 IF (.NOT. ALT) GO TO 130
      BMIX = BMIX + B
      A = -A
      B = -B
  130 IF (I .EQ. 1) GO TO 140
C
C        COMPLETION OF EVEN CALCULATIONS.
C
      BMIX = BMIX + B + A*RXF1**F
      GO TO 180
  140 IF (M .GE. 0) A = B + C*A
      I = M
C
C        OUTER LOOP.
C
  150 IF (I .LE. 1) GO TO 160
      FI = I
      A = (A - A/FI)*XF1 + B
      I = I - 2
      GO TO 150
C
C        COMPLETION OF ODD CALCULATIONS.
C
  160 A = A*RX
      IF (I .EQ. 0 .OR. I .EQ. -2) GO TO 170
      BMIX = BMIX + C1*(ATAN(RX/RXF1)*B + RXF1*A)
      GO TO 180
  170 BMIX = BMIX + A
```

```
  180 IF (SEC) RETURN
      SEC = .TRUE.
      I = NN - 1
      J = N
      IF (ALT) J = 2
      GO TO 100
      END
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE CVSUR1(N, M, L, MM, U, A, Z, IROWS, ITERS, F, THETA,
     *  VAR, CRI1, CRI2, F0, DF1, DF2, DDF1, DDF2, IFAULT)
C
C       ALGORITHM AS 125 APPL. STATIST. (1978) VOL.27, NO.2
C
C       THIS SUBROUTINE USES A NEWTON-RAPHSON ITERATION TO DETERMINE
C       THE MAXIMUM LIKELIHOOD ESTIMATES FOR AN EXPONENTIAL COVARIATE
C       SURVIVAL MODEL.
C
C  as r38 vol 30 no 3 1981 p 355 -- code changes included
C
C       Auxiliary routines required: CHOL = AS6, SYMINV = AS7
C
      DIMENSION U(N), A(N), Z(IROWS, M), THETA(L), DF1(L), DF2(L),
     *  VAR(MM), DDF1(MM), DDF2(MM)
      PARAMETER (ZERO=0.0, ONE=1.0)
C
      IFAULT = 3
      IF (N .LT. 1) GOTO 160
      IFAULT = 4
      IF (M .LT. 1) GOTO 160
C
C       FLAST PRERESENTS THE FALUE OF THE LOG LIKELIHOOD FUNCTION
C       FOR THE PREVIOUS ITERATION.
C
      FLAST = ZERO
      R = ZERO
      W = ZERO
      DO 10 I = 1, N
        IFAULT = 5
        AI = A(I)
        IF (AI .NE. ZERO .AND. AI .NE. ONE) GOTO 160
C
C       R IS THE NUMBER OF UNCENSORED OBSERVATIONS
C       W IS THE TOTAL OBSERVED TIME
C
        R = R + AI
        IFAULT = 6
        IF (U(I) .LT. ZERO) GOTO 160
        W = W + U(I)
10    CONTINUE
      IFAULT = 7
      IF (R .EQ. ZERO) GOTO 160
      IFAULT = 8
      IF (W .EQ. ZERO) GOTO 160
      IFAULT = 9
      IF (R .LT. FLOAT(L)) GOTO 160
C
C       THETA(L) CONTAINS THE ESTIMATE OF ALPHA,
C       INITIALIZE TO ITS MAXIMUM LIKELIHOOD ESTIMATE
C       VALUE FOR NONCOVARIATE CASE.
C
      THETA(L) = ALOG(R / W)
C
C       COMPUTE F0, THE MAXIMIZED LOG LIKELIHOOD WHEN
C       ALL BETAS ARE ZERO.
C
      F0 = -R + R * THETA(L)
C
```

```
C         INITIALIZE AND COMPUTE ZSUM AND STORE IN FIRST
C         M ELEMENTS OF ARRAY VAR.
C
        DO 30 J = 1, M
          THETA(J) = ZERO
          VARJ = ZERO
          DO 20 I = 1, N
            VARJ = VARJ + Z(I, J) * A(I)
20        CONTINUE
          VAR(J) = VARJ
30      CONTINUE
C
C         BEGIN ITERATING
C
        DO 60 K = 1, ITERS
          CALL CVSUR2(N, M, L, MM, U, A, R, Z, IROWS, VAR, THETA, F,
     *    DF1, DDF1)
C
C         DETERMINE WHETHER THE RELATIVE CHANGE IN LOG LIKELIHOODS IS
C         LESS THAN CRI1 OR NOT AND SET ICODE1 = 0, 1 ACCORDINGLY.
C
          ICODE1 = 1
          DIFF = F - FLAST
          FCRI1 = FLAST * CRI1
          IF (DIFF .LE. -FCRI1 .AND. DIFF .GE. ZERO) ICODE1 = 0
          FLAST = F
C
C         FIND THE CHOLESKY DECOMPOSITION OF DDF1 AND RETURN
C         RESULTS IN DDF2.
C
          CALL CHOL(DDF1, L, DDF2, NULLTY, IFAULT)
          IF (IFAULT .NE. 0) GOTO 160
C
C         SOLVE THE LINEAR SYSTEM DDF1 * X = DF1.
C         X IS RETURNED IN DF2 AND REPRESENTS THE INCREMENT
C         IN THE MLE ESTIMATES.
C
          CALL SOLVE(DDF2, DF2, DF1, L, MM, IER)
          IFAULT = 2
          IF (IER .EQ. 1) GOTO 160
C
C         IF ICODE1 = 1, F HAS NOT CONVERGED.
C
          IF (ICODE1 .EQ. 1) GOTO 45
C
C         SET ICODE2 = 1 IF THE RELATIVE CHANGE IN ANY THETA
C         IS GREATER THAN OR EQUAL TO CRI2 AND 0 OTHERWISE.
C
          ICODE2 = 0
          DO 40 J = 1, L
            IF (ABS(DF2(J)) .GE. ABS(THETA(J)) * CRI2) ICODE2 = 1
40        CONTINUE
C
C         FOR CONVERGENCE TO BE REACHED, BOTH THE RELATIVE CHANGE IN
C         SUCCESSIVE MLES MUST BE LESS THAN CRI2 AND THE RELATIVE
C         CHANGE IN SUCCESSIVE LOG LIKELIHOODS MUST BE LESS THAN CRI1.
C
          IF (ICODE1 .EQ. 0 .AND. ICODE2 .EQ. 0) GOTO 70
C
C         CONVERGENCE WAS NOT REACHED.
C         FIND NEW VALUES OF THETA AND BEGIN NEXT ITERATION.
```

```
C
45        DO 50 J = 1, L
            THETA(J) = THETA(J) + DF2(J)
50        CONTINUE
60      CONTINUE
C
C         CONVERGENCE WAS NOT REACHED IN ITERS ITERATIONS
C
        IFAULT = 1
        GOTO 160
70      ITERS = K
        IFAULT = 0
C
C         CONVERGENCE WAS REACHED IN K ITERATIONS.
C
        CALL CVSUR2(N, M, L, MM, U, A, R, Z, IROWS, VAR,
     *  THETA, F, DF2, DDF2)
        CALL SYMINV(DDF2, L, VAR, U, NULLTY, IFAULT)
160     RETURN
        END
C
C
        SUBROUTINE CVSUR2(N,M,L,MM,U,WORK,R,Z,IROWS,ZSUM,THETA,
     *  F, DF, DDF)
C
C         ALGORITHM AS 125.1 APPL. STATIST. (1978) VOL.27, NO.2
C
C       THIS SUBROUTINE CALCULATES THE LOG LIKELIHOOD (F), FIRST
C       DERIVATIVE ARRAY (DF), AND (-1)*SECOND DERIVATIVE MATRIX (DDF).
C       THE ELEMENTS OF DDF ARE IN SYMMETRIC STORAGE MODE.
C
        DIMENSION U(N), Z(IROWS, M), ZSUM(M), THETA(L), DF(L), DDF(MM),
     *  WORK(N)
        PARAMETER (ZERO=0.0)
C
        ETHETA = EXP(THETA(L))
        UEPROD = ZERO
        ZPROD = ZERO
        DO 20 J = 1, M
          ZPROD = ZPROD + THETA(J) * ZSUM(J)
20      CONTINUE
        DO 40 I = 1, N
          WORK(I) = ZERO
C
C         CALCULATE LOG LIKELIHOOD
C         THETA(1) - THETA(M) CONTAIN BETA ESTIMATES
C         THETA(M+1) CONTAINS THE ESTIMATE OF ALPHA.
C
          DO 30 J = 1, M
            WORK(I) = WORK(I) + THETA(J) * Z(I, J)
30        CONTINUE
          WORK(I) = EXP(WORK(I))
          UEPROD = UEPROD + U(I) * WORK(I)
40      CONTINUE
        F = R * THETA(L) + ZPROD - ETHETA * UEPROD
C
C         CALCULATE FIRST DERIVATIVE ARRAY DF.
C         FROM HERE THROUGH STATEMENT 60, DDF IS USED AS A WORK ARRAY
C
        TERM1 = -ETHETA * UEPROD
        DF(L) = R + TERM1
```

```
          DO 60 J = 1, M
            DDF(J) = ZERO
            DO 50 I = 1, N
              DDF(J) = DDF(J) + U(I) * Z(I, J) * WORK(I)
50          CONTINUE
            DDF(J) = -ETHETA * DDF(J)
            DF(J) = ZSUM(J) + DDF(J)
60        CONTINUE
C
C           CALCULATE (-1)*SECOND DERIVATIVE ARRAY DDF
C
          KK = (L - 1) * L / 2
          DO 70 J = 1, M
            KK = KK + 1
            DDF(KK) = -DDF(J)
70        CONTINUE
          DDF(MM) = -TERM1
          KK = 0
          DO 85 J = 1, M
            DO 80 K = 1, J
              KK = KK + 1
              DDF(KK) = ZERO
              DO 75 I = 1, N
                DDF(KK) = DDF(KK) + U(I) * Z(I, J) * Z(I, K) * WORK(I)
75            CONTINUE
              DDF(KK) = ETHETA * DDF(KK)
80          CONTINUE
85        CONTINUE
          RETURN
          END
C
C
          SUBROUTINE SOLVE(A, X, B, N, NN, IFAULT)
C
C           ALGORITHM AS 125.2 APPL. STATIST. (1978) VOL.27, NO.2
C
C           THIS SOLVES THE LINEAR SYSTEM L*(L-TRANSPOSE)*X=B,
C           WHERE L IS A LOWER TRIANGULAR MATRIX WITH POSITIVE
C           DIAGONALS.  THE ELEMENTS OF L ARE STORED IN ARRAY A
C           IN THE FOLLOWING ORDER - L(1,1), L(2,1), L(2,2),
C           L(3,1), L(3.2),.....
C           X AND B COULD OCCUPY THE SAME LOCATIONS, BUT THIS IS
C           PROHIBITED BY THE FORTRAN STANDARD (SECTION 8.4.2)
C
          PARAMETER (ZERO=0.0)
          DIMENSION A(NN), X(N), B(N)
C
C           SOLVE LW = B FOR W AND PUT RESULT IN X.
C
          IF (A(1) .LE. ZERO) GOTO 50
          X(1) = B(1) / A(1)
          IF (N .EQ. 1) GOTO 25
          DO 20 J = 2, N
            J1 = (J -1) * J / 2
            SUM = B(J)
            JM1 = J - 1
            DO 10 I = 1, JM1
              I1 = J1 + I
              SUM = SUM - A(I1) * X(I)
10          CONTINUE
            I1 = J1 + J
```

```
            IF (A(I1) .LE. ZERO) GOTO 50
            X(J) = SUM / A(I1)
   20     CONTINUE
   C
   C        SOLVE (L-TRANSPOSE)*W = X AND PUT RESULT IN X
   C
   25     X(N) = X(N) / A(NN)
          IF (N .EQ. 1) GOTO 45
          DO 40 JJ = 2, N
            J = N - JJ + 1
            SUM = X(J)
            JP1 = J + 1
            DO 30 I = JP1, N
              I1 = (I -1) * I / 2 + J
              SUM = SUM - A(I1) * X(I)
   30       CONTINUE
            I1 = J * (J + 1) / 2
            X(J) = SUM / A(I1)
   40     CONTINUE
   45     IFAULT = 0
          RETURN
   50     IFAULT = 1
          RETURN
          END
```

```
      double precision function rngpi(t, n, ifault)
c
c        Algorithm AS 126  Appl. Statist. (1978) Vol. 27, No. 2
c
c        Computes the probability of the normal range given t, the
c        upper limit of integration, and n, the sample size.
c
c     Auxiliary function required: ALNORM = algorithm AS66
c
      implicit double precision (a-h,o-z)
      dimension g(8), h(8)
c
      data g(1), g(2), g(3), g(4), g(5), g(6), g(7), g(8)
     */0.4947004675d0, 0.4722875115d0, 0.4328156012d0, 0.3777022042d0,
     * 0.3089381222d0, 0.2290083888d0, 0.1408017754d0, 0.04750625492d0/
c
      data h(1), h(2), h(3), h(4), h(5), h(6), h(7), h(8)
     */0.01357622971d0, 0.03112676197d0, 0.04757925584d0,
     * 0.06231448563d0,
     * 0.07479799441d0, 0.08457825969d0, 0.09130170752d0,
     * 0.09472530523d0/
c
      data zero, half, two, eight /0.0d0, 0.5d0, 2.0d0, 8.0d0/
c
      risf(x) = 0.3989422804d0 * exp(-half*x*x) * (alnorm(x, .false.)
     *  - alnorm(x-t, .false.)) ** (n-1)
c
      ifault = 0
      rngpi = zero
      if (t .le. zero .or. n .le. 1) return

      ifault = 0
      xl = half * t
      a = half * (eight + xl)
      b = eight - xl
      y = zero
      do 10 i = 1, 8
      c = b * g(i)
      y = y + h(i) * (risf(a + c) + risf(a - c))
   10 continue
      rngpi = (two * (alnorm(xl, .false.) - half)) ** n +
     *  two * b * y * n
      return
      end
```

```
      SUBROUTINE RNORTM(N, NP1, IBCONF, NB, B, NDBI, DBI, ISEED, A,
     *  CHISQ, FAB, U, W, IFAULT)
C
C       ALGORITHM AS 127 APPL. STATIST. (1978) VOL.27, NO.2
C
C       RNORTM GENERATES ORTHOGONAL MATRICES  A  FROM A DISTRIBUTION
C       WITH DENSITY FUNCTION DEFINED ON THE GROUP OF ORTHOGONAL
C       MATRICES.  WHEN THE INPUT PARAMETER IBCONF = 1 THE
C       MATRICES ARE GENERATED FROM THE INVARIANT HAAR MEASURE.
C
C     Corrections in remark AS R42 (Appl. Statist., vol.31, 1982) have
C     been incorporated.
C
      REAL B(NB), DBI(NDBI), A(N, N), CHISQ(N), U(NP1), W(N), FAB, UL,
     *  WL, WIL, WWIL, T2, HV, HW, ZERO, ONE
C
      DATA ZERO /0.0/, ONE /1.0/
C
C       STATEMENT FUNCTIONS FOR DOUBLE PRECISION
C       DOUBLE PRECISION SQRT, SIGN, ABS
C       SQRT(UL) = DSQRT(UL)
C       SIGN(UL, WL) = DSIGN(UL, WL)
C       ABS(UL) = DABS(UL)
C
C       SECTION 1   INITIALIZATION
C       CHECK CONSISTENCY OF INPUT DIMENSIONS
C
      IFAULT = 0
      IF (N .LT. 1) GOTO 1
      IF (NP1 .NE. N + 1) GOTO 1
      IF (IBCONF .LT. 1 .OR. IBCONF .GT. 3) GOTO 1
      IF (IBCONF .EQ. 1 .AND. (NB .NE. 1 .OR. NDBI .NE. 1)) GOTO 1
      IF (IBCONF .EQ. 2 .AND. (NB .NE. N * (N + 1) / 2 .OR. NDBI .NE. N))
     *  GOTO 1
      IF (IBCONF .EQ. 3 .AND. (NB .NE. N * (N + 1) * (2 * N + 1) / 6
     *  .OR. NDBI .NE. N)) GOTO 1
      GOTO 4
C
C       ERROR RETURNS
C
1     IFAULT = 1
      RETURN
2     IFAULT = 2
      RETURN
C
C       INITIALIZE DENSITY, H(STORED IN A), AND IBSUB
C
4     FAB = ONE
      DO 100 I = 1, N
        DO 50 J = 1, N
50      A(I, J) = ZERO
        A(I, I) = ONE
100   CONTINUE
      IBSUB = 0
C
C       CONSTRUCT A ONE COLUMN AT A TIME
C
      DO 315 L = 1, N
        NP1L = NP1 - L
C
C       SECTION 2
```

```
C         CONSTRUCT U(L...N), UNIFORMLY DISTRIBUTED ON THE SURFACE OF THE
C         (N+1-L)-SPHERE OF RADIUS UL = SQRT(CHISQ(L)).
C
          CALL NORMAL(U(L), NP1L + 1, ISEED, CHISQ(L))
          UL = SQRT(CHISQ(L))
          IF (UL .EQ. ZERO) GOTO 2
          IF (L .EQ. N) GOTO 301
C
C         SECTION 3
C         CALCULATE  W = B*U  AND STORE ITS LENGTH IN WL.
C         SIGN OF WL PREVENTS LOSS OF SIGNIFICANCE IN WWIL
C
          GOTO (215, 220, 225), IBCONF
C
C         B IS IDENTITY
C
215       WL = SIGN(UL, U(L))
          DO 219 J = L, N
219       W(J) = U(J)
          GOTO 229
C
C         DIAGONAL MATRICES B
C
220       WL = ZERO
          DO 221 J = L, N
            ITEMP = IBSUB + J
            T2 = B(ITEMP) * U(J)
            WL = WL + T2 * T2
            W(J) = T2
221       CONTINUE
          IBSUB = IBSUB + N - L
          GOTO 228
C
C         FULL MATRICES B
C
225       WL = ZERO
          DO 227 J = L, N
            T2 = ZERO
            DO 226 K = L, N
              ITEMP = IBSUB + K
              T2 = T2 + B(ITEMP) * U(K)
226         CONTINUE
            IBSUB = IBSUB + NP1L
            WL = WL + T2 * T2
            W(J) = T2
227       CONTINUE
          IBSUB = IBSUB - 1
C
C         CALCULATE DENSITY FOR IBCONF = 2 OR 3
C
228       WL = SIGN(SQRT(WL), -W(L))
          FAB = (WL / UL) ** NP1L * DBI(L) * FAB
          IF (FAB .EQ. ZERO) GOTO 2
229       CONTINUE
C
C         SECTION 4
C         PROJECT W ONTO ORTHOGONAL COMPLEMENT OF FIRST L-1
C         COLUMNS OF A, NORMALIZE, AND STORE IN L COLUMN OF A.
C         A(L) = H(L) * W * WIL
C         CALCULATE PROJECTION MATRIX H(L) FOR ORTHOGONAL
C         COMPLEMENT OF FIRST L COLUMNS OF A AND STORE IN LAST
```

```
C         N-L COLUMNS OF A.
C
          LP1 = L + 1
          WIL = ONE / WL
          WWIL = ONE / (WL - W(L))
          DO 250 I = 1, N
            T2 = ZERO
            DO 230 J = L, N
230         T2 = T2 + A(I, J) * W(J)
            HV = T2 * WIL
            HW = (HV - A(I, L)) * WWIL
            A(I, L) = HV
            DO 240 J = LP1, N
240         A(I, J) = A(I, J) - HW * W(J)
250       CONTINUE
C
C         END OF L COLUMN OF MATRIX A
C
C         SECTION 5
C         WHEN L = N ONLY SIGN NEEDS TO BE CHOSEN
C         REMEMBER MATRIX H(N) IS STORED IN N COLUMN OF A.
C
301     IF (U(L) .GT. ZERO) GOTO 315
        DO 310 I = 1, N
310     A(I, L) = -A(I, L)
315  CONTINUE
C
C         FAB = ABS(B(LAST) * DBI(N) * FAB)
C
320     FAB = ABS(FAB)
C
        RETURN
        END
```

```
      subroutine covmat(v,n,mdim,v11,ex1,ex2,summ2,ifault)
c
c      Appl. Statist. algorithm as 128 (1978), vol. 27
c      Davis C.S. and Stephens M.A.
c
c      Computes and normalises the David-Johnson approximation
c      for the covariance matrix of normal order statistics.
c
c      Auxiliary function equired: PPND = algorithm AS111
c      N.B. This file also includes ASR72 to calculate v11
c
      integer n,mdim,ifault
      double precision v(mdim,n),v11,ex1,ex2,summ2
c
c      local integer variables
c
      integer i,j,k,ni,nj
c
c      local real variables
c
      double precision cnst,dxr,d2xr,d3xr,d4xr,d5xr,dxs,d2xs,
     1       d3xs,d4xs,d5xs,pr,ps,qr,rn,rn1,rn2,rn22,rn23,sum,
     2       two,xr,xs,zero
c
      common /cons/ rn2,rn22,rn23
c
c     initialise constants
c
      data zero /0.0d0/,half /0.5d0/, one /1.0d0/ ,two/2.0d0/
c
      ifault = 1
      if(n .gt. mdim .or. n. lt. 2) return
      ifault = 0
      rn=n
      rn1=rn+one
      rn2=rn+two
      rn22=rn2*rn2
      rn23=rn22*rn2
      nhalf1=(n+1) / 2
c
c     the elements of the upper triangle
c     are first computed
c
      ni=n
      do 50 i=1,nhalf1
         pr=float(i)/rn1
         qr=one-pr
c
c        in function ppnd, xr is computed to satisfy
c        prob(z.lt.pr) = pr, where z is n(0,1)
c
         xr = ppnd(pr, ifault)
         call der(xr,dxr,d2xr,d3xr,d4xr,d5xr)
         do 40 j=i,ni
            if (i .ne. j) goto 30
c
c     if i is equal to j, var(xr) is calculated
c
            v(i,j)=var(dxr,d2xr,d3xr,d4xr,d5xr,pr,qr)
            go to 40
c
```

```
c     if i is not equal to j, cov(xr,xs) is calculated
c
   30       ps=float(j)/rn1
            xs=ppnd(ps,ifault)
            call der(xs,dxs,d2xs,d3xs,d4xs,d5xs)
            v(i,j)=cov(dxr,d2xr,d3xr,d4xr,d5xr,pr,qr,
     1                  dxs,d2xs,d3xs,d4xs,d5xs,ps)
            v(j,i)=v(j,i)
   40    continue
         ni=ni-1
   50 continue
c
c        By symmetry the other elements of v will now be filled
c
      nj = n
      do 70 i = 2,n
      njm1 = nj - 1
      im1 = i - 1
      do 60 j = nj, n
      v(i,j) = v(im1, njm1)
      im1 = im1 - 1
   60 continue
      nj = nj - 1
   70 continue
c
c     insert exact values of v(1,1) and v(1,2)
c
      v(1,1)=v11
      v(n,n)=v(1,1)
      v(1,2)=v(1,1)+ex1*(ex1-ex2)-one
      v(2,1)=v(1,2)
      nsub1=n-1
      v(n,nsub1)=v(1,2)
      v(nsub1,n)=v(1,2)
c
c     normalise the first row of v, leaving
c     v(1,1) and v(1,2) fixed
c
      if(n.eq.2) return
      sum=zero
      do 80 j=3,n
         sum=sum+v(1,j)
   80 continue
      cnst=(one-v(1,1)-v(1,2))/sum
      nj=n-2
      do 90 j=3,n
         v(1,j)=v(1,j)*cnst
         v(j,1)=v(1,j)
         v(n,nj)=v(1,j)
         v(nj,n)=v(1,j)
         nj=nj-1
   90 continue
c
c     normalise rows 2 through n-1 of v
c
      call rwnorm(v,n,mdim,0)
c
c     modify v(2,2) and its equal
c     v(n-1,n-1) so the trace identity is satisfied
c
      sum=zero
```

```
      do 100 k=1,n
         if(k.eq.2.or.k.eq.nsub1) go to 100
         sum=sum+v(k,k)
  100 continue
      v(2,2)=half*(float(n)-summ2-sum)
      v(nsub1,nsub1)=v(2,2)
c
c     renormalise rows 2 through n-1 of v,
c     leaving diagonal elements fixed,
c
      call rwnorm(v,n,mdim,1)
      return
      end
c
      subroutine rwnorm(v,n,mdim,id)
c
c      Appl. Statist algorithm as 128.4 (1978), vol. 27
c      Davis C.S. and Stephens M.A.
c
c      Normalises rows of covariance matrix of normal order statistics
c      so that sum of row elements equals one.
c
c      arguments :      v - array (mdim,mdim) containing the covariance
c                           matrix approximation.
c                       n - sample size.
c                    mdim - row dimension of v in the calling program.
c                      id -
c
      integer n,mdim,id
      double precision v(mdim,n)
c
c      local integer variables
c
      integer i,j,k,l,m,ni,nj
c
c      local real variables
c
      double precision cnst,one,small,sum,term,zero
c
      data zero /0.0d0/, small /1.0d-12/, one /1.0d0/
c
      nhalf1 = (n+1)/2
      ni=n-1
      do 75 i=2,nhalf1
c
c     find sums of computed terms in each row
c
         sum=zero
         do 55 j=i,ni
            sum=sum+v(i,j)
   55    continue
         if(id .ne.0) sum = sum - v(i,i)
         if(abs(sum).lt.small) go to 75
c
c     normalise rows leaving appropriate elements fixed
c
         k=i-1
         if(id .ne. 0) k = i
         term=zero
         do 60 j=1,k
   60       term=term+v(i,j)
```

```
            l=ni+1
            do 65 j=l,n
        65     term=term+v(i,j)
            cnst=(one-term)/sum
            m=i
            if(id.ne.0) m=i+1
            nj=n-m+1
            do 70 j=m,ni
               v(i,j)=v(i,j)*cnst
               v(j,i)=v(i,j)
               v(ni,nj)=v(i,j)
               v(nj,ni)=v(i,j)
               nj=nj-1
        70    continue
            ni=ni-1
        75 continue
           return
           end
      c
           subroutine der(x,dx,d2x,d3x,d4x,d5x)
      c
      c     Appl. Statist. algorithm as 128.1 (1978), vol 27.
      c     Davis C.S. and Stephens M.A.
      c
      c     Computes derivatives for the david-johnson approximation to the
      c     variances and covariances of normal order statistics.
      c
      c     arguments : x - real number at which derivative is calculated.
      c                 dx - first derivative of normal probability integral
      c                      evaluated at x.
      c                  :                 :               :
      c                d5x - fifth derivative of normal probability integral
      c                      evaluated at x.
      c
           double precision x,dx,d2x,d3x,d4x,d5x
      c
      c     local real variables
      c
           double precision forty6,one,onept5,rad2pi,seven,six,
          1       term,twent4,two,twopi,x2
      c
      c     initialise constants
      c
           data one /1.0d0/, onept5 /1.5d0/, two /2.0d0/,
          1     six /6.0d0/,seven /7.0d0/, twent4 /24.0d0/,
          2     forty6 /46.0d0/, rad2pi /2.506628274631d0/,
          3     twopi/6.2831853071796d0/
           x2=x*x
           dx=rad2pi*exp(x2/two)
           d2x=twopi*x*exp(x2)
           d3x=twopi*rad2pi*(two*x2+one)*exp(onept5*x2)
           term=twopi*twopi*exp(two*x2)
           d4x=term*x*(six*x2+seven)
           d5x=term*dx*(x2*(twent4*x2+forty6)+seven)
           return
           end
      c
           double precision function var(dxr,d2xr,d3xr,d4xr,d5xr,pr,qr)
      c
      c     Appl. Statist. algorithm 128.2 (1978), vol 27
      c
```

```
c      Computes David-Johnson approximation for the variance of
c      the rth largest order statistic from the normal dist. for
c      a sample size n.
c
c      arguments : dxr - first derivative of normal probability integral
c                        evaluated at xr.
c                  :                  :                 :
c                d5xr - fifth derivative of normal probability integral
c                        evaluated at xr.
c                  pr - expected value of rth largest order statistic
c                        from uniform dist. ( = r/(n+1) r=1,...,n).
c                  qr - 1-pr
c                        n.b. xr is the inverse normal probability integral
c                        of pr.
c
       double precision dxr,d2xr,d3xr,d4xr,d5xr,pr,qr
c
c      local real variables
c
       double precision d2xr2,dxr2,fiveth,fourth,half,onept5,prqr,
      1       qrmpr,rn2,rn22,rn23,two,three
c
       common /cons/ rn2,rn22,rn23
c
c      initialise constants
c
       data fourth /0.25d0/, half /0.5d0/, onept5 /1.5d0/,
      1     fiveth /1.6666666667d0/, two /2.0d0/ , three /3.0d0/
       dxr2=dxr*dxr
       prqr=pr*qr
       var=prqr*dxr2/rn2
c
c     to order (n+2)**(-2)
c
       qrmpr=qr-pr
       d2xr2=d2xr*d2xr
       var = var+prqr/rn22*(two*qrmpr*dxr*d2xr+prqr*
      1    (dxr*d3xr+half*d2xr2))
c
c     to order (n+2)**(-3)
c
       var = var+prqr/rn23*(-two*qrmpr*dxr*d2xr+
      1    (qrmpr*qrmpr-prqr)*(two*dxr*d3xr+onept5*d2xr2)
      2    +prqr*qrmpr*(fiveth*dxr*d4xr+three*d2xr*d3xr)
      3    +fourth*prqr*prqr*(dxr*d5xr+two*d2xr*d4xr+
      4    fiveth*d3xr*d3xr))
       return
       end
c
       double precision function cov(dxr,d2xr,d3xr,d4xr,d5xr
      1                     ,pr,qr,dxs,d2xs,d3xs,d4xs,d5xs,ps)
c
c      Appl. statist. algorithm as 128.3 (1978), vol. 27
c      Davis C.S. and Stephens M.A.
c
c      Computes David-Johnson approximation for covariance between rth
c      and sth order statistics from the normal dist. for a sample size n.
c
c      arguments : dxr - first derivative of normal probability integral
c                        evaluated at xr.
c                  :                   :              :
```

```
c                d5xr - fifth derivative of normal probability integral
c                       evaluated at xr.
c                  pr - expected value of rth order statistic from
c                       uniform dist. ( = r/(n+1) r=1,...n ).
c                  qr - 1-pr.
c                 dxs - first derivative of normal probability integral
c                       evaluated at xs.
c                  :                    :            :
c                d5xs - fifth derivative of normal probability integral
c                       evaluated at xs.
c                  ps - expected value of sth order statistic from
c                       uniform distribution. ( = s/(n+1) s=1,...,n).
c                       n.b. xr is the inverse normal probability
c                       integral of pr etc.
c
      double precision dxr,d2xr,d3xr,d4xr,d5xr,pr,qr,dxs,d2xs,
     1         d3xs,d4xs,d5xs,ps
c
c      local real variables
c
      double precision eigth,five6,fourth,half,one,onept5,pr2,
     1       prqr,prqs,ps2,psqr,psqs,qr2,qrmpr,qs,qs2,qsmps,
     2       rn2,rn22,rn23,term1,term2,term3,term4,term5,
     3       three,twelth,two
c
      common /cons/ rn2,rn22,rn23
c
c      initialise constants
c
      data twelth /0.0833333333333d0/, eigth /0.125d0/,
     1     fourth /0.25d0/, half /0.5d0/,
     2     five6 /0.833333333333d0/, one /1.0d0/,
     3     onept5 /1.5d0/, two /2.0d0/, three /3.0d0/
c
      qs=one-ps
      prqs=pr*qs
      cov=prqs*dxr*dxs/rn2
c
c     to order (n+2)**(-2)
c
      qrmpr=qr-pr
      qsmps=qs-ps
      prqr=pr*qr
      psqs=ps*qs
      cov = cov+prqs/rn22*(qrmpr*d2xr*dxs+
     1    qsmps*dxr*d2xs+half*prqr*d3xr*dxs+
     2    half*psqs*dxr*d3xs+half*prqs*d2xr*d2xs)
c
c     to order (n+2)**(-3)
c
      pr2=pr*pr
      qr2=qr*qr
      ps2=ps*ps
      qs2=qs*qs
      psqr=ps*qr
      term1=-d2xr*dxs*qrmpr-qsmps*dxr*d2xs+
     1      (qrmpr*qrmpr-prqr)*d3xr*dxs
      term2=(qsmps*qsmps-psqs)*dxr*d3xs+(onept5*qrmpr*
     1       qsmps+half*psqr-two*prqs)*d2xr*d2xs
      term3=five6*(prqr*qrmpr*d4xr*dxs+psqs*qsmps*dxr*
     1      d4xs)+(prqs*qrmpr+half*prqr*qsmps)*d3xr*d2xs
```

```
       term4=(prqs*qsmps+half*psqs*qrmpr)*d2xr*d3xs+
      1      eigth*(pr2*qr2*d5xr*dxs+ps2*qs2*dxr*d5xs)
       term5=fourth*(pr2*qr*qs*d4xr*d2xs+pr*ps*qs2*
      1      d2xr*d4xs)+twelth*(two*pr2*qs2+three*pr*qr*
      2      ps*qs)*d3xr*d3xs
       cov = cov+prqs/rn23*(term1+term2+term3+term4+term5)
       return
       end
c
        double precision function v11(n, ifault)
c
c       ASR 72 (Remark on AS 128) Applied Stats. (1988) vol. 37 (1)
c
c       Calculates an approximation to the variance of the largest
c       normal order statistic.
c
        integer n, ifault
c
c       Local variables
c
        double precision zero, one, a0, a1, a2, a3, a4, a5, a6, x,
     +          d0, d1, d2, d3, d4, d5, d6, pt09, c0, c1, c2, c3,
     +          c4, c5, c6, c7, c8, c9, mpt15, b0, b1, b2, b3, b4
        parameter (mpt15 = -0.15, b0 = -0.934d-4, zero = 0.d0,
     +          one = 1.d0,
     +          b1 = -0.5950321d0, b2 = 0.0165504d0, b3 = 0.0056975d0,
     +          pt09 = 0.091105452691946d0, c0 = 0.7956d-11,
     +          c1 = -0.595628869836878d0, c2 = 0.08967827948053d0,
     +          c3 = -0.007850066416039d0, c4 = -0.296537314353d-3,
     +          c5 = 0.215480033104d-3, c6 = -0.33811291323d-4,
     +          c7 = 0.2738431187d-5, c8 = -0.106432868d-6,
     +          c9 = 0.1100251d-8, a0 = 0.04619831847696d0,
     +          a1 = -0.147930264017706d0, a2 = -0.451288155800301d0,
     +          a3 = 0.010055707621709d0, a4 = 0.007412441980877d0,
     +          a5 = -0.001143407259055d0, a6 = 0.54428754576d-4,
     +          d0 = 0.093256818332708d0, d1 = 1.336952989217635d0,
     +          d2 = -1.783195691545387d0, d3 = 0.488682076188729d0,
     +          d4 = -0.078737246197474d0, d5 = 0.00662561987806d0,
     +          d6 = -0.226486218258d-3, b4 = -0.8531d-3)
c
        v11 = zero
        ifault = 1
        if (n .lt. 1) return
        ifault = 0
        if (n .eq. 1) then
          v11 = one
          return
        end if
c
        x = n
        if (n .gt. 370) then
          x = (x**mpt15 - one) / mpt15
          v11 = exp(b0 + x*(b1 + x*(b2 + x*(b3 + x*b4))))
        else if (n .le. 100) then
          x = (x**pt09 - one) / pt09
          v11 = exp(c0 + x*(c1 + x*(c2 + x*(c3 + x*(c4 + x*(c5 + x*
     +          (c6 + x*(c7 + x*(c8 + x*c9)))))))))
        else if (n .le. 200) then
          x = log(a0 + x)
          v11 = exp(a1 + x*(a2 + x*(a3 + x*(a4 + x*(a5 + x*a6)))))
        else
```

```
            x = log(d0 + x)
            v11 = exp(d1 + x*(d2 + x*(d3 + x*(d4 + x*(d5 + x*d6)))))
          end if
c
          return
          end
```

```
      SUBROUTINE SIMLP(N, X, Y, SAD, ALPHA, BETA, D, ITER, INEXT,
     *   IFAULT)
C
C     ALGORITHM AS 132  APPL. STATIST. (1978) VOL.27, NO.3
C
C     SIMPL:   Fit  Y = ALPHA + BETA.X + error
C
      REAL X(N), Y(N), D(N)
      INTEGER INEXT(N)
      DATA ACU/1.0E-06/, BIG/1.0E19/, HALF/0.5/, ZERO/0.0/, ONE/1.0/,
     *     TWO/2.0/
C
C     Initial settings
C
      IFAULT = 0
      ITER = 0
      AHALF = HALF + ACU
      AONE = AHALF + AHALF
C
C     Determine initial basis
C
      D(1) = ZERO
      Y1 = Y(1)
      IBAS1 = 1
      A1 = X(1)
      DO 10 I = 2, N
        IF (ABS(A1 - X(I)) .LT. ACU) GO TO 10
        A2 = X(I)
        IBAS2 = I
        Y2 = Y(I)
        GO TO 20
   10 CONTINUE
      IFAULT = 1
      RETURN
C
C     Calculate initial beta value
C
   20 DET = ONE / (A2 - A1)
      AAAA = (A2 * Y1 - A1 * Y2) * DET
      BBBB = (Y2 - Y1) * DET
C
C     Calculate initial D-vector
C
      DO 30 I = 2, N
        DDD = Y(I) - (AAAA + BBBB * X(I))
        D(I) = SIGN(ONE, DDD)
   30 CONTINUE
      TOT1 = ONE
      TOT2 = X(IBAS2)
      D(IBAS2) = - ONE
      DO 40 I = 2, N
        TOT1 = TOT1 + D(I)
        TOT2 = TOT2 + D(I) * X(I)
   40 CONTINUE
      T = (A2 * TOT1 - TOT2) * DET
      IF (ABS(T) .LT. AONE) GO TO 50
      DET = - DET
      GO TO 70
C
C     Main iterative loop begins
C
```

```
     50 T = (TOT2 - A1 * TOT1) * DET
        IF (ABS(T) .LT. AONE) GO TO 160
        IFLAG = 2
        IOUT = IBAS2
        X(IOUT) = A1
        AAA = A1
        BBB = A2
        GO TO 80
     60 T = (TOT2 - A2 * TOT1) * DET
        IF (ABS(T) .LT. AONE) GO TO 160
     70 IFLAG = 1
        BBB = A1
        AAA = A2
        IOUT = IBAS1
     80 RHO = SIGN(ONE, T)
        T = HALF * ABS(T)
        DET = DET * RHO
C
C     Perform partial sort of ratios
C
        INEXT(IBAS1) = IBAS2
        RATIO = BIG
        SUM = AHALF
        DO 120 I = 1, N
          DDD = (X(I) - AAA) * DET
          IF (DDD * D(I) .LE. ACU) GO TO 120
          TEST = (Y(I) - AAAA - BBBB * X(I)) / DDD
          IF (TEST .GE. RATIO) GO TO 120
          J = IBAS1
          SUM = SUM + ABS(DDD)
     90   ISAVE = ABS(INEXT(J))
          IF (TEST .GE. D(ISAVE)) GO TO 110
          IF (SUM .LT. T) GO TO 100
          SUBT = ABS((X(ISAVE) - AAA) * DET)
          IF (SUM - SUBT .LT. T) GO TO 100
          SUM = SUM - SUBT
          D(ISAVE) = SIGN(1, INEXT(J))
          INEXT(J) = INEXT(ISAVE)
          GO TO 90
    100   J = ISAVE
          ISAVE = ABS(INEXT(J))
          IF (TEST .LT. D(ISAVE)) GO TO 100
    110   INEXT(I) = INEXT(J)
          INEXT(J) = SIGN(I, INT(D(I)))
          D(I) = TEST
          IF (SUM .LT. T) GO TO 120
          IIN = ABS(INEXT(IBAS1))
          RATIO = D(IIN)
    120 CONTINUE
C
C     Update basic indicators
C
        IIN = ABS(INEXT(IBAS1))
        J = IIN
    130 ISAVE = ABS(INEXT(J))
        IF (ISAVE .EQ. IBAS2) GO TO 140
        ZZZ = SIGN(1, INEXT(J))
        TOT1 = TOT1 - ZZZ - ZZZ
        TOT2 = TOT2 - TWO * ZZZ * X(ISAVE)
        D(ISAVE) = - ZZZ
        J = ISAVE
```

```
          GO TO 130
    140 ZZZ = SIGN(1, INEXT(IBAS1))
          TOT1 = TOT1 - RHO - ZZZ
          TOT2 = TOT2 - RHO * BBB - ZZZ * X(IIN)
          D(IOUT) = - RHO
          ITER = ITER + 1
          IF (IFLAG .EQ. 1) GO TO 150
          X(IBAS2) = A2
          IBAS2 = IIN
          D(IBAS2) = - ONE
          A2 = X(IIN)
          Y2 = Y(IIN)
          DET = ONE / (A1 - A2)
          AAAA = (A1 * Y2 - A2 * Y1) * DET
          BBBB = (Y1 - Y2) * DET
          GO TO 60
    150 IBAS1 = IIN
          A1 = X(IIN)
          D(IBAS1) = ZERO
          Y1 = Y(IIN)
          DET = ONE / (A2 - A1)
          AAAA = (A2 * Y1 - A1 * Y2) * DET
          BBBB = (Y2 - Y1) * DET
          GO TO 50
C
C     Calculate optimal sum of absolute deviations
C
    160 SAD = ZERO
          DO 170 I = 1, N
            D(I) = Y(I) - (AAAA + BBBB * X(I))
            SAD = SAD + ABS(D(I))
    170 CONTINUE
          ALPHA = AAAA
          BETA = BBBB
C
          RETURN
          END
```

```
      SUBROUTINE EXTR(BP,EP,M,N,F,YM,XM,NE,E1,E2,N1,NP,X1,Y1,X2,Y2,
     +                IFAULT)
C
C        ALGORITHM AS 133 APPL. STATIST. (1978) VOL.27, NO.3
C
C        OPTIMIZATION OF ONE DIMENSIONAL MULTIMODAL FUNCTIONS
C
C     Auxiliary routines required: The user must provide a real function
C     F(X) to return the value of the function to be minimized.
C     Brent's LOCALM has been added to this routine.
C
      DIMENSION X1(NP),Y1(NP),X2(NP),Y2(NP)
      EXTERNAL YL,F
      REAL LOCALM
      REAL ONE,TEN,TWENT5,ZERO,SEVEN,SMALL,NINET8,PT99,TWO,THREE,FIVE
      DOUBLE PRECISION EE1, EE2, EE3, DM17
C
      COMMON /EXTR1/A, EB, YH, XJ1, XJ2, YJ1, YJ2
C
C     K is a machine-dependant constant.   10^(-K) should be set to the
C     smallest number which can be represented in the machine.
C
      DATA K/37/, EPSL/0.0/
      DATA ONE/1.0/, TEN/10.0/, TWENT5/25.0/, ZERO/0.0/, SEVEN/7.0/,
     +    SMALL/3.0E-4/, NINET8/98.0/, PT99/0.99/, TWO/2.0/, THREE/3.0/,
     +    FIVE/5.0/
C
      K1 = K/3
      CUND = -2*K
      CP5 = TEN** (K1-1)
      CP6 = TEN*CP5
      CM6 = ONE/CP6
      CM5 = ONE/CP5
      DM17 = 10.0D0** (-K+2)
      CM17 = DM17
      IFAULT = 0
      IF (NE.GE.6) GO TO 5
      IFAULT = 3
      RETURN

    5 N1 = NE
      N2 = N1 + 2
      CM = M
      A = MIN(BP,EP)
      EB = MAX(BP,EP) - A
      EB2 = EB/TWENT5
      IF (EB.GE.CM5 .AND. EB.LE.CP5) GO TO 10
      IFAULT = 1
      RETURN

   10 DA = EB/REAL(N1-1)
      N6 = N1 + 3
      DO 20 J = 4,N6
        X1(J) = A
        Y1(J) = F(A)*CM
        A = A + DA
   20 CONTINUE
      A = ZERO
      YM = Y1(4)
      XM = X1(4)
C
```

```
C        A IS ESTIMATE OF PARAMETER OF WIENER PROCESS
C        (PARAMETER OF MODEL OF FUNCTION F)
C
      DO 30 J = 5,N6
        A = A + (Y1(J)-Y1(J-1))**2
        IF (YM.LE.Y1(J)) GO TO 30
        YM = Y1(J)
        XM = X1(J)
   30 CONTINUE
      YH = YM
      A = SQRT(A/EB)*SEVEN
      IF (A.GE.CM6 .AND. A.LE.CP6) GO TO 40
      IFAULT = 2
      RETURN
C
C        A IS EVALUATED.  N5 IS THE NUMBER OF LOCAL OPTIMA
C        COMPUTED WITH ACCURACY E1, E2
C
   40 N5 = 0
      DO 50 J = 4,N2
   50 Y2(J) = ZERO
C
C        BAYESIAN STEP - FIND INTERVAL OF BIGGEST IMPROVEMENT (J)
C
   60 DO 70 J = 4,N2
        IF (Y2(J).GT.ZERO) GO TO 70
        XJ1 = X1(J)
        XJ2 = X1(J+1)
        YJ1 = Y1(J)
        YJ2 = Y1(J+1)
        TL = (XJ2-XJ1)*SMALL
        Y2(J) = LOCALM(XJ1,XJ2,EPSL,TL,YL,X2(J))
   70 CONTINUE
   80 P1 = Y2(4)
      J = 4
      DO 90 N3 = 5,N2
        IF (Y2(N3).GE.P1) GO TO 90
        J = N3
        P1 = Y2(N3)
   90 CONTINUE
C
C        SET UP X1, Y1, X2, Y2 FOR NEXT STEP
C
      N4 = N1 - J + 3
      N7 = N1 + 4
      DO 100 N3 = 1,N4
        N7 = N7 - 1
        X1(N7+1) = X1(N7)
        Y1(N7+1) = Y1(N7)
        X2(N7) = X2(N7-1)
        Y2(N7) = Y2(N7-1)
  100 CONTINUE
      X1(J+1) = X2(J)
      Y1(J+1) = F(X2(J))*CM
      N4 = 1
      IF (Y1(J+1).GT.YM) GO TO 110
      YM = Y1(J+1)
      XM = X1(J+1)
C
C        IF NEW EVALUATION IS BETTER THAN YM,
C        X2 AND Y2 WILL BE UPDATED (N4 = 2)
```

```
C
      N4 = 2
  110 N1 = N1 + 1
      N2 = N1 + 2
      YH = YM
C
C        IF NUMBER OF EVALUATIONS OF F EXCEEDS (N-1) GO TO END
C
      IF (N1.LT.N) GO TO 115
      IFAULT = 4
      GO TO 310
C
C        P4 IS PROBABILITY OF EVALUATING
C        XM, YM WITH ACCURACY E1, E2
C
  115 P4 = ONE
      DO 120 N3 = 4,N2
        IF (Y2(N3).GT.ZERO) GO TO 120
        P1 = -NINET8* ((Y1(N3)-YM+E1)/A)* ((Y1(N3+1)-YM+E1)/A)/
     +         (X1(N3+1)-X1(N3))
        IF (P1.GT.CUND) P4 = P4* (ONE-EXP(P1))
  120 CONTINUE
      IF (P4.GE.PT99) GO TO 310
C
C        IF N5 = 0 A IS UPDATED IN EACH FIFTH STEP
C
      DO 130 N3 = 1,N1,5
        IF (N3.EQ.N1) GO TO 140
  130 CONTINUE
      GO TO 160

  140 IF (N5.GT.0) GO TO 160
      A = ZERO
      N4 = 2
      DO 150 N3 = 4,N2
  150 A = A + ((Y1(N3+1)-Y1(N3))**2)/ (X1(N3+1)-X1(N3))
      A = SQRT(A/REAL(N1-1))*SEVEN
C
C        PREPARATION OF BOUNDS OF X1, Y1 FOR TESTING -
C        IS INTERVAL OF LOCAL OPTIMUM FOUND
C
  160 X1(3) = (THREE*X1(4)-X1(5))/TWO
      X1(1) = X1(3)
      X1(2) = X1(3)
      X1(N1+4) = (THREE*X1(N1+3)-X1(N1+2))/TWO
      X1(N1+5) = X1(N1+4)
      X1(N1+6) = X1(N1+4)
      NR = N1 + 3
      DO 170 N3 = 1,3
        Y1(N3) = Y1(5)
        NR = NR + 1
        Y1(NR) = Y1(N1+2)
  170 CONTINUE
      N7 = MAX(1,J-5)
      N8 = MIN(N1,J+1)
C
C        TEST IF INTERVAL OF LOCAL OPTIMUM FOUND
C
      DO 180 N3 = N7,N8
        IF (Y1(N3).GE.Y1(N3+1) .AND. Y1(N3+1).GE.Y1(N3+2) .AND.
     +        Y1(N3+2).GT.Y1(N3+3) .AND. Y1(N3+3).LT.Y1(N3+4) .AND.
```

```
     +        Y1(N3+4).LE.Y1(N3+5) .AND. Y1(N3+5).LE.Y1(N3+6) .AND.
     +        (X1(N3+6)-X1(N3)).LE. (EB/FIVE)) GO TO 200
  180 CONTINUE
      IF (N4.EQ.2) GO TO 60
C
C        IF INTERVAL OF LOCAL OPTIMUM IS FOUND GO TO
C        LOCAL OPTIMIZATION, ELSE BAYESIAN STEP
C
      N7 = J + 1
      DO 190 N3 = J,N7
        XJ1 = X1(N3)
        XJ2 = X1(N3+1)
        YJ1 = Y1(N3)
        YJ2 = Y1(N3+1)
        IF ((XJ2-XJ1).LT.EB2 .AND. Y2(N3-1).GT.ZERO .AND.
     +        YJ1.LT.YJ2) GO TO 185
        TL = (XJ2-XJ1)*SMALL
        Y2(N3) = LOCALM(XJ1,XJ2,EPSL,TL,YL,X2(N3))
        GO TO 190

  185   Y2(N3) = ONE
  190 CONTINUE
      IF (Y2(N7+1).GT.ZERO .AND. Y1(N7).GT.Y1(N7+1) .AND.
     +    X1(N7+1)-X1(N7).LT.EB2) Y2(N7) = ONE
      IF (Y2(N7).GT.ZERO .AND. Y1(N7-1).GT.Y1(N7) .AND.
     +    X1(N7)-X1(N7-1).LT.EB2) Y2(N7-1) = ONE
      GO TO 80
C
C        START OF LOCAL OPTIMIZATION
C
  200 N6 = N3 + 5
      DO 210 N7 = N3,N6
  210 Y2(N7) = ONE
      N3 = N3 + 3
      N5 = N5 + 1
      N7 = 1
  220 EE1 = DBLE(Y1(N3-1)-Y1(N3))
      EE2 = DBLE(X1(N3-1)-X1(N3))
      IF (ABS(EE1).LE.DM17 .OR. ABS(EE2).LE.DM17) GO TO 60
      EE3 = DBLE(TWO* (Y1(N3-1)-Y1(N3+1)))/EE1 -
     +      DBLE(TWO* (X1(N3-1)-X1(N3+1)))/EE2
      IF (ABS(EE3).LE.DM17) GO TO 60
      XA = (DBLE(X1(N3-1)+X1(N3))*DBLE(Y1(N3-1)-Y1(N3+1))/EE1-
     +     DBLE(X1(N3-1)-X1(N3+1))*DBLE(X1(N3-1)+X1(N3+1))/EE2)/EE3
      EE1 = DBLE(X1(N3)-XA)
      EE2 = DBLE(X1(N3-1)-XA)
      EE3 = DBLE(Y1(N3-1)-Y1(N3))
      IF (ABS(EE2).GE.DM17) GO TO 230
      P1 = Y1(N3-1)
      GO TO 250

  230 EE1 = ONE - (EE1/EE2)**2
      IF (ABS(EE1).GE.DM17) GO TO 240
      IF (EE3.LE.ABS(EE1)*1.0D6) GO TO 240
      P1 = Y1(N3-1) - 1.0E5
      GO TO 250

  240 EE3 = EE3/EE1
      P1 = Y1(N3-1) - EE3
  250 YA = F(XA)*CM
      P3 = ABS(XA-X1(N7))
```

```
          IF (YA.GE.YM) GO TO 260
          YM = YA
          XM = XA
      260 N1 = N1 + 1
          N2 = N1 + 2
          YH = YM
          N7 = N3
          IF (XA.GT.X1(N3)) N7 = N7 + 1
          N6 = N1 - N7 + 3
          N4 = N1 + 3
          DO 290 N8 = 1,N6
            N4 = N4 - 1
            X1(N4+1) = X1(N4)
            Y1(N4+1) = Y1(N4)
            X2(N4) = X2(N4-1)
            Y2(N4) = Y2(N4-1)
      290 CONTINUE
          X1(N7) = XA
          Y1(N7) = YA
          Y2(N7) = ONE
          Y2(N7-1) = ONE
          X1(3) = (THREE*X1(4)-X1(5))/TWO
          X1(1) = X1(3)
          X1(2) = X1(3)
          X1(N1+4) = (THREE*X1(N1+3)-X1(N1+2))/TWO
          X1(N1+5) = X1(N1+4)
          X1(N1+6) = X1(N1+4)
          NR = N1 + 3
          DO 300 N8 = 1,3
            Y1(N8) = Y1(5)
            NR = NR + 1
            Y1(NR) = Y1(N1+2)
      300 CONTINUE
          IF (N1.LT.N) GO TO 305
          IFAULT = 4
          GO TO 310

      305 P2 = ABS(P1-YA)
          IF (P2.LE.E1 .AND. P3.LE.E2 .OR. P2.LE.CM17 .OR.
         +    P3.LE.CM17) GO TO 60
          IF (N3.EQ.N7 .AND. Y1(N7).GT.Y1(N7+1) .OR.
         +    N3+1.EQ.N7 .AND. Y1(N7).LT.Y1(N7-1)) N3 = N3 + 1
          GO TO 220

      310 YM = YM*CM
          RETURN

          END
C
C
          FUNCTION YL(C)
C
C            ALGORITHM AS 133.1 APPL. STATIST. (1978) VOL.27, NO.3
C
C            -YL IS THE MEAN OF THE IMPROVEMENT
C
          REAL VSMALL,ZERO
          COMMON /EXTR1/A,EB,YH,XJ1,XJ2,YJ1,YJ2
          DATA VSMALL/1.0E-17/,ZERO/0.0/
C
          P1 = EB*VSMALL
```

```
      IF (C-XJ1.LE.P1 .OR. XJ2-C.LE.P1) GO TO 1
      P1 = XJ2 - XJ1
      P2 = A*SQRT((C-XJ1)* (XJ2-C)/P1)
      P1 = (YJ1* (XJ2-C)+YJ2* (C-XJ1))/P1 - YH
      P3 = P1/P2
      YL = P1*0.65*EXP(-0.443* (0.75+P3)**2) -
     +     P2*0.3989*EXP(- (P3**2)/2.0)
      RETURN

    1 YL = ZERO
      RETURN

      END
C
C
      REAL FUNCTION LOCALM(A, B, EPS, T, F, X)
C
C     Entered from pages 188-190 of 'Algorithms for minimization without
C     derivatives' by Richard P. Brent, Prentice-Hall, 1973
C     Comments added from the Algol version on pages 79-80.
C
      REAL A, B, EPS, T, F, X
      EXTERNAL F
C
C     Local variables
C
      REAL CONST
      REAL SA, SB, D, E, M, P, Q, R, TOL, T2, U, V, W, FU, FV, FW, FX
C
C     CONST = (3 - sqrt(5))/2
C
      DATA CONST/0.381966/
C
      SA = A
      SB = B
      X = SA + CONST*(SB - SA)
      W = X
      V = W
      E = 0.0
      FX = F(X)
      FW = FX
      FV = FW
C
C     Main loop
C
   10 M = 0.5*(SA + SB)
      TOL = EPS*ABS(X) + T
      T2 = 2.0*TOL
C
C     Check stopping criterion
C
      IF (ABS(X-M) .LE. T2-0.5*(SB-SA)) GO TO 190
      R = 0.0
      Q = R
      P = Q
      IF (ABS(E) .LE. TOL) GO TO 40
C
C     Fit parabola
C
      R = (X - W)*(FX - FV)
      Q = (X - V)*(FX - FW)
```

```
      P = (X - V)*Q - (X - W)*R
      Q = 2.0*(Q - R)
      IF (Q .LE. 0.0) GO TO 20
      P = -P
      GO TO 30
   20 Q = -Q
   30 R = E
      E = D
   40 IF (ABS(P) .GE. ABS(0.5*Q*R)) GO TO 60
      IF ((P .LE. Q*(SA-X)) .OR. (P .GE. Q*(SB-X))) GO TO 60
C
C     A parabolic interpolation step
C
      D = P/Q
      U = X + D
C
C     F must not be evaluated too close to A or B
C
      IF ((U-SA .GE. T2) .AND. (SB-U .GE. T2)) GO TO 90
      IF (X .GE. M) GO TO 50
      D = TOL
      GO TO 90
   50 D = -TOL
      GO TO 90
C
C     A golden section step
C
   60 IF (X .GE. M) GO TO 70
      E = SB - X
      GO TO 80
   70 E = SA - X
   80 D = CONST*E
C
C     F must not be evaluated too close to X
C
   90 IF (ABS(D) .LT. TOL) GO TO 100
      U = X + D
      GO TO 120
  100 IF (D .LE. 0.0) GO TO 110
      U = X + TOL
      GO TO 120
  110 U = X - TOL
  120 FU = F(U)
C
C     Update A, B, V, W and X
C
      IF (FU .GT. FX) GO TO 150
      IF (U .GE. X) GO TO 130
      SB = X
      GO TO 140
  130 SA = X
  140 V = W
      FV = FW
      W = X
      FW = FX
      X = U
      FX = FU
      GO TO 10
  150 IF (U .GE. X) GO TO 160
      SA = U
      GO TO 170
```

```
      160 SB = U
      170 IF ((FU .GT. FW) .AND. (W .NE. X)) GO TO 180
          V = W
          FV = FW
          W = U
          FW = FU
          GO TO 10
      180 IF ((FU .GT. FV) .AND. (V .NE. X) .AND. (V .NE. W)) GO TO 10
          V = U
          FV = FU
          GO TO 10
C
      190 LOCALM = FX
          RETURN
          END
```

```
      SUBROUTINE BET1GT(BETAR)
C
C        ALGORITHM AS 134.1 APPL. STATIST. (1979) VOL.28 NO.1
C
C        GENERATES BETA RANDOM VARIABLES
C        WITH 0.0 .LT. ALPHA .LT. 1.0 AND BETA  .GT.  1.0
C        BY THE SWITCHING ALGORITHM OF ATKINSON AND WHITTAKER
C
C     The random number call has been changed from RANF(..) to RAND()
C     so that the Wichmann & Hill random number generator, AS 183
C     can be used.   To conform with Fortran-77, SAVE instructions have
C     been added.
C
      REAL ONE
      DATA ONE/1.0/
      COMMON /BETCOM/ AM1, BM1, ARECIP, BRECIP, T, R
      SAVE /BETCOM/
C
    1 UVAR = RAND()
      CALL FNE(EVAR)
      IF(UVAR .GT. R) GOTO 3
C
C     WARNING.  With some compilers, underflow can occur for small
C               values of alpha in executing the next instruction.
C
      BETAR = T*((UVAR/R)**ARECIP)
      IF(-BM1*LOG(ONE-BETAR) .GT. EVAR) GOTO 1
      RETURN
    3 BETAR = ONE - (ONE-T)*(((ONE-UVAR)/(ONE-R))**BRECIP)
      IF(-AM1*LOG(BETAR/T) .GT. EVAR) GOTO 1
      RETURN
      END
C
      SUBROUTINE TOPT(ALPHA, BETA, IFAULT)
C
C        ALGORITHM AS 134.2 APPL. STATIST. (1979) VOL.28 NO.1
C
C        SETS CONSTANTS FOR BETA VARIABLE GENERATION.
C        TOL IS TOLERANCE FOR ACCEPTING T AS SOLUTION OF GX = 0
C
      COMMON /BETCOM/ AM1, BM1, ARECIP, BRECIP, T, R
      SAVE /BETCOM/
      REAL ZERO, ONE, TWO, TOL
      DATA ZERO/0.0/, ONE/1.0/, TWO/2.0/, TOL/1.0E-5/
C
C        TEST PARAMETERS AND CALCULATE CONSTANTS
C
      IF(ALPHA .GT. ZERO .AND. ALPHA .LT. ONE) GOTO 1
      IFAULT = 1
      RETURN
    1 IF(BETA .GT. ONE) GOTO 2
      IFAULT = 2
      RETURN
    2 AM1 = ALPHA - ONE
      TTILDE = AM1/(AM1-BETA)
      IF(ONE-TTILDE*TTILDE .LT. ONE) GOTO 3
      IFAULT = 3
      RETURN
    3 BM1 = BETA - ONE
      ARECIP = ONE/ALPHA
      BRECIP = ONE/BETA
```

```
C
C        INITIAL VALUES FOR FALSE POSITION
C
      XA = ZERO
      GA = AM1
      XB = ONE
      GB = BETA
C
C        FALSE POSITION
C
    4 X = (XA*GB - XB*GA)/(GB - GA)
      AA = ONE - X
      BT = BETA*X
      GX = BT + AA**BM1*(AM1*AA - BT)
      IF(ABS(GX) .LT. TOL) GOTO 6
      IF(GX*GA .LT. ZERO) GOTO 5
C
C        MODIFICATION
C
      XA = X
      GA = GX
      GB = GB/TWO
      GOTO 4
C
C        MODIFICATION
C
    5 XB = XA
      GB = GA
      XA = X
      GA = GX
      GOTO 4
C
C        CALCULATE REMAINING CONSTANTS
C
    6 T = X
      R = BT/(BT + ALPHA*AA**BETA)
      RETURN
      END
C
      SUBROUTINE FNE(REX)
C
C        ALGORITHM AS134.3 APPL. STATIST. (1979) VOL.28 NO.1
C
C        GENERATES EXPONENTIAL RANDOM VARIABLES
C        BY THE METHOD OF VON NEUMANN
C
      REAL ZERO, ONE
      DATA ZERO/0.0/, ONE/1.0/
      A = ZERO
    1 U = RAND()
      UO = U
    2 USTAR = RAND()
      IF(U .LT. USTAR) GOTO 3
      U = RAND()
      IF(U .LT. USTAR) GOTO 2
      A = A + ONE
      GOTO 1
    3 REX = A + UO
      RETURN
      END
```

```
      SUBROUTINE LFNORM(N, M, NDIM, MDIM, X, Y, BETA, Z, KY, IFAULT)
C
C     ALGORITHM AS 135  APPL. STATIST. (1979) VOL.28, NO. 1
C
C     Min-Max estimates for a linear multiple regression problem
C
      DOUBLE PRECISION X(NDIM, MDIM), Y(NDIM), BETA(MDIM), Z, HILO(20),
     +  XRXF(20), XSXF(20), LU(20, 20), ACU, BIG, ZERO, ONE, DEV1,
     +  SIGR, SUMXR, DEVIAT, YEST, YDEV, SUMXS, RATIO, TEST, DELTA,
     +  DIV, SWING, SAVE, SIGS, TOP, TWO
      INTEGER IBASE(20), SSS, RRR, INDX(20)
      LOGICAL INTL
C
      DATA ACU/1.D-15/, BIG/1.D30/, ZERO/0.D0/, ONE/1.D0/, TWO/2.D0/
C
      IFAULT = 0
      KY = 0
      Z = ZERO
      M1 = M - 1
C
C     Set up initial LU decomposition
C
      DO 10 I = 1, M
   10 INDX(I) = I
      INTL = .TRUE.
      KKK = 1
      CALL UPDATE(KKK, X, LU, IBASE, INDX, INTL, N, M, NDIM, MDIM,
     +  IFAULT)
      IF (IFAULT .NE. 0) RETURN
      INTL = .FALSE.
      IROW = KKK
C
C     Calculate beta value
C
      K = INDX(1)
      K1 = IBASE(1)
      BETA(K) = Y(K1) / LU(K, 1)
      DO 30 II = 2, M
        K = INDX(II)
        K1 = IBASE(II)
        BETA(K) = Y(K1)
        II1 = II - 1
        DO 20 I = 1, II1
          KK = INDX(I)
          BETA(K) = BETA(K) - LU(KK, II) * BETA(KK)
   20   CONTINUE
        BETA(K) = BETA(K) / LU(K, II)
   30 CONTINUE
      DO 40 II = 1, M1
        K1 = M - II
        K = INDX(K1)
        DO 40 I = 1, II
          KK = M - I + 1
          K2 = INDX(KK)
          BETA(K) = BETA(K) - LU(K2, K1) * BETA(K2)
   40 CONTINUE
C
C     Search for and set first violated constraint as R-th constraint
C
   50 IROW = IROW + 1
      IF (IROW .GT. N) RETURN
```

```
      DEV1 = ZERO
      DO 60 I = 1, M
   60 DEV1 = DEV1 + X(IROW, I) * BETA(I)
      DEV1 = DEV1 - Y(IROW)
      IF (ABS(DEV1) .LT. ACU) GO TO 50
      SIGR = SIGN(ONE, DEV1)
      RRR = IROW
C
C     Adjust for the R-th constraint
C
      K = INDX(1)
      XRXF(1) = X(RRR, K)
      DO 80 II = 2, M
        K = INDX(II)
        XRXF(II) = X(RRR, K)
        II1 = II - 1
        DO 70 I = 1, II1
   70   XRXF(II) = XRXF(II) - LU(K, I) * XRXF(I)
   80 CONTINUE
      K = INDX(M)
      XRXF(M) = XRXF(M) / LU(K, M)
      HILO(M) = SIGN(ONE, -SIGR * XRXF(M))
      SUMXR = SIGR - HILO(M) * XRXF(M)
      DO 100 II = 1, M1
        K1 = M - II
        K = INDX(K1)
        DO 90 I = 1, II
          K2 = M - I + 1
          XRXF(K1) = XRXF(K1) - LU(K, K2) * XRXF(K2)
   90   CONTINUE
        XRXF(K1) = XRXF(K1) / LU(K, K1)
        HILO(K1) = SIGN(ONE, -SIGR * XRXF(K1))
        SUMXR = SUMXR - HILO(K1) * XRXF(K1)
  100 CONTINUE
      Z = ABS(DEV1 / SUMXR)
C
C     Start of main iterative loop.
C     Search for the most violated S-th constraint
C
  110 SSS = 0
      DEVIAT = ACU
C
C     Calculate beta value
C
      K = INDX(1)
      K1 = IBASE(1)
      BETA(K) = (Y(K1) + Z * HILO(1)) / LU(K, 1)
      DO 130 II = 2, M
        K = INDX(II)
        K1 = IBASE(II)
        BETA(K) = Y(K1) + Z * HILO(II)
        II1 = II - 1
        DO 120 I = 1, II1
          KK = INDX(I)
          BETA(K) = BETA(K) - LU(KK, II) * BETA(KK)
  120   CONTINUE
        BETA(K) = BETA(K) / LU(K, II)
  130 CONTINUE
      DO 140 II = 1, M1
        K1 = M - II
        K = INDX(K1)
```

```
            DO 140 I = 1, II
              KK = M - I + 1
              K2 = INDX(KK)
              BETA(K) = BETA(K) - LU(K2, K1) * BETA(K2)
    140   CONTINUE
C
C       Calculate residuals
C
          DO 160 I = 1, N
            YEST = ZERO
            DO 150 J = 1, M
    150     YEST = YEST + X(I, J) * BETA(J)
            DEV1 = ABS(Y(I) - YEST) - Z
            IF (DEV1 .LE. DEVIAT) GO TO 160
            YDEV = YEST - Y(I)
            DEVIAT = DEV1
            SSS = I
    160   CONTINUE
C
C       Check if at optimum
C
          IF (SSS .EQ. 0) RETURN
C
C       Set up information on the S-th constraint
C
          SIGS = SIGN(ONE, YDEV)
          K = INDX(1)
          XSXF(1) = X(SSS, K)
          DO 180 II = 2, M
            K = INDX(II)
            XSXF(II) = X(SSS, K)
            II1 = II - 1
            DO 170 I = 1, II1
    170     XSXF(II) = XSXF(II) - LU(K, I) * XSXF(I)
    180   CONTINUE
          K = INDX(M)
          XSXF(M) = XSXF(M) / LU(K, M)
          SUMXS = -SIGS + HILO(M) * XSXF(M)
          DO 200 II = 1, M1
            K1 = M - II
            K = INDX(K1)
            DO 190 I = 1, II
              K2 = M - I + 1
              XSXF(K1) = XSXF(K1) - LU(K, K2) * XSXF(K2)
    190     CONTINUE
            XSXF(K1) = XSXF(K1) / LU(K, K1)
            SUMXS = SUMXS + HILO(K1) * XSXF(K1)
    200   CONTINUE
C
C       Search for minimum ratio
C
    210   KKK = 0
          RATIO = BIG
          DO 220 I = 1, M
            IF (SIGS * SIGN(ONE, XSXF(I)) .NE. HILO(I) .OR.
         +          ABS(XSXF(I)) .LT. ACU) GO TO 220
            TEST = ABS(XRXF(I) / XSXF(I))
            IF (TEST .GE. RATIO) GO TO 220
            RATIO = TEST
            KKK = I
    220   CONTINUE
```

```
C
C     Check if R-th constraint moves interior
C
      IF (KKK .NE. 0) GO TO 260
C
C     Process the movement of the R-th constraint
C
      DELTA = ABS(DEVIAT / SUMXS)
C
C     Calculate the largest tolerable delta
C
      DIV = ABS(SUMXR) - TWO
      IF (DIV .LT. ACU) GO TO 240
      SWING = TWO * Z / DIV
      IF (SWING .GE. DELTA) GO TO 240
C
C     Switch R and S constraint indicators
C
      SAVE = SUMXS
      SUMXS = -SUMXR + SIGR + SIGR
      SUMXR = -SAVE
      SAVE = SIGR
      SIGR = SIGS
      SIGS = -SAVE
      DEVIAT = ABS(SUMXS * DELTA) - TWO * Z
      Z = Z + DELTA
      DO 230 I = 1, M
        SAVE = XSXF(I)
        XSXF(I) = XRXF(I)
        XRXF(I) = SAVE
  230 CONTINUE
      I = RRR
      RRR = SSS
      SSS = I
      GO TO 210
C
C     Replace the R-th constraint with the S-th constraint
C
  240 SIGR = SIGS
      DO 250 I = 1, M
  250 XRXF(I) = XSXF(I)
      SUMXR = -SUMXS
      Z = Z + DELTA
      RRR = SSS
      GO TO 110
C
C     Process the movement of the K-th constraint
C
  260 DELTA = ABS(XRXF(KKK) * DEVIAT /
     +          (XRXF(KKK) * SUMXS + XSXF(KKK) * SUMXR))
      TOP = -TWO * Z * XRXF(KKK)
      DIV = XRXF(KKK) * XRXF(KKK) + HILO(KKK) * SUMXR
      IF (SIGN(ONE, TOP) .NE. SIGN(ONE, DIV)) GO TO 270
      IF (ABS(DIV) .LT. ACU) GO TO 270
      SWING = TOP / DIV
C
C     Check to see if the K-th constraint swings across
C
      IF (SWING .GE. DELTA) GO TO 270
      Z = Z + SWING
      DEVIAT = DEVIAT - SWING *
```

```
     +          ABS(SUMXS + XSXF(KKK) * SUMXR / XRXF(KKK))
      SUMXR = SUMXR + TWO * HILO(KKK) * XRXF(KKK)
      SUMXS = SUMXS - TWO * HILO(KKK) * XSXF(KKK)
      HILO(KKK) = -HILO(KKK)
      GO TO 210
C
C     Update XRXF and the LU of the current basis
C
  270 HILO(KKK) = SIGS
      SUMXR = SIGR
      XRXF(KKK) = XRXF(KKK) / XSXF(KKK)
      SUMXR = SUMXR - HILO(KKK) * XRXF(KKK)
      DO 280 I = 1, M
        IF (I .EQ. KKK) GO TO 280
        XRXF(I) = XRXF(I) - XSXF(I) * XRXF(KKK)
        SUMXR = SUMXR - HILO(I) * XRXF(I)
  280 CONTINUE
      IBASE(KKK) = SSS
C
C     Update LU decomposition
C
      CALL UPDATE(KKK, X, LU, IBASE, INDX, INTL, N, M, NDIM, MDIM,
     +  IFAULT)
      IF (IFAULT .NE. 0) RETURN
      Z = Z + DELTA
      KY = KY + 1
      GO TO 110
      END
C
      SUBROUTINE UPDATE(KKK, X, LU, IBASE, INDX, INTL, N, M, NDIM,
     +  MDIM, IFAULT)
C
C     ALGORITHM AS 135.1  APPL. STATIST. (1979) VOL.28, NO. 1
C
C     Update LU decomposition matrix
C
      DOUBLE PRECISION X(NDIM, MDIM), LU(20, 20), ACU, SUBT, PIVOT
      INTEGER IBASE(20), INDX(20)
      LOGICAL INTL
      DATA ACU/1.D-15/
C
      IROW = 0
      DO 90 II = KKK, M
        IF (INTL) GO TO 10
        IROW = IBASE(II)
        GO TO 20
   10   IROW = IROW + 1
        IF (IROW .LE. N) GO TO 20
        IFAULT = 1
        RETURN
   20   DO 30 I = 1, M
   30   LU(I, II) = X(IROW, I)
C
C     Set up representation of incoming row
C
        IF (II .EQ. 1) GO TO 60
        II1 = II - 1
        DO 50 ICOL = 1, II1
          K = INDX(ICOL)
          SUBT = LU(K, II)
          J = ICOL + 1
```

```
            DO 40 I = J, M
              K = INDX(I)
              LU(K, II) = LU(K, II) - SUBT * LU(K, ICOL)
   40       CONTINUE
   50     CONTINUE
C
C     Find maximum entry
C
   60     PIVOT = ACU
          KK = 0
          DO 70 I = II, M
            K = INDX(I)
            IF (ABS(LU(K, II)) .LE. PIVOT) GO TO 70
            PIVOT = ABS(LU(K, II))
            KK = I
   70     CONTINUE
          IF (KK .EQ. 0) GO TO 10
C
C     Switch order
C
          ISAVE = INDX(KK)
          INDX(KK) = INDX(II)
          INDX(II) = ISAVE
C
C     Put into columns of LU one at a time
C
          IF (INTL) IBASE(II) = IROW
          IF (II .EQ. M) GO TO 90
          J = II + 1
          DO 80 I = J, M
            K = INDX(I)
            LU(K, II) = LU(K, II) / LU(ISAVE, II)
   80     CONTINUE
   90   CONTINUE
        KKK = IROW
        RETURN
        END
```

```
      SUBROUTINE KMNS(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D,
     *    ITRAN, LIVE, ITER, WSS, IFAULT)
C
C     ALGORITHM AS 136  APPL. STATIST. (1979) VOL.28, NO.1
C
C     Divide M points in N-dimensional space into K clusters so that
C     the within cluster sum of squares is minimized.
C
      INTEGER IC1(M), IC2(M), NC(K), NCP(K), ITRAN(K), LIVE(K)
      REAL    A(M,N), D(M), C(K,N), AN1(K), AN2(K), WSS(K), DT(2)
      REAL    ZERO, ONE
C
C     Define BIG to be a very large positive number
C
      DATA BIG /1.E30/, ZERO /0.0/, ONE /1.0/
C
      IFAULT = 3
      IF (K .LE. 1 .OR. K .GE. M) RETURN
C
C     For each point I, find its two closest centres, IC1(I) and
C     IC2(I).     Assign it to IC1(I).
C
      DO 50 I = 1, M
        IC1(I) = 1
        IC2(I) = 2
        DO 10 IL = 1, 2
          DT(IL) = ZERO
          DO 10 J = 1, N
            DA = A(I,J) - C(IL,J)
            DT(IL) = DT(IL) + DA*DA
   10   CONTINUE
        IF (DT(1) .GT. DT(2)) THEN
          IC1(I) = 2
          IC2(I) = 1
          TEMP = DT(1)
          DT(1) = DT(2)
          DT(2) = TEMP
        END IF
        DO 50 L = 3, K
          DB = ZERO
          DO 30 J = 1, N
            DC = A(I,J) - C(L,J)
            DB = DB + DC*DC
            IF (DB .GE. DT(2)) GO TO 50
   30     CONTINUE
          IF (DB .LT. DT(1)) GO TO 40
          DT(2) = DB
          IC2(I) = L
          GO TO 50
   40     DT(2) = DT(1)
          IC2(I) = IC1(I)
          DT(1) = DB
          IC1(I) = L
   50 CONTINUE
C
C     Update cluster centres to be the average of points contained
C     within them.
C
      DO 70 L = 1, K
        NC(L) = 0
        DO 60 J = 1, N
```

```
      60   C(L,J) = ZERO
      70 CONTINUE
         DO 90 I = 1, M
           L = IC1(I)
           NC(L) = NC(L) + 1
           DO 80 J = 1, N
      80   C(L,J) = C(L,J) + A(I,J)
      90 CONTINUE
C
C     Check to see if there is any empty cluster at this stage
C
         DO 120 L = 1, K
           IF (NC(L) .EQ. 0) THEN
             IFAULT = 1
             RETURN
           END IF
           AA = NC(L)
           DO 110 J = 1, N
     110   C(L,J) = C(L,J) / AA
C
C     Initialize AN1, AN2, ITRAN & NCP
C     AN1(L) = NC(L) / (NC(L) - 1)
C     AN2(L) = NC(L) / (NC(L) + 1)
C     ITRAN(L) = 1 if cluster L is updated in the quick-transfer stage,
C              = 0 otherwise
C     In the optimal-transfer stage, NCP(L) stores the step at which
C     cluster L is last updated.
C     In the quick-transfer stage, NCP(L) stores the step at which
C     cluster L is last updated plus M.
C
           AN2(L) = AA / (AA + ONE)
           AN1(L) = BIG
           IF (AA .GT. ONE) AN1(L) = AA / (AA - ONE)
           ITRAN(L) = 1
           NCP(L) = -1
     120 CONTINUE
         INDX = 0
         DO 140 IJ = 1, ITER
C
C     In this stage, there is only one pass through the data.   Each
C     point is re-allocated, if necessary, to the cluster that will
C     induce the maximum reduction in within-cluster sum of squares.
C
           CALL OPTRA(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D,
     *          ITRAN, LIVE, INDX)
C
C     Stop if no transfer took place in the last M optimal transfer
C     steps.
C
           IF (INDX .EQ. M) GO TO 150
C
C     Each point is tested in turn to see if it should be re-allocated
C     to the cluster to which it is most likely to be transferred,
C     IC2(I), from its present cluster, IC1(I).   Loop through the
C     data until no further change is to take place.
C
           CALL QTRAN(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D,
     *          ITRAN, INDX)
C
C     If there are only two clusters, there is no need to re-enter the
C     optimal transfer stage.
```

```
C
        IF (K .EQ. 2) GO TO 150
C
C     NCP has to be set to 0 before entering OPTRA.
C
        DO 130 L = 1, K
  130   NCP(L) = 0
  140 CONTINUE
C
C     Since the specified number of iterations has been exceeded, set
C     IFAULT = 2.   This may indicate unforeseen looping.
C
      IFAULT = 2
C
C     Compute within-cluster sum of squares for each cluster.
C
  150 DO 160 L = 1, K
        WSS(L) = ZERO
        DO 160 J = 1, N
          C(L,J) = ZERO
  160 CONTINUE
      DO 170 I = 1, M
        II = IC1(I)
        DO 170 J = 1, N
          C(II,J) = C(II,J) + A(I,J)
  170 CONTINUE
      DO 190 J = 1, N
        DO 180 L = 1, K
  180   C(L,J) = C(L,J) / FLOAT(NC(L))
        DO 190 I = 1, M
          II = IC1(I)
          DA = A(I,J) - C(II,J)
          WSS(II) = WSS(II) + DA*DA
  190 CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE OPTRA(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D,
     *      ITRAN, LIVE, INDX)
C
C     ALGORITHM AS 136.1  APPL. STATIST. (1979) VOL.28, NO.1
C
C     This is the optimal transfer stage.
C
C     Each point is re-allocated, if necessary, to the cluster that
C     will induce a maximum reduction in the within-cluster sum of
C     squares.
C
      INTEGER IC1(M), IC2(M), NC(K), NCP(K), ITRAN(K), LIVE(K)
      REAL    A(M,N), D(M), C(K,N), AN1(K), AN2(K), ZERO, ONE
C
C     Define BIG to be a very large positive number.
C
      DATA BIG /1.0E30/, ZERO /0.0/, ONE/1.0/
C
C     If cluster L is updated in the last quick-transfer stage, it
C     belongs to the live set throughout this stage.   Otherwise, at
C     each step, it is not in the live set if it has not been updated
C     in the last M optimal transfer steps.
```

```
C
      DO 10 L = 1, K
        IF (ITRAN(L) .EQ. 1) LIVE(L) = M + 1
   10 CONTINUE
      DO 100 I = 1, M
        INDX = INDX + 1
        L1 = IC1(I)
        L2 = IC2(I)
        LL = L2
C
C     If point I is the only member of cluster L1, no transfer.
C
        IF (NC(L1) .EQ. 1) GO TO 90
C
C     If L1 has not yet been updated in this stage, no need to
C     re-compute D(I).
C
        IF (NCP(L1) .EQ. 0) GO TO 30
        DE = ZERO
        DO 20 J = 1, N
          DF = A(I,J) - C(L1,J)
          DE = DE + DF*DF
   20   CONTINUE
        D(I) = DE * AN1(L1)
C
C     Find the cluster with minimum R2.
C
   30   DA = ZERO
        DO 40 J = 1, N
          DB = A(I,J) - C(L2,J)
          DA = DA + DB*DB
   40   CONTINUE
        R2 = DA * AN2(L2)
        DO 60 L = 1, K
C
C     If I >= LIVE(L1), then L1 is not in the live set.   If this is
C     true, we only need to consider clusters that are in the live set
C     for possible transfer of point I.   Otherwise, we need to consider
C     all possible clusters.
C
          IF (I .GE. LIVE(L1) .AND. I .GE. LIVE(L) .OR. L .EQ. L1 .OR.
     *        L .EQ. LL) GO TO 60
          RR = R2 / AN2(L)
          DC = ZERO
          DO 50 J = 1, N
            DD = A(I,J) - C(L,J)
            DC = DC + DD*DD
            IF (DC .GE. RR) GO TO 60
   50     CONTINUE
          R2 = DC * AN2(L)
          L2 = L
   60     CONTINUE
          IF (R2 .LT. D(I)) GO TO 70
C
C     If no transfer is necessary, L2 is the new IC2(I).
C
          IC2(I) = L2
          GO TO 90
C
C     Update cluster centres, LIVE, NCP, AN1 & AN2 for clusters L1 and
C     L2, and update IC1(I) & IC2(I).
```

```
C
   70     INDX = 0
          LIVE(L1) = M + I
          LIVE(L2) = M + I
          NCP(L1) = I
          NCP(L2) = I
          AL1 = NC(L1)
          ALW = AL1 - ONE
          AL2 = NC(L2)
          ALT = AL2 + ONE
          DO 80 J = 1, N
            C(L1,J) = (C(L1,J) * AL1 - A(I,J)) / ALW
            C(L2,J) = (C(L2,J) * AL2 + A(I,J)) / ALT
   80     CONTINUE
          NC(L1) = NC(L1) - 1
          NC(L2) = NC(L2) + 1
          AN2(L1) = ALW / AL1
          AN1(L1) = BIG
          IF (ALW .GT. ONE) AN1(L1) = ALW / (ALW - ONE)
          AN1(L2) = ALT / AL2
          AN2(L2) = ALT / (ALT + ONE)
          IC1(I) = L2
          IC2(I) = L1
   90   CONTINUE
        IF (INDX .EQ. M) RETURN
  100 CONTINUE
      DO 110 L = 1, K
C
C     ITRAN(L) = 0 before entering QTRAN.   Also, LIVE(L) has to be
C     decreased by M before re-entering OPTRA.
C
        ITRAN(L) = 0
        LIVE(L) = LIVE(L) - M
  110 CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE QTRAN(A, M, N, C, K, IC1, IC2, NC, AN1, AN2, NCP, D,
     *    ITRAN, INDX)
C
C     ALGORITHM AS 136.2  APPL. STATIST. (1979) VOL.28, NO.1
C
C     This is the quick transfer stage.
C     IC1(I) is the cluster which point I belongs to.
C     IC2(I) is the cluster which point I is most likely to be
C         transferred to.
C     For each point I, IC1(I) & IC2(I) are switched, if necessary, to
C     reduce within-cluster sum of squares.  The cluster centres are
C     updated after each step.
C
      INTEGER IC1(M), IC2(M), NC(K), NCP(K), ITRAN(K)
      REAL    A(M,N), D(M), C(K,N), AN1(K), AN2(K), ZERO, ONE
C
C     Define BIG to be a very large positive number
C
      DATA BIG /1.0E30/, ZERO /0.0/, ONE /1.0/
C
C     In the optimal transfer stage, NCP(L) indicates the step at which
C     cluster L is last updated.   In the quick transfer stage, NCP(L)
```

```
C     is equal to the step at which cluster L is last updated plus M.
C
      ICOUN = 0
      ISTEP = 0
   10 DO 70 I = 1, M
        ICOUN = ICOUN + 1
        ISTEP = ISTEP + 1
        L1 = IC1(I)
        L2 = IC2(I)
C
C     If point I is the only member of cluster L1, no transfer.
C
        IF (NC(L1) .EQ. 1) GO TO 60
C
C     If ISTEP > NCP(L1), no need to re-compute distance from point I to
C     cluster L1.   Note that if cluster L1 is last updated exactly M
C     steps ago, we still need to compute the distance from point I to
C     cluster L1.
C
        IF (ISTEP .GT. NCP(L1)) GO TO 30
        DA = ZERO
        DO 20 J = 1, N
          DB = A(I,J) - C(L1,J)
          DA = DA + DB*DB
   20   CONTINUE
        D(I) = DA * AN1(L1)
C
C     If ISTEP >= both NCP(L1) & NCP(L2) there will be no transfer of
C     point I at this step.
C
   30   IF (ISTEP .GE. NCP(L1) .AND. ISTEP .GE. NCP(L2)) GO TO 60
        R2 = D(I) / AN2(L2)
        DD = ZERO
        DO 40 J = 1, N
          DE = A(I,J) - C(L2,J)
          DD = DD + DE*DE
          IF (DD .GE. R2) GO TO 60
   40   CONTINUE
C
C     Update cluster centres, NCP, NC, ITRAN, AN1 & AN2 for clusters
C     L1 & L2.   Also update IC1(I) & IC2(I).   Note that if any
C     updating occurs in this stage, INDX is set back to 0.
C
        ICOUN = 0
        INDX = 0
        ITRAN(L1) = 1
        ITRAN(L2) = 1
        NCP(L1) = ISTEP + M
        NCP(L2) = ISTEP + M
        AL1 = NC(L1)
        ALW = AL1 - ONE
        AL2 = NC(L2)
        ALT = AL2 + ONE
        DO 50 J = 1, N
          C(L1,J) = (C(L1,J) * AL1 - A(I,J)) / ALW
          C(L2,J) = (C(L2,J) * AL2 + A(I,J)) / ALT
   50   CONTINUE
        NC(L1) = NC(L1) - 1
        NC(L2) = NC(L2) + 1
        AN2(L1) = ALW / AL1
        AN1(L1) = BIG
```

```
          IF (ALW .GT. ONE) AN1(L1) = ALW / (ALW - ONE)
          AN1(L2) = ALT / AL2
          AN2(L2) = ALT / (ALT + ONE)
          IC1(I) = L2
          IC2(I) = L1
C
C     If no re-allocation took place in the last M steps, return.
C
   60   IF (ICOUN .EQ. M) RETURN
   70 CONTINUE
      GO TO 10
      END
```

```
      SUBROUTINE EM(N, X1, X2, P, XMEAN, XSIGMA, E1, E2, MAXITS,
     *  COV, NOBS, K, IFAULT)
C
C       ALGORITHM AS 138 APPL. STATIST. (1979) VOL.28, NO.2
C
C       COMPUTE THE MAXIMUM LIKELIHOOD ESTIMATES OF THE MEAN
C       AND STANDARD DEVIATION OF A SINGLE NORMAL POPULATION,
C       THE DATA MAY CONTAIN CENSORED OR CONFINED OBSERVATIONS.
C
      DIMENSION X1(N), X2(N), COV(2, 2)
      INTEGER P(N), NOBS(4)
C
      DATA C /0.39894228/, ONEPLS /1.0001/, TOL /0.00001/,
     *  TOLL /0.00001/
C
      IFAULT = -2
      IF (N .LT. 2) RETURN
C
C       INITIALIZE COUNTERS
C
      K = 0
      SUM = 0.0
      SUM2 = 0.0
      SUMG = 0.0
      SUMG2 = 0.0
      IP = 0
      IQ = 0
      IR = 0
      IS = 0
C
C       EXACTLY SPECIFIED OBSERVATIONS ARE REMOVED,
C       THE REMAINING DATA PACKED INTO FIRST PART OF ARRAY X
C
      IFAULT = -4
      DO 200 I = 1, N
      IPT = P(I)
      IF (IPT .EQ. 0) GOTO 100
      IF (IPT .EQ. 2 .AND. ABS(X1(I) - X2(I)) .LE. ABS(X1(I) * TOLL))
     *  GOTO 100
C
C       OBSERVATION NOT EXACTLY SPECIFIED
C
      IS = IS + 1
      P(IS) = IPT
      X1(IS) = X1(I)
C
C       HANDLE GROUPED DATA
C
      IF (IPT .NE. 2) GOTO 50
      IQ = IQ + 1
      IF (X1(I) .GT. X2(I)) RETURN
      X2(IS) = X2(I)
      XTEMP = 0.5 * (X1(I) + X2(I))
      SUMG = SUMG + XTEMP
      SUMG2 = SUMG2 + XTEMP ** 2
      GOTO 200
C       ACCUMULATE NUMBER OF OBSERVATIONS CENSORED ON THE RIGHT
C
50    IF (IPT .EQ. 1) IR = IR + 1
      GOTO 200
C
```

```
C         HANDLE EXACTLY-SPECIFIED OBSERVATIONS
C
100     IP = IP + 1
        XTEMP = X1(I)
        SUM = SUM + XTEMP
        SUM2 = SUM2 + XTEMP ** 2
200     CONTINUE
C
C         INITIAL PASS THROUGH DATA COMPLETED
C
        NOBS(1) = IP
        NOBS(2) = IR
        NOBS(3) = N - IP - IR - IQ
        NOBS(4) = IQ
        RIM = IP + IQ
        IF (IP .EQ. N) GOTO 230
        IF (XSIGMA .GT. 0.0) GOTO 350
        IF (RIM .GT. ONEPLS) GOTO 250
C
C         AT MOST ONE OBSERVATION HAS BEEN EXACTLY
C         SPECIFIED OR CONFINED
C
        XMEAN = 1.0
        XSIGMA = 1.0
        GOTO 350
C
C         ALL OBSERVATIONS EXACTLY SPECIFIED
C
230     XMEAN = SUM / RIM
        XSIGMA = SQRT((SUM2 - RIM * XMEAN ** 2) / RIM)
        COV(1, 1) = XSIGMA ** 2 / RIM
        COV(2, 2) = COV(1, 1) * 0.5
        COV(1, 2) = 0.0
        COV(2, 1) = 0.0
C
C         NORMAL RETURN
C
240     IFAULT = 0
        RETURN
C
C         OBTAIN INITIAL ESTIMATES
C
250     XMEAN = (SUM + SUMG) / RIM
        XSIGMA = SQRT((SUM2 + SUMG2 - RIM * XMEAN ** 2) / RIM)
C
C         INITIALIZE BEFORE STARTING FIRST ITERATION
C
350     RP = IP
        RN = N
C
C         START OF ITERATION CYCLE,
C         ESTIMATE CONDITIONAL EXPECTATION OF CONFINED AND
C         CENSORED OBSERVATIONS
C
        IFAULT = -3
400     TS = SUM
        SUMG2 = SUM2
        TD = RP
        DO 610 I = 1, IS
        YS = (X1(I) - XMEAN) / XSIGMA
        IF (P(I) - 1) 500, 450, 550
```

```
C
C         OBSERVATION CENSORED ON THE RIGHT
C
450     CALL RMILLS(YS, F, TOL)
        W = XMEAN + XSIGMA * F
        TD = TD + F * (F - YS)
        GOTO 600
C
C         OBSERVATION CENSORED ON THE LEFT
C
500     CALL RMILLS(-YS, F, TOL)
        W = XMEAN - XSIGMA * F
        TD = TD + F * (F + YS)
        GOTO 600
C
C         CONFINED OBSERVATION.
C         USE MILLS RATIO RECIPROCAL TO COMPUTE PROBABILITY
C         INTEGRALS THAT ARE REQUIRED,
C         AS IN ORIGINAL ALGORITHM ASSUMING X1(I) IS
C         NEVER GREATER THAN X2(I) FOR CONFINED OBSERVATIONS
C
550     YN = EXP(-0.5 * YS ** 2) * C
        CALL RMILLS(YS, F, TOL)
        YQ = YN / F
        YSU = (X2(I) - XMEAN) / XSIGMA
        YNU = EXP(-0.5 * YSU ** 2) * C
        CALL RMILLS(YSU, FU, TOL)
        YQU = YNU / FU
        YD = YQ - YQU
C
C         IF INTEGRAL NOT EQUAL TO ZERO, CARRY ON
C
        IF (YD .LT. TOLL) RETURN
        A = (YN - YNU) / YD
        W = XMEAN + XSIGMA * A
        TD = TD + (A ** 2 + (YSU * YNU - YS * YN) / YD)
C
600     TS = TS + W
        SUMG2 = SUMG2 + W ** 2
610     CONTINUE
C
C         CALCULATE NEW ESTIMATES
C
        XNEW = TS / RN
        YNEW = SQRT((SUMG2 + RN * XMEAN ** 2 - 2.0 * TS * XMEAN) / TD)
        K = K + 1
        IF (ABS(XNEW - XMEAN) .LT. E1 .AND. ABS(YNEW - XSIGMA) .LT. E2)
     *  GOTO 700
        IF (K .GE. MAXITS) GOTO 650
        XMEAN = XNEW
        XSIGMA = YNEW
        GOTO 400
C
C         MAXIMUM NUMBER OF ITERATIONS EXCEEDED
C
650     IFAULT = -1
        COV(1, 1) = 0.0
        COV(2, 2) = 0.0
        COV(1, 2) = XNEW - XMEAN
        COV(2, 1) = YNEW - XSIGMA
        RETURN
```

```
C
C         CONVERGENCE OBTAINED
C
700     XMEAN = XNEW
        XSIGMA = YNEW
        XSIG2 = XSIGMA ** 2
C
C         CALCULATE VARIANCE-COVARIANCE MATRIX
C
        X11 = RP
        X12 = (SUM - RP * XMEAN) / XSIGMA
        X22 = RP + (SUM2 + RP * XMEAN ** 2 - 2.0 * SUM * XMEAN) / XSIG2
        DO 800 I = 1, IS
        YS = (X1(I) - XMEAN) / XSIGMA
        IF (P(I) - 1) 740, 710, 770
710     CALL RMILLS(YS, F, TOL)
C
C         OBSERVATION CENSORED ON THE RIGHT
C
        FL = F * (F - YS)
730     X11 = X11 + FL
        FL = FL * YS
        X12 = X12 + FL
        FL = FL * YS
        X22 = X22 + FL
        GOTO 800
C
740     CALL RMILLS(-YS, F, TOL)
C
C         OBSERVATION CENSORED ON THE LEFT
C
        FL = F * (F + YS)
        GOTO 730
C
770     CALL RMILLS(YS, F, TOL)
C
C         OBSERVATION CONFINED BETWEEN 2 FINITE LIMITS
C
        YN = EXP(-0.5 * YS ** 2) * C
        YQ = YN / F
        YSU = (X2(I) - XMEAN) / XSIGMA
        CALL RMILLS(YSU, FU, TOL)
        YNU = EXP(-0.5 * YSU ** 2) * C
        YQU = YNU / FU
        YD = YQ - YQU
        A = (YN - YNU) / YD
        B = (YNU * YSU - YN * YS) / YD
        X11 = X11 + A ** 2 + B
        B1 = (YS ** 2 * YN - YSU ** 2 * YNU) / YD
        X12 = X12 - A * B - B1
        B1 = (YS ** 3 * YN - YSU ** 3 * YNU) / YD
        X22 = X22 - B1 + B ** 2
800     CONTINUE
        CONST = XSIG2 / (X11 * X22 - X12 * X12)
        COV(1, 1) = CONST * X22
        COV(2, 2) = CONST * X11
        COV(1, 2) = -CONST * X12
        COV(2, 1) = COV(1, 2)
        GOTO 240
        END
C
```

```
      SUBROUTINE RMILLS(X, FUNC, TOL)
C
C       ALGORITHM AS 138.1 APPL. STATIST. (1979) VOL.28 NO.2
C
C       COMPUTE THE RECIPROCAL OF MILLS RATIO
C
      DATA FPI /1.2533141/, FPII /0.7978846/
C
      FUNC = 0.0
      IF (X .LT. -10.0) RETURN
      FUNC = FPII
      Y = ABS(X)
      IF (Y .LT. 0.000001) RETURN
      SGN = 1.0
      IF (X .LT. 0.0) SGN = -1.0
      IF (Y .GT. 2.0) GOTO 100
      S = 0.0
      A = 1.0
      T = Y
      R = Y
      B = Y ** 2
40    A = A + 2.0
      S = T
      R = R * B / A
      T = T + R
      IF (R .GT. TOL) GOTO 40
      FUNC = 1.0 / (FPI * EXP(0.5 * B) - SGN * T)
      RETURN
100   A = 2.0
      B1 = Y
      S = Y
      A1 = Y ** 2 + 1.0
      A2 = Y * (A1 + 2.0)
      B2 = A1 + 1.0
      T = A2 / B2
140   A = A + 1.0
      A0 = A1
      A1 = A2
      A2 = Y * A1 + A * A0
      B0 = B1
      B1 = B2
      B2 = Y * B1 + A * B0
      R = S
      S = T
      T = A2 / B2
      IF (T - R .GT. TOL .OR. T - S .GT. TOL) GOTO 140
      FUNC = T
      IF (SGN .LT. 0.0) FUNC =
     *  T / (2.0 * FPI * EXP(0.5 * Y ** 2) * T - 1.0)
      RETURN
      END
```

```
      program t_as139
c
c     Test AS139
c
      IMPLICIT NONE
      INTEGER n, p(100), m, mplone, rowx, colx, lenw, lenwrk, maxits,
     *        ifault, i, i1, j
      PARAMETER (n=100, m=5, mplone=m+1, rowx=n, colx=m, lenw=mplone+n,
     *        lenwrk=m*n, maxits=25)
      DOUBLE PRECISION y, y1(n), y2(n), x(rowx, colx), w(lenw),
     *        vcov(lenwrk), work(lenwrk), alpha(mplone), tol(mplone),
     *        temp, se(m)
      REAL rand, randn
      INTEGER IX, IY, IZ
      COMMON /RANDC/ IX, IY, IZ
c
c     Generate artificial data
c
      WRITE(*, *)'Enter 3 integers for random number seeds: '
      READ(*, *) ix, iy, iz

      DO i = 1, n
        x(i, 1) = 1.0
        x(i, 2) = i/10
        x(i, 3) = MOD(i, 10)
        x(i, 4) = x(i, 2) + rand() - 0.5
        x(i, 5) = x(i, 3) + rand() - 0.5
        y = x(i, 1) + x(i, 2) + x(i, 3) + x(i, 4) + x(i, 5) + randn()

        temp = rand()
        IF (temp .LT. 0.1) THEN
          p(i) = 1
          y1(i) = INT(y)
        ELSE IF (temp .LT. 0.2) THEN
          p(i) = -1
          y1(i) = INT(y) + 1
        ELSE IF (temp .LT. 0.3) THEN
          p(i) = 2
          y1(i) = INT(y)
          y2(i) = INT(y) + 1
        ELSE
          p(i) = 0
          y1(i) = y
        END IF
      END DO

      DO i = 1, mplone
        tol(i) = 1.d-06
      END DO

      alpha(mplone) = -1.0             ! No initial estimate for sigma

      CALL REGRES(N, Y1, Y2, P, MPLONE, X, ROWX, COLX, W, LENW,
     *    VCOV, WORK, LENWRK, ALPHA, TOL, MAXITS, IFAULT)

      IF (ifault .LT. 0) THEN
        WRITE(*, *)'IFAULT = ', ifault
        STOP
      ELSE
        WRITE(*, *)'No. of iterations = ', ifault
      END IF
```

```
      WRITE(*, 900) (alpha(i), i=1,m)
  900 FORMAT(1x, 'Regression coefficients:'/ 1x, 5f10.4)
      DO i = 1, m
        se(i) = SQRT( vcov((i-1)*mplone + i) )
      END DO
      WRITE(*, 910) (se(i), i=1,m)
  910 FORMAT(1x, 'Estimated standard errors:'/ 1x, 5f10.4)
      WRITE(*, 920) alpha(mplone)
  920 FORMAT(1x, 'Residual variance estimate = ', f10.4)
      WRITE(*, *) 'Variance-covariance matrix:-'
      i1 = 1
      DO i = 1, m
        WRITE(*, '(1x, 7f10.4)') (vcov(j),j=i1,i1+m-1)
        i1 = i1 + mplone
      END DO
      end


      REAL FUNCTION RAND()
C
C     The Wichmann & Hill random number generator
C     Algorithm AS183, Appl. Statist., 31, 188-190, 1982.
C     The cycle length is 6.95E+12.
C     This random number generator is very slow compared with most
C     others, but it is dependable, and the results are reproducible.
C
      INTEGER IX, IY, IZ
      COMMON /RANDC/ IX, IY, IZ
C
C     Initialize IX, IY & IZ if necessary
C
      IF(IX .LE. 0) IX = 777
      IF(IY .LE. 0) IY = 777
      IF(IZ .LE. 0) IZ = 777
C
   10 IX = MOD(171*IX, 30269)
      IY = MOD(172*IY, 30307)
      IZ = MOD(170*IZ, 30323)
      RAND = MOD(FLOAT(IX)/30269. + FLOAT(IY)/30307. +
     1       FLOAT(IZ)/30323. , 1.0)
      RETURN
      END


      REAL FUNCTION RANDN()
C
C      ALGORITHM 712, COLLECTED ALGORITHMS FROM ACM.
C      THIS WORK PUBLISHED IN TRANSACTIONS ON MATHEMATICAL SOFTWARE,
C      VOL. 18, NO. 4, DECEMBER, 1992, PP. 434-435.
C
C  The function RANDN() returns a normally distributed pseudo-random
C  number with zero mean and unit variance.  Calls are made to a
C  function subprogram RAND() which must return independent random
C  numbers uniform in the interval (0,1).
C
C  The algorithm uses the ratio of uniforms method of A.J. Kinderman
C  and J.F. Monahan augmented with quadratic bounding curves.
C
      DATA S,T,A,B / 0.449871, -0.386595, 0.19600, 0.25472/
      DATA R1,R2/ 0.27597, 0.27846/
```

```
C
C  Generate P = (u,v) uniform in rectangle enclosing acceptance region
 50   U = RAND()
      V = RAND()
      V = 1.7156 * (V - 0.5)
C  Evaluate the quadratic form
      X = U - S
      Y = ABS(V) - T
      Q = X**2 + Y*(A*Y - B*X)
C  Accept P if inside inner ellipse
      IF (Q .LT. R1) GO TO 100
C  Reject P if outside outer ellipse
      IF (Q .GT. R2) GO TO 50
C  Reject P if outside acceptance region
      IF (V**2 .GT. -4.0*ALOG(U)*U**2) GO TO 50
C  Return ratio of P's coordinates as the normal deviate
 100  RANDN = V/U
      RETURN
      END



      SUBROUTINE REGRES(N, Y1, Y2, P, MPLONE, X, ROWX, COLX, W, LENW,
     *    VCOV, WORK, LENWRK, ALPHA, TOL, MAXITS, IFAULT)
C***** NOTE: this routine uses the auxiliary routine
C***** AS7  with the modified argument lists suggested by Freeman
C***** Vol 31 No 3 p336--339.
C
C  as r31 Vol 29 No 2 1980 p228 -- incorporated
C  as r32 Vol 30 No 1 1981 p105 -- incorporated
C  as r91 Vol 42 No 3 1993 p583 -- incorporated
C
C       ALGORITHM AS139 APPL. STATIST. (1979) VOL.28, NO.2
C
C       COMPUTE MAXIMUM LIKELIHOOD ESTIMATES
C       FROM A LINEAR MODEL WITH NORMAL HETEROGENOUS VARIANCE.
C       THE DESIGN MATRIX MUST BE NON-SINGULAR.  THE DEPENDENT
C       VARIABLE MAY INCLUDE OBSERVATIONS CENSORED IN EITHER TAIL
C       AND/OR OBSERVATIONS CONFINED BETWEEN FINITE LIMITS.
C
C     Auxiliary routine required: RMILLS which is the last routine in
C     AS138.
C
      IMPLICIT NONE

      INTEGER N, MPLONE, ROWX, COLX, P(N), LENW, LENWRK, MAXITS, IFAULT
      DOUBLE PRECISION X(ROWX,COLX), TOL(MPLONE), Y1(N), Y2(N),
     *    ALPHA(MPLONE)
      DOUBLE PRECISION VCOV(LENWRK), WORK(LENWRK), W(LENW)
C
C     Local variables
C
      INTEGER M, I, J, K, II, NDIMC, NUL, JJ, IIK, IJ, IPT, NJ, IIJ,
     *      JJI, KK
      DOUBLE PRECISION QLIMIT, RLIMIT, C, ZERO, HALF, ONE
      PARAMETER (QLIMIT = 0.00001D0, RLIMIT=0.00001D0, C=0.39894228D0)
      PARAMETER (zero=0.d0, half=0.5D0, one=1.D0)
      DOUBLE PRECISION TEMP, SUM2, DEMP, R, R2, XSIG, TD, YMEAN, F, A,
     *      FU, YN, YQ, TMPU, YNU, YQU, TINT, TEMP2, YS, YSU, B
C
C       INITIALIZATION
```

```
C
      M = MPLONE-1
C
C        CHECK ARRAY SIZES, ETC
C
      IFAULT = -7
      IF(ROWX.LT.N) RETURN
      IFAULT = -8
      IF(COLX.LT.M) RETURN
      IFAULT = -9
      IF(LENW.LT.(MPLONE+N)) RETURN
      IFAULT = -10
      IF(LENWRK.LT.(M*N)) RETURN
C
C        COMPUTE X'X IN LOWER TRIANGULAR FORM
C
      II = 0
      DO 53 I = 1, M
        DO 50 J = 1, I
          TEMP = ZERO
          DO 40 K = 1, N
   40     TEMP = TEMP + X(K,I)*X(K,J)
          II = II + 1
          VCOV(II) = TEMP
   50   CONTINUE
   53 CONTINUE
      NDIMC = M*(M+1)/2
      CALL SYMINV(VCOV, M, NDIMC, WORK, W, NUL, IFAULT)
      IF(IFAULT.NE.0) GOTO 60
      IF(NUL.EQ.0) GOTO 70
   60 VCOV(2) = IFAULT
      VCOV(1) = NUL
      IFAULT = -5
      RETURN
C
C        MATRIX NON-SINGULAR AND INVERSE OBTAINED
C        COMPUTE (X'X)INVERSE*X
C        FOLLOWING SCHEME USED TO REDUCE NUMBER OF STORAGE ARRAYS
C        NEEDED.   EXPAND FROM TRIANGULAR TO SQUARE MATRIX
C
   70 CALL UNPACK(WORK, M, LENWRK)
C
C        DO MULTIPLICATION - ONE ROW AT A TIME - STARTING
C        WITH THE LAST ONE
C
      JJ = N*M
      II = M*M
      DO 220 I = 1, M
        II = II - M
        DO 200 J = 1, N
          TEMP = ZERO
          DO 170 K = 1, M
            IIK = II + K
            TEMP = TEMP + WORK(IIK)*X(J,K)
  170     CONTINUE
          W(J) = TEMP
  200   CONTINUE
        DO 210 J = 1, N
          IJ = N+1-J
          WORK(JJ) = W(IJ)
          JJ = JJ-1
```

```
  210   CONTINUE
  220 CONTINUE
C
      XSIG = ALPHA(MPLONE)
      IF(XSIG.GT.ZERO) GOTO 500
C
C        NO ACCEPTABLE INITIAL VALUE FOR SIGMA HAS BEEN INPUT,
C        OBTAIN INITIAL ESTIMATES FROM EXACTLY SPECIFIED
C        OBSERVATIONS ONLY (ALTHOUGH MATRIX BASED ON ALL
C        OBSERVATIONS) AND CONFINED OBSERVATIONS
C
      II = -N
      DO 300 I = 1, M
        II = II + N
        TEMP = ZERO
        DO 280 J = 1, N
          IIJ = II + J
          IPT = P(J)
          IF(IPT.EQ.0) GOTO 270
          IF(IPT.EQ.2) TEMP = TEMP + WORK(IIJ)*(Y1(J) + Y2(J))*HALF
          GOTO 280
  270     TEMP = TEMP + WORK(IIJ)*Y1(J)
  280   CONTINUE
        ALPHA(I) = TEMP
  300 CONTINUE
C
C        CALCULATE INITIAL ESTIMATE OF SIGMA
C
      SUM2 = ZERO
      TEMP = ZERO
      DO 350 I = 1, N
        IPT = P(I)
        IF(IABS(IPT).EQ.1) GOTO 350
        DEMP = Y1(I)
        IF(IPT.EQ.2) DEMP = (DEMP + Y2(I))*HALF
        DO 320 J = 1, M
  320   DEMP = DEMP - ALPHA(J)*X(I, J)
        SUM2 = SUM2 + DEMP**2
        TEMP = TEMP + ONE
  350 CONTINUE
      XSIG = SQRT(SUM2/TEMP)
C
C        COMPUTE SOME CONSTANTS NEEDED THROUGHOUT
C
  500 R = ZERO
      R2 = ZERO
      IFAULT = -2
      DO 600 I = 1, N
        IPT = P(I)
        IF(IPT.EQ.0) GOTO 550
        IF(IPT.EQ.2 .AND. ABS(Y1(I)-Y2(I)).LE. QLIMIT*ABS(Y1(I)))
     *                    GOTO 540
        IF(IPT.NE.2) GOTO 600
        R2 = R2 + ONE
        IF(Y1(I).LT.Y2(I)) GOTO 600
        RETURN
  540   P(I) = 0
  550   R = R + ONE
        W(I) = Y1(I)
  600 CONTINUE
      I = R + R2 + 0.01
```

```
      IFAULT = -4
      IF(I.LT.MPLONE) RETURN
      IFAULT = 0
C
C        START OF ITERATION PROCEDURE
C
  620 TD = R
      SUM2 = ZERO
C
C        COMPLETE W-VECTOR
C
      DO 1000 I = 1, N
        IPT = P(I)
        YMEAN = ZERO
        DO 650 J = 1, M
  650   YMEAN = YMEAN + ALPHA(J)*X(I,J)
        IF(IPT.EQ.0) GOTO 990
C
C         OBSERVATION NOT EXACTLY SPECIFIED
C
        TEMP = (Y1(I) - YMEAN)/XSIG
        IF(IPT-1)750, 700, 800
C
C         OBSERVATION CENSORED FROM ABOVE - LOWER BOUND KNOWN
C
  700   CALL RMILLS(TEMP, F, RLIMIT)
        W(I) = YMEAN + XSIG*F
        TD = TD + F*(F-TEMP)
        GOTO 990
C
C         OBSERVATION CENSORED FROM BELOW - UPPER BOUND KNOWN
C
  750   CALL RMILLS(-TEMP, F, RLIMIT)
        W(I) = YMEAN - XSIG*F
        TD = TD + F*(F+TEMP)
        GOTO 990
C
C         OBSERVATION CONFINED TO LIE BETWEEN TWO FINITE LIMITS
C
  800   YN = EXP(-HALF*TEMP**2)*C
        CALL RMILLS(TEMP, F, RLIMIT)
        YQ = YN/F
        TMPU = (Y2(I) - YMEAN)/XSIG
        YNU = EXP(-HALF*TMPU**2)*C
        CALL RMILLS(TMPU, FU, RLIMIT)
        YQU = YNU/FU
        TINT = YQ - YQU
        IF(TINT.GE.QLIMIT) GOTO 820
C
C         AFTER STANDARDIZING, UPPER AND LOWER LIMITS RESULT IN
C         SAME PROBABILITY INTEGRAL
C
        IFAULT = -3
        RETURN
  820   A = (YN - YNU)/TINT
        W(I) = YMEAN + XSIG*A
        TD = TD + (A**2 + (TMPU*YNU - TEMP*YN)/TINT)
C
C         CALCULATE RESIDUAL SUM OF SQUARES
C
  990   SUM2 = SUM2 + (W(I) - YMEAN)**2
```

```
 1000 CONTINUE
C
C        UPDATE PARAMETER ESTIMATES - STORE IN END OF W-VECTOR
C
      JJ = -N
      DO 1200 J = 1, M
        JJ = JJ + N
        TEMP = ZERO
        DO 1100 I = 1, N
          JJI = JJ + I
          TEMP = TEMP + WORK(JJI)*W(I)
 1100   CONTINUE
        NJ = N + J
        W(NJ) = TEMP
 1200 CONTINUE
      NJ = N + MPLONE
      W(NJ) = SQRT(SUM2/TD)
C
C        TEST FOR CONVERGENCE
C
      DO 1300 J = 1, MPLONE
        NJ = N + J
        IF(ABS(ALPHA(J)-W(NJ)).GE.TOL(J)) GOTO 1400
 1300 CONTINUE
C
C        IF WE REACH HERE, CONVERGENCE OBTAINED
C
      IJ = IFAULT
      IFAULT = -1
C
C        UPDATE VALUES
C
 1400 DO 1450 J = 1, MPLONE
        NJ = N + J
        ALPHA(J) = W(NJ)
 1450 CONTINUE
      XSIG = ALPHA(MPLONE)
      IFAULT = IFAULT + 1
      IF(IFAULT.EQ.0) GOTO 1600
      IF(IFAULT.LE.MAXITS) GOTO 620
      IFAULT = -1
      RETURN
C
C        CONVERGENCE OBTAINED - COMPUTE VARIANCE-COVARIANCE
C        MATRIX. INITIALIZE WORK ARRAY
C
 1600 II = MPLONE*(MPLONE + 1)/2
      DO 1650 I = 1, II
 1650 WORK(I) = ZERO
      DO 2500 I = 1, N
        IPT = P(I)
        YS = Y1(I)
        DO 1680 J = 1, M
 1680   YS = YS - ALPHA(J)*X(I,J)
        YS = YS/XSIG
        JJ = 0
        IF(IPT.NE.0) GOTO 1900
C
C        EXACTLY SPECIFIED OBSERVATION
C
        DO 1750 K = 1, M
```

```
            DO 1720 J = 1, K
              JJ = JJ + 1
              WORK(JJ) = WORK(JJ) + X(I,K)*X(I,J)
 1720       CONTINUE
            KK = II - MPLONE + K
            WORK(KK) = WORK(KK) + YS*X(I,K)
 1750     CONTINUE
          WORK(II) = WORK(II) + ONE + YS**2
          GOTO 2500
 1900     IF(IPT-1) 2100, 2000, 2300
C
C          OBSERVATION CONSORED FROM ABOVE - LOWER BOUND KNOWN
C
 2000     CALL RMILLS(YS, F, RLIMIT)
          TEMP = F*(F - YS)
          GOTO 2150
C
C          OBSERVATION CONSORED FROM BELOW - UPPER BOUND KNOWN
C
 2100     CALL RMILLS(-YS, F, RLIMIT)
          TEMP = F*(F + YS)
C
C          ROUTINE FOR CENSORED OBSERVATIONS
C
 2150     DO 2190 K = 1, M
            DO 2170 J = 1, K
              JJ = JJ + 1
              WORK(JJ) = WORK(JJ) + X(I,J)*X(I,K)*TEMP
 2170       CONTINUE
            KK = II - MPLONE + K
            WORK(KK) = WORK(KK) + YS*X(I,K)*TEMP
 2190     CONTINUE
          WORK(II) = WORK(II) + YS**2*TEMP
          GOTO 2500
C
C          OBSERVATION CONFINED BETWEEN TWO FINITE LIMITS
C
 2300     YN = EXP(-HALF*YS**2)*C
          CALL RMILLS(YS, F, RLIMIT)
          YQ = YN/F
          YSU = YS + (Y2(I) - Y1(I))/XSIG
          CALL RMILLS(YSU, FU, RLIMIT)
          YNU = EXP(-HALF*YSU**2)*C
          YQU = YNU/FU
          TINT = YQ - YQU
          A = (YN - YNU)/TINT
          B = (YNU*YSU - YN*YS)/TINT
          TEMP = A**2 + B
          TEMP2 = A*B + (YS**2 * YN - YSU**2 * YNU) / TINT
          DO 2350 K = 1, M
            DO 2330 J = 1, K
              JJ = JJ + 1
              WORK(JJ) = WORK(JJ) + X(I,J)*X(I,K)*TEMP
 2330       CONTINUE
            KK = II - MPLONE + K
            WORK(KK) = WORK(KK) - X(I,K) * TEMP2
 2350     CONTINUE
          TEMP = (YS**3*YN - YSU**3*YNU)/TINT
          WORK(II) = WORK(II) - TEMP + B**2
 2500   CONTINUE
C
```

```
C        INVERT THE MATRIX
C
      CALL SYMINV(WORK, MPLONE, II, VCOV, W, NUL, IFAULT)
      IF(IFAULT.EQ.0 .AND. NUL.EQ.0) GOTO 2550
      VCOV(2) = IFAULT
      VCOV(1) = NUL
      IFAULT = -6
      RETURN
C
C        RESTORE ITERATION COUNTER
C
 2550 IFAULT = IJ
C
C        MULTIPLY BY SIGMA-SQUARED
C
      TEMP = XSIG**2
      DO 2580 I = 1, II
 2580 VCOV(I) = VCOV(I)*TEMP
C
C        UNPACK THE MATRIX
C
      CALL UNPACK(VCOV, MPLONE, LENWRK)
      RETURN
      END
*************************************************************
      SUBROUTINE UNPACK(X, N, LENX)
C
C        ALGORITHM AS139.1 APPL. STATIST. (1979) VOL.28 NO.2
C
C        THIS SUBROUTINE EXPANDS A SYMMETRIC MATRIX STORED IN LOWER
C        TRIANGULAR FORM IN THE FIRST N*(N+1)/2 POSITIONS OF X
C        INTO A MATRIX USING THE FIRST N*N POSITIONS
C
C        LENX = LENGTH OF VECTOR X - MUST BE LESS THAN N*N
C
      INTEGER N, LENX
      DOUBLE PRECISION X(LENX)
C
C     Local variables
C
      INTEGER NSQ, II, JJ, I, IJ, KK, J

      NSQ = N*N
      II = NSQ
      JJ = N*(N+1)/2
C
C        STORE LAST ROW
C
      DO 10 I = 1, N
        X(II) = X(JJ)
        II = II-1
        JJ = JJ-1
   10 CONTINUE
      DO 80 I = 2, N
C
C        OBTAIN UPPER PART OF MATRIX FROM PART ALREADY SHIFTED
C
        IJ = I - 1
        KK = NSQ+1-I
        DO 50 J = 1, IJ
          X(II) = X(KK)
```

```
            II = II - 1
            KK = KK - N
   50   CONTINUE
C
C        OBTAIN LOWER PART OF MATRIX FROM
C        ORIGINAL TRIANGULAR STORAGE
C
         IJ = N - IJ
         DO 70 J = 1, IJ
           X(II) = X(JJ)
           II = II - 1
           JJ = JJ - 1
   70   CONTINUE
   80 CONTINUE
      RETURN
      END


      SUBROUTINE RMILLS(X, FUNC, TOL)
C
C       ALGORITHM AS 138.1 APPL. STATIST. (1979) VOL.28 NO.2
C
C       COMPUTE THE RECIPROCAL OF MILLS RATIO
C
      DOUBLE PRECISION X, FUNC, TOL
C
C     Local variables
C
      DOUBLE PRECISION Y, SGN, S, A, T, R, B, B1, A1, A2, B2, A0, B0
      DOUBLE PRECISION FPI, FPII, ZERO, HALF, ONE, TWO, TEN, SMALL
      DATA FPI /1.2533141D0/, FPII /0.7978846D0/, ZERO /0.D0/,
     *     HALF /0.5D0/, ONE /1.D0/, TWO /2.D0/, TEN /10.D0/,
     *     SMALL /0.000001D0/
C
      FUNC = ZERO
      IF (X .LT. -TEN) RETURN
      FUNC = FPII
      Y = ABS(X)
      IF (Y .LT. SMALL) RETURN
      SGN = ONE
      IF (X .LT. ZERO) SGN = -ONE
      IF (Y .GT. TWO) GOTO 100
      S = ZERO
      A = ONE
      T = Y
      R = Y
      B = Y ** 2
40    A = A + TWO
      S = T
      R = R * B / A
      T = T + R
      IF (R .GT. TOL) GOTO 40
      FUNC = ONE / (FPI * EXP(HALF * B) - SGN * T)
      RETURN
C
100   A = TWO
      B1 = Y
      S = Y
      A1 = Y ** 2 + ONE
      A2 = Y * (A1 + TWO)
      B2 = A1 + ONE
```

```
      T = A2 / B2
140   A = A + ONE
      A0 = A1
      A1 = A2
      A2 = Y * A1 + A * A0
      B0 = B1
      B1 = B2
      B2 = Y * B1 + A * B0
      R = S
      S = T
      T = A2 / B2
      IF (T - R .GT. TOL .OR. T - S .GT. TOL) GOTO 140
      FUNC = T
      IF (SGN .LT. ZERO) FUNC = T / (TWO*FPI*EXP(HALF * Y**2) * T - ONE)
      RETURN
      END



        subroutine syminv(a, n, nn, c, w, nullty, ifault)
c
c       Algorithm AS7, Applied Statistics, vol.17, 1968, p.198.
c
c       Forms in c( ) as lower triangle, a generalised inverse
c       of the positive semi-definite symmetric matrix a( )
c       order n, stored as lower triangle.
c
c       arguments:-
c       a()     = input, the symmetric matrix to be inverted, stored in
c                 lower triangular form
c       n       = input, order of the matrix
c       nn      = input, the size of the a and c arrays     n*(n+1)/2
c       c()     = output, the inverse of a (a generalized inverse if c is
c                 singular), also stored in lower triangular.
c                 c and a may occupy the same locations.
c       w()     = workspace, dimension at least n.
c       nullty  = output, the rank deficiency of a.
c       ifault  = output, error indicator
c                       = 1 if n < 1
c                       = 2 if a is not +ve semi-definite
c                       = 3 if nn < n*(n+1)/2
c                       = 0 otherwise
c
c*********************************************************************
c
        double precision a(nn), c(nn), w(n), x, zero, one
c
        data zero, one /0.0d0, 1.0d0/
c
c       cholesky factorization of a, result in c
c
        call chol(a, n, nn, c, nullty, ifault)
        if(ifault.ne.0) return
c
c       invert c & form the product (cinv)'*cinv, where cinv is the inverse
c       of c, row by row starting with the last row.
c       irow = the row number, ndiag = location of last element in the row.
c
        irow=n
        ndiag=nn
   10   l=ndiag
```

```
          if (c(ndiag) .eq. zero) goto 60
          do 20 i=irow,n
            w(i)=c(l)
            l=l+i
     20   continue
          icol=n
          jcol=nn
          mdiag=nn
     30   l=jcol
          x=zero
          if(icol.eq.irow) x=one/w(irow)
          k=n
     40   if(k.eq.irow) go to 50
          x=x-w(k)*c(l)
          k=k-1
          l=l-1
          if(l.gt.mdiag) l=l-k+1
          go to 40
     50   c(l)=x/w(irow)
          if(icol.eq.irow) go to 80
          mdiag=mdiag-icol
          icol=icol-1
          jcol=jcol-1
          go to 30
     60   do 70 j=irow,n
            c(l)=zero
            l=l+j
     70   continue
     80   ndiag=ndiag-irow
          irow=irow-1
          if(irow.ne.0) go to 10
          return
          end



        SUBROUTINE CHOL (A, N, NN, U, NULLTY, IFAULT)
C
C       Algorithm AS6, Applied Statistics, vol.17, (1968)
C
C       Given a symmetric matrix order n as lower triangle in a( )
C       calculates an upper triangle, u( ), such that uprime * u = a.
C       a must be positive semi-definite.  eta is set to multiplying
C       factor determining effective zero for pivot.
C
C       arguments:-
C       a()     = input, a +ve definite matrix stored in lower-triangula
C                 form.
C       n       = input, the order of a
C       nn      = input, the size of the a and u arrays      n*(n+1)/2
C       u()     = output, a lower triangular matrix such that u*u' = a.
C                 a & u may occupy the same locations.
C       nullty  = output, the rank deficiency of a.
C       ifault  = output, error indicator
C                       = 1 if n < 1
C                       = 2 if a is not +ve semi-definite
C                       = 3 if nn < n*(n+1)/2
C                       = 0 otherwise
C
C*********************************************************************
C
```

```
      DOUBLE PRECISION A(NN), U(NN), ETA, ETA2, X, W, ZERO
C
C       The value of eta will depend on the word-length of the
C       computer being used.  See introductory text.
C
      DATA ETA, ZERO/1.D-9, 0.0D0/
C
      IFAULT = 1
      IF (N.LE.0) RETURN
      IFAULT = 3
      IF (NN.LT.N*(N+1)/2) RETURN
      IFAULT = 2
      NULLTY = 0
      J = 1
      K = 0
      ETA2 = ETA*ETA
      II = 0
C
C       Factorize column by column, icol = column no.
C
      DO 80 ICOL = 1,N
        II = II+ICOL
        X = ETA2*A(II)
        L = 0
        KK = 0
C
C       IROW = row number within column ICOL
C
        DO 40 IROW = 1,ICOL
          KK = KK+IROW
          K = K+1
          W = A(K)
          M = J
          DO 10 I = 1,IROW
            L = L+1
            IF (I.EQ.IROW) GO TO 20
            W = W-U(L)*U(M)
            M = M+1
 10       CONTINUE
 20       IF (IROW.EQ.ICOL) GO TO 50
          IF (U(L).EQ.ZERO) GO TO 30
          U(K) = W/U(L)
          GO TO 40
 30       IF (W*W.GT.ABS(X*A(KK))) RETURN
          U(K) = ZERO
 40     CONTINUE
 50     IF (ABS(W).LE.ABS(ETA*A(K))) GO TO 60
        IF (W.LT.ZERO) RETURN
        U(K) = SQRT(W)
        GO TO 70
 60     U(K) = ZERO
        NULLTY = NULLTY+1
 70     J = J+ICOL
 80   CONTINUE
      IFAULT = 0
      END
```

```
      SUBROUTINE INIT(X, N, XLEN, NCLUS, MXCLUS, MXSIZE, ITYPE,
     *  CLUS, Y, SIZE, P, IOLD, INEW, IFAULT)
C
C       ALGORITHM AS 140.1 APPL. STATIST. (1979) VOL.28, NO.2
C
C       CONSTRUCT AN INITIAL PARTITION OF THE NODES
C
      INTEGER XLEN, FIRST, X(XLEN), Y(XLEN), CLUS(N), SIZE(MXCLUS),
     *  INEW(N), IOLD(N)
      REAL P(MXCLUS, MXCLUS)
      LOGICAL FLAG
C
C       CHECK INPUTS
C
      IFAULT = 1
      IF (NCLUS .LE. 1 .OR. NCLUS .GE. N .OR. NCLUS .GT. MXCLUS) RETURN
      IFAULT = 2
      IF (NCLUS * MXSIZE .LE. N) RETURN
      IFAULT = 3
      DO 1 I = 2, N
      L = I - 1
      IF (X(I) .LE. X(L)) RETURN
1     CONTINUE
      M = N + 1
      IF (X(1) .NE. M .OR. X(N) .GT. XLEN) RETURN
      IFAULT = 4
      DO 2 I = M, XLEN
      IF (X(I) .LE. 0 .OR. X(I) .GT. N) RETURN
2     CONTINUE
      IFAULT = 5
      DO 5 I = 1, N
      FIRST = X(I)
      L = I + 1
      LAST = X(L) - 1
      IF (I .EQ. N) LAST = XLEN
      IF (FIRST .EQ. LAST) GOTO 5
      JLAST = LAST - 1
      DO 4 J = FIRST, JLAST
      JTEST = X(J)
      KFIRST = J + 1
      DO 3 K = KFIRST, LAST
      IF (JTEST .EQ. X(K)) RETURN
3     CONTINUE
4     CONTINUE
5     CONTINUE
C
C       CONSTRUCT Y VECTOR.  THE Y VECTOR HOLDS THE ASSOCIATION
C       MATRIX BY COLUMNS RATHER THAN BY ROWS AS X DOES.
C
      DO 6 I = 1, N
      INEW(I) = 0
      Y(I) = N + 1
6     CONTINUE
      Y(M) = 0
      DO 7 I = 1, N
      FIRST = X(I)
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 7 J = FIRST, LAST
      KFIRST = X(J) + 1
      DO 7 K = KFIRST, N
```

```
          Y(K) = Y(K) + 1
    7     CONTINUE
          DO 8 I = 1, N
          FIRST = X(I)
          LAST = X(I + 1) - 1
          IF (I .EQ. N) LAST = XLEN
          DO 8 J = FIRST, LAST
          ITEMP = X(J)
          L = Y(ITEMP) + INEW(ITEMP)
          Y(L) = I
          INEW(ITEMP) = INEW(ITEMP) + 1
    8     CONTINUE
          IFAULT = 6
          DO 9 I = 2, N
          IF (Y(I) .LE. Y(I - 1)) RETURN
    9     CONTINUE
          IF (Y(1) .NE. M .OR. Y(N) .GT. XLEN) RETURN
          IF (ITYPE .EQ. 1) GOTO 33
          IFAULT = 0
    C
    C       APPROXIMATE FIRST EIGENVECTOR
    C
          DO 11 I = 1, N
    11    IOLD(I) = 1
          DO 13 ITER = 1, 8
          DO 12 I = 1, N
          LAST = X(I + 1) - 1
          IF (I .EQ. N) LAST = XLEN
          DO 12 J = FIRST, LAST
          ITEMP = X(J)
          INEW(I) = INEW(I) + IOLD(ITEMP)
    12    CONTINUE
          DO 13 I = 1, N
          IOLD(I) = INEW(I)
    13    CONTINUE
    C
    C       SORT BY FIRST EIGENVECTOR
    C
          DO 14 I = 1, N
    14    INEW(I) = I
          DO 16 I = 1, N
          FLAG = .TRUE.
          DO 15 J = 2, N
          L = J - 1
          ITEMP = INEW(L)
          M = INEW(J)
          IF (IOLD(M) .GE. IOLD(ITEMP)) GOTO 15
          FLAG = .FALSE.
          INEW(J) = INEW(L)
          INEW(L) = M
    15    CONTINUE
          IF (FLAG) GOTO 17
    16    CONTINUE
    C
    C       PARTITION INTO INITIAL CLUSTERS
    C
    17    KSTART = N / NCLUS
          DO 21 I = 1, NCLUS
    21    SIZE(I) = KSTART
          ILAST = MOD(N, NCLUS)
          IF (ILAST .EQ. 0) GOTO 31
```

```
      DO 22 I = 1, ILAST
22    SIZE(I) = SIZE(I) + 1
31    J = 1
      DO 32 I = 1, NCLUS
      KLAST = SIZE(I)
      DO 32 K = 1, KLAST
      ITEMP = INEW(J)
      CLUS(ITEMP) = I
      J = J + 1
32    CONTINUE
      GOTO 40
33    DO 34 I = 1, NCLUS
34    SIZE(I) = 0
      IFAULT = 7
      DO 35 I = 1, N
      J = CLUS(I)
      IF (J .LE. 0 .OR. J .GT. NCLUS) RETURN
      SIZE(J) = SIZE(J) + 1
35    CONTINUE
      IFAULT = 8
      DO 36 I = 1, NCLUS
      IF (SIZE(I) .LE. 0 .OR. SIZE(I) .GT. MXSIZE) RETURN
36    CONTINUE
      IFAULT = 0
C
C       SET UP P MATRIX, SUCCESS COUNTS
C
40    DO 41 I = 1, NCLUS
      DO 41 J = 1, NCLUS
41    P(I, J) = 0.0
      DO 42 I = 1, N
      FIRST = X(I)
      LAST = X(I + 1) - 1
      IF (I .EQ. N) LAST = XLEN
      DO 42 J = FIRST, LAST
      IF (X(J) .EQ. I) GOTO 42
      ITEMP = X(J)
      ITEMP = CLUS(ITEMP)
      ITEMP2 = CLUS(I)
      P(ITEMP2, ITEMP) = P(ITEMP2, ITEMP) + 1.0
42    CONTINUE
      RETURN
      END
C
      SUBROUTINE ALLOC(X, Y, N, XLEN, NCLUS, MXCLUS, MXSIZE,
     *  TH, MXS2, CLUS, SIZE, P, R1, R2, TLOG, IOLD, INEW)
C
C       ALGORITHM AS 140.2 APPL. STATIST. (1979) VOL.28, NO.2
C
C       FROM AN INITIAL PARTITION OF THE NODES OF A GRAPH,
C       REALLOCATE NODES TO CLUSTERS TO FIND A LOCALLY
C       MAXIMUM LIKELIHOOD PARTITION
C
      INTEGER XLEN, FIRST, X(XLEN), Y(XLEN), CLUS(N),
     *  SIZE(MXCLUS), IOLD(N), INEW(N)
      REAL P(MXCLUS, MXCLUS), TLOG(MXS2), R1(MXCLUS), R2(MXCLUS)
      LOGICAL GLOBAL
C
C       INITIALIZE TLOG, INEW, IOLD, PASS.
C
      DO 1 I = 1, MXS2
```

```
1       TLOG(I) = FLOAT(I) * ALOG(FLOAT(I))
2       DO 3 I = 1, NCLUS
        INEW(I) = 0
        IOLD(I) = 1
3       CONTINUE
        GLOBAL = .TRUE.
C
C         MOVE A NODE TO A NEW CLUSTER IF THE MOVE INCREASES
C         THE LIKELIHOOD.
C         ONLY CHECK MOVES IF ONE OF THE CLUSTERS HAS IOLD = 1.
C
4       DO 59 ITER = 1, N
        NBEST = CLUS(ITER)
        BTEST = TH
C
C         SET UP ARRAY OF ASSOCIATIONS (R) FOR THIS NODE
C
        DO 10 I = 1, NCLUS
        R1(I) = 0.0
        R2(I) = 0.0
10      CONTINUE
        FIRST = X(ITER)
        L = ITER + 1
        LAST = X(L) - 1
        IF (ITER .EQ. N) LAST = XLEN
        DO 11 I = FIRST, LAST
        IF (X(I) .EQ. ITER) GOTO 11
        ITEMP = X(I)
        ITEMP = CLUS(ITEMP)
        R1(ITEMP) = R1(ITEMP) + 1.0
11      CONTINUE
        FIRST = Y(ITER)
        LAST = Y(L) - 1
        IF (ITER .EQ. N) LAST = XLEN
        DO 12 I = FIRST, LAST
        IF (Y(I) .EQ. ITER) GOTO 12
        ITEMP = Y(I)
        ITEMP = CLUS(ITEMP)
        R2(ITEMP) = R2(ITEMP) + 1.0
12      CONTINUE
C
C         CHECK EACH CLUSTER FOR AN INCREASE IN LIKELIHOOD
C
        L = CLUS(ITER)
        DO 49 M = 1, NCLUS
        IF (L .EQ. M .OR. SIZE(M) .GE. MXSIZE .OR.
     *  IOLD(L) + IOLD(M) .EQ. 0) GOTO 49
        TEST = 0.0
        DO 20 J = 1, NCLUS
        IF (J .EQ. L .OR. J .EQ. M) GOTO 20
        IF (P(L, J) .GT. 0.0) TEST = TEST + XLIKE(P(L, J), R1(J), SIZE(L),
     *  SIZE(J), SIZE(L) - 1, SIZE(J), 1.0, TLOG, MXS2)
        IF (P(M, J) .GT. 0.0 .OR. R1(J) .GT. 0.0) TEST = TEST +
     *  XLIKE(P(M, J), R1(J), SIZE(M), SIZE(J), SIZE(M) + 1, SIZE(J),
     *  -1.0, TLOG, MXS2)
        IF (P(J, L) .GT. 0.0) TEST = TEST + XLIKE(P(J, L), R2(J), SIZE(L),
     *  SIZE(J), SIZE(L) - 1, SIZE(J), 1.0, TLOG, MXS2)
        IF (P(J, M) .GT. 0.0 .OR. R2(J) .GT. 0.0) TEST = TEST +
     *  XLIKE(P(J, M), R2(J), SIZE(M), SIZE(J), SIZE(M) + 1, SIZE(J),
     *  -1.0, TLOG, MXS2)
20      CONTINUE
```

```
         TEST = TEST + XLIKE(P(L, L), R1(L) + R2(L) , SIZE(L) - 1, SIZE(L),
     *   SIZE(L) - 1, SIZE(L) - 2, 1.0, TLOG, MXS2) +
     *   XLIKE(P(L, M), R1(M) - R2(L) , SIZE(L), SIZE(M), SIZE(L) - 1,
     *   SIZE(M) + 1, 1.0, TLOG, MXS2) +
     *   XLIKE(P(M, L), R2(M) - R1(L) , SIZE(L), SIZE(M), SIZE(L) - 1,
     *   SIZE(M) + 1, -1.0, TLOG, MXS2) +
     *   XLIKE(P(M, M), R1(M) + R2(M) , SIZE(M) - 1, SIZE(M),
     *   SIZE(M) + 1, SIZE(M), -1.0, TLOG, MXS2)
         IF (TEST .LE. BTEST) GOTO 49
         BTEST = TEST
         NBEST = M
49       CONTINUE
C
C          MOVE TO BEST CLUSTER
C
         IF (NBEST .EQ. L) GOTO 59
         M = NBEST
         DO 50 II = 1, NCLUS
         P(L, II) = P(L, II) - R1(II)
         P(M, II) = P(M, II) + R1(II)
         P(II, L) = P(II, L) - R2(II)
         P(II, M) = P(II, M) + R2(II)
50       CONTINUE
         SIZE(L) = SIZE(L) - 1
         SIZE(M) = SIZE(M) + 1
         CLUS(ITER) = NBEST
         INEW(L) = 1
         INEW(M) = 1
59       CONTINUE
C
C          CHECK FOR OPTIMUM, WERE THERE ANY MOVES THIS PASS.
C
         DO 60 I = 1, NCLUS
         IF (INEW(I) .GT. 0) GOTO 62
60       CONTINUE
C
C          NO MOVES, IF A GLOBAL CHECK, FINISH,
C          IF A LOCAL CHECK, MAKE A GLOBAL CHECK.
C
         IF (GLOBAL) GOTO 70
         GOTO 2
C
C          SOME MOVES, RESET IOLD, INEW, MAKE A LOCAL CHECK.
62       GLOBAL = .FALSE.
         DO 63 I = 1, NCLUS
         IOLD(I) = INEW(I)
         INEW(I) = 0
63       CONTINUE
         GOTO 4
C
C        COMPUTE OVERALL LOG LIKELIHOOD
C
70       R1(I) = 0.0
         DO 72 I = 1, NCLUS
         R1(1) = R1(1) + XLIKE(0.0, P(I, I), I, I, SIZE(I) - 1,
     *   SIZE(I), -1.0, TLOG, MXS2)
         DO 71 J = 1, NCLUS
         IF (I .NE. J) R1(1) = R1(1) + XLIKE(0.0, P(I, J),
     *   I, I, SIZE(I), SIZE(J), -1.0, TLOG, MXS2)
71       CONTINUE
72       CONTINUE
```

```
      RETURN
      END
C
      REAL FUNCTION XLIKE(P1, R1, S1, S2, S3, S4, Y1, TLOG, MXS2)
C
C       ALGORITHM AS 140.3 APPL. STATIST. (1979) VOL.28, NO.2
C
C       EVALUATE THE CHANGE IN LOG LIKELIHOOD BETWEEN P SUCCESSES IN
C       S1 * S2 TRIALS AND P1 - Y1 * R1 SUCCESSES IN S3 * S4 TRIALS.
C
      INTEGER S1, S2, S3, S4, P, R, X, Z
      REAL TLOG(MXS2)
      XLIKE = 0.0
      P = P1
      Z = S1 * S2
      R = Z - P
      IF (R .NE. 0 .AND. P .NE. 0) XLIKE = TLOG(Z) - TLOG(P) - TLOG(R)
      X = P1 - Y1 * R1
      Z = S3 * S4
      R = Z - X
      IF (R .NE. 0 .AND. X .NE. 0)
     *  XLIKE = XLIKE + TLOG(X) + TLOG(R) - TLOG(Z)
      RETURN
      END
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE SINV(A, K, L, LO, PVT, IFAULT)
C
C       ALGORITHM AS 141 APPL. STATIST. (1979) VOL.28, NO.2
C
C       CALCULATE THE INVERSE OF A SYMMETRIC MATRIX
C       IGNORING A SPECIFIED ROW/COLUMN IF LO .NE. 0,
C       USING EITHER THE ORIGINAL MATRIX OR A COMPLETE INVERSE OF IT.
C
C  Correction mentioned in vol 28 no 3 1979 p 336 applied
C
      DIMENSION A(L, L)
      REAL A, AA, AIP, BIG, EP, PVT, SMALL, T
      INTEGER P, PM, PP
C
      PARAMETER (BIG=1.0E38, SMALL=1.0E-7)
C
C       PARAMETER CHECKS
C
      IFAULT = 3
      IF (ABS(LO) .GT. K .OR. K .LT. 1 .OR. K .GT. L) RETURN
C
C       INITIAL VALUES
C
      IFAULT = 0
      IF (LO .GE. 0) GOTO 1
      EP = 1.0
      P = -LO
      PVT = ABS(A(P, P))
      T = PVT
      GOTO 3
1     EP = -1.0
      P = 1
      PVT = BIG
C
C       PIVOT BY PIVOT INVERSION
C
2     IF (P .EQ. LO) GOTO 12
      T = ABS(A(P, P))
      IF (T .LT. PVT) PVT = T
3     IF (T .LT. SMALL) IFAULT = 1
      IF (T .EQ. 0.0) GOTO 15
      PM = P - 1
      PP = P + 1
      AA = 1.0 / A(P, P)
      A(P, P) = -AA
      IF (P .EQ. 1) GOTO 8
      DO 7 I = 1, PM
        AIP = A(I, P) * AA
        DO 4 J = I, PM
4       A(I, J) = A(I, J) - AIP * A(J, P)
        IF (P .EQ. K) GOTO 6
        DO 5 J = PP, K
5       A(I, J) = A(I, J) - AIP * A(P, J)
6       A(I, P) = AIP * EP
7     CONTINUE
8     IF (P .EQ. K) GOTO 11
      DO 10 I = PP, K
        AIP = A(P, I) * AA
        DO 9 J = I, K
```

```
9         A(I, J) = A(I, J) - AIP * A(P, J)
          A(P, I) = AIP * EP
10      CONTINUE
11      IF (EP .GT. 0.0) RETURN
12      P = P + 1
        IF (P .LE. K) GOTO 2
C
C         SIGN CORRECTION
C
        DO 14 I = 1, K
          DO 13 J = I, K
13        A(I, J) = -A(I, J)
14      CONTINUE
        RETURN
C
C         NIL PIVOT EXIT
C
15      IFAULT = 2
        RETURN
        END
```

```
      SUBROUTINE EXACT(X, YOBS, ALPHA, N, K, P, XSIZE, PLT, PEQ, IFAULT)
C
C       ALGORITHM AS 142 APPL. STATIST. (1979) VOL.28, NO.3
C
C       CALCULATES THE EXACT PROBABILITIES
C       P(T(P) .LT. ALPHA) AND P(T(P) .EQ. ALPHA)
C       WHERE T(P) IS THE CONDITIONAL TEST STATISTIC FOR THE
C       REGRESSION PARAMETER BETA(P) IN A BINARY REGRESSION MODEL
C
      INTEGER P, XSIZE, YOBS(N), Y(30), P1
      INTEGER CLT, CEQ, CTOT
      REAL ALPHA, PLT, PEQ, X(XSIZE, P)
      REAL T(10), SUM
C
C       CHECK FOR INPUT PARAMETERS OUT OF RANGE
C
      IFAULT = 1
      IF (N .GT. 30) RETURN
      IFAULT = 2
      IF (P .GT. 10) RETURN
      IFAULT = 3
      IF (K .LT. 0 .OR. K .GT. N) RETURN
      IFAULT = 0
      PEQ = 1.0
      PLT = 0.0
      IF (K .EQ. 0 .OR. K .EQ. N) RETURN
C
C       INITIALIZATION
C       1.  CALCULATE OBSERVED VALUES OF CONDITIONING STATISTICS
C       T(2), ... ,T(P-1)
C       2.  CALCULATE INITIAL PERMUTATION OF K ONES AND
C       N-K ZEROES (1, ..., 1, 0, ..., 0)
C       3.  INITIALIZE NUMERATORS AND DENOMINATOR TO ZERO
C
      DO 2 I = 1, P
      T(I) = 0
      DO 1 J = 1, N
1     IF (YOBS(J) .EQ. 1) T(I) = T(I) + X(J, I)
2     CONTINUE
      P1 = P - 1
      DO 3 I = 1, K
3     Y(I) = 1
      KK = K + 1
      DO 4 I = KK, N
4     Y(I) = 0
      CLT = 0
      CEQ = 0
      CTOT = 0
C
C       CALCULATE VALUES OF CONDITIONING STATISTICS (T(2),...T(P-1))
C       FOR CURRENT PERMUTATION.
C       ADD ONE TO DENOMINATOR ONLY IF ALL MATCH OBSERVED VALUES
C
5     IF (P1 .EQ. 0) GOTO 8
      DO 7 I = 1, P1
      SUM = 0.0
      DO 6 J = 1, N
      IF (Y(J) .NE. 0) SUM = SUM + X(J, I)
6     CONTINUE
      IF (SUM .NE. T(I)) GOTO 10
7     CONTINUE
```

```
8       CTOT = CTOT + 1
C
C         CALCULATE VALUE OF T(P) FOR CURRENT PERMUTATION
C
        SUM = 0.0
        DO 9 I = 1, N
        IF (Y(I) .NE. 0) SUM = SUM + X(I, P)
9       CONTINUE
C
C         ADD ONE TO NUMERATOR(S) ONLY IF TEST IS PASSED
C
        IF (SUM .LT. ALPHA) CLT = CLT + 1
        IF (SUM .EQ. ALPHA) CEQ = CEQ + 1
C
C         GET ANOTHER PERMUTATION.
C         IF LAST IS ALREADY PROCESSED THEN CALCULATE
C         PROBABILITY AS NUMERATOR/DENOMINATOR
C
10      CALL GETONE(Y, N, IFLAG)
        IF (IFLAG .EQ. 0) GOTO 5
        PEQ = FLOAT(CEQ) / FLOAT(CTOT)
        PLT = FLOAT(CLT) / FLOAT(CTOT)
        RETURN
        END
C
        SUBROUTINE GETONE(RET, N, IFLAG)
C
C         ALGORITHM AS 142.1 APPL. STATIST. (1979) VOL.28, NO.3
C
C         SET DEFAULT RETURN FLAG
C         FLAG=1 - LAST PERMUTATION
C         (0, ..., 0, 1, ..., 1) GENERATED ON PREVIOUS CALL
C         FLAG=0 - RETURNED PERMUTATION SHOULD BE PROCESSED
C
        INTEGER RET(N)
        IFLAG = 0
C
C         INITIAL ACTION DEPENDS ON LEADING ONE OR ZERO
C
        IF (RET(1) .EQ. 0) GOTO 2
C
C         IF LEADING ONE, LOOK FOR FIRST ZERO, THEN
C         INTERCHANGE WITH LAST ONE
C
        DO 1 I = 2, N
        IF (RET(I) .EQ. 1) GOTO 1
        RET(I) = 1
        RET(I - 1) = 0
        RETURN
1       CONTINUE
        RETURN
C
C         INITIAL DIGIT IS ZERO
C         FIND FIRST ONE THEN NEXT ZERO (COORDINATE J)
C         CHECK FOR LAST PERMUTATION
C
2       N2 = N - 1
        DO 3 I = 2, N2
        IF (RET(I) .EQ. 1) GOTO 4
3       CONTINUE
        IFLAG = 1
```

```
        RETURN
4       I2 = I + 1
        DO 5 J = I2, N
        IF (RET(J) .EQ. 0) GOTO 6
5       CONTINUE
        IFLAG = 1
        RETURN
C
C         NOT ON LAST PERMUTATION THEREFORE
C         DEMOTE LAST ONE (INTERCHANGE LAST ONE - NEXT ZERO)
C         AND REVERSE PRECEDING ZEROES-ONES
C
6       RET(J) = 1
        RET(J - 1) = 0
        IF (J .EQ. 2) RETURN
        K = J - 2
        J = K / 2
        DO 7 I = 1, J
        SAVE = RET(K)
        RET(K) = RET(I)
        RET(I) = SAVE
        K = K - 1
7       CONTINUE
        RETURN
        END
```

```
      subroutine median (m, n, x, norm, c, f, med, ifault,
     *                   maxm, maxn, md, g, q, l)
c
c        Algorithm AS 143  Appl. Statist. (1979) Vol. 28, No. 3
c
c        The Mediancentre
c        At least 10 times faster than AS 78.
c
c        Parameters:
c
c        m       integer     input: number of dimensions
c        n       integer     input: number of sample points
c        x       real array  input: data, n points, m dimensions
c        norm    real        output: norm of gradient
c        c       real        output: multiplicity of m in S ; 0 if m not in S
c        med     real array  output: coordinates of the mediancentre
c        ifault  integer     output: -1 if m or n out of range
c                                     0 if matrix of second derivatives
c                                       is positive definite
c                                     1 if matrix of second derivatives
c                                       is not positive definite
c
      implicit double precision (a-h, o-z)
      dimension x(maxn,maxm), med(maxm), md(maxm), g(maxm),
     *          q(maxm,maxm), l(maxn)
      double precision norm, med, md
c
      data zero, one /0.0d0, 1.0d0/
      data eps /0.0001d0/
c
      ifault = -1
      if (m .lt. 1 .or. m .gt. maxm .or. n .lt. 1 .or. n .gt. maxn)
     *    return
      ifault = 0
      do 1 i = 1, m
    1 med(i) = zero
      nn = n
      do 2 i = 1, n
    2 l(i) = i
c
c        computation of (residual) mean
c
    3 f = one / float(nn)
      do 5 i = 1, m
        s = zero
        g(i) = zero
        do 4 j = 1, nn
          k = l(j)
          s = s + x(k, i)
    4   continue
        md(i) = s * f
        med(i) = med(i) + md(i)
    5 continue
c
c        Computation of function, gradient, and norm of gradient
c
      c = zero
      f = zero
      s = zero
      do 9 i = 1, n
        t = zero
```

```
          do 6 j = 1, n
            x(i, j) = x(i, j) - md(j)
            t = t + x(i, j) * x(i, j)
     6    continue
          if (t .gt. zero) goto 7
          c = c + one
          goto 9
     7    t = sqrt(t)
          r = one / t
          s = s + r
          f = f + t
          do 8 j = 1, m
            g(j) = g(j) - r * x(i, j)
     8    continue
     9  continue
        norm = zero
        do 10 i = 1, m
          norm = norm + g(i) * g(i)
    10  continue
        norm = sqrt(norm)
c
c          Check for extremal points
c
        if (norm .le. c + eps) return
        if (nn .eq. 1) goto 13
c
c          Simplex
c
        nt = nn
        nn = 0
        do 12 i = 1, nt
          k = l(i)
          ang = zero
          do 11 j = 1, m
            ang = ang + g(j) * x(k, j)
    11    continue
          if (ang .gt. zero) goto 12
          nn = nn + 1
          l(nn) = k
    12  continue
        if (nn .gt. 0) goto 3
c
c          Starting value
c
    13  r = (c / norm - one) / s
        do 14 i = 1, m
          md(i) = r * g(i)
    14  continue
c
c          Newton - Raphson procedure
c
    15  c = zero
        f = zero
        s = zero
        do 16 i = 1, m
          g(i) = zero
          med(i) = med(i) + md(i)
          do 16 j = i, m
            q(i, j) = zero
    16  continue
        do 19 i = 1, n
```

```
           t = zero
           do 17 j = 1, m
             x(i, j) = x(i, j) - md(j)
             t = t + x(i, j) * x(i, j)
      17   continue
           if (t .eq. zero) goto 19
           t = sqrt(t)
           r = one / t
           s = s + r
           rr = r * r * r
           f = f + t
           do 18 j = 1, m
             g(j) = g(j) - r * x(i, j)
             do 18 k = j, m
               q(j, k) = q(k, j) - rr * x(i, j) * x(i, k)
      18   continue
      19 continue
         norm = zero
         do 20 j = 1, m
           norm = norm + g(j) * g(j)
      20 continue
         norm = sqrt(norm)
         if (norm .le. eps) return
c
c        Cholesky and solution of equation
c
         ifault = 1
         if (q(1,1) + s .le. zero) return
         q(1,1) = sqrt(q(1,1) + s)
         do 21 i = 2, m
           q(i, 1) = q(1, i) / q(1, 1)
      21 continue
         do 25 j = 2, m
           tmp = s + q(j, j)
           j1 = j - 1
           do 22 i = 1, j1
             tmp = tmp - q(j, i) * q(j, i)
      22   continue
           if (tmp .le. zero) return
           q(j, j) = sqrt(tmp)
           j2 = j + 1
           if (j2 .gt. m) goto 25
           do 24 k = j2, m
             tmp = q(j, k)
             do 23 i = 1, j1
               tmp = tmp - q(j, i) * q(k, i)
      23     continue
             q(k, j) = tmp / q(j, j)
      24   continue
      25 continue
         md(1) = -g(1) / q(1, 1)
         do 27 j = 2, m
           j1 = j - 1
           tmp = -g(j)
           do 26 i = 1, j1
             tmp = tmp - q(j, i) * md(i)
      26   continue
           md(j) = tmp / q(j, j)
      27 continue
         md(m) = md(m) / q(m, m)
         j1 = m - 1
```

```
      do 29 j2 = 1, j1
        i = m - j2
        tmp = md(i)
        k = i + 1
        do 28 j = k, m
          tmp = tmp - q(j, i) * md(j)
28      continue
        md(i) = tmp / q(i, i)
29    continue
      ifault = 0
      goto 15
      end
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE MULMAX(N, K, NMIN, NN, NNK, PROB, A, FAC, IFAULT)
C
C       ALGORITHM AS 145 APPL. STATIST. (1979) VOL.28, NO.3
C
C       N IS THE NUMBER OF BALLS
C       K IS THE NUMBER OF BOXES
C       PROB(I) IS THE PROBABILITY THAT THE MAXIMUM
C       NUMBER OF BALLS IN A BOX IS I + NMIN - 1
C
      REAL PROB(NN), FAC(NNK), NUM, NUP
      INTEGER A(K), SUM
C
      PARAMETER (ZERO=0.0,TEN=10.0,EXPLIM=75.0)
C
      IFAULT = 1
      NUP = FLOAT(N) * ALOG10(FLOAT(K))
      IF (NUP .GT. EXPLIM) RETURN
      NUP = TEN ** NUP
      IFAULT = 2
      SUM = (N + K -1) / K
      IF (NMIN .GT. N .OR. NMIN .LT. SUM) RETURN
      IFAULT = 3
      IF (N .LE. 1) RETURN
      IFAULT = 0
      DO 1 I = 1, NN
1     PROB(I) = ZERO
C
C       THIS GENERATES THE FIRST PARTITION
C
      M = 1
      A(1) = N
C
C       A NEW PARTITION IS STORED IN A(1) TO A(M)
C
2     NUM = FAC(N)
C
C       THIS COMPUTES THE LOGARITHM OF THE NUMBER
C       OF PARTITIONS OF BALLS
C
      DO 3 I = 1, M
        NA = A(I)
        NUM = NUM - FAC(NA)
3     CONTINUE
C
C       THIS COMPUTES THE LOGARITHM OF THE NUMBER
C       OF PERMUTATIONS OF BOXES
C
      NUM = NUM + FAC(K)
      KM = K - M
      IF (M .LT. K) NUM = NUM - FAC(KM)
      IF (M .EQ. 1) GOTO 6
      I = 1
      II = 2
4     IF (A(II) .EQ. A(I)) GOTO 5
      IJ = II - I
      NUM = NUM - FAC(IJ)
      I = II
5     II = II + 1
      IF (II .LE. M) GOTO 4
```

```
        IJ = II - I
        IF (IJ .GT. 1) NUM = NUM - FAC(IJ)
C
C         THE NUMBER OF ALLOCATIONS GIVING RISE TO THIS PARTITION
C         IS CUMULATED ACCORDING TO THE MAXIMUM FREQUENCY
C
6       KX = A(1) - NMIN + 1
        PROB(KX) = PROB(KX) + EXP(NUM)
C
C         THE MAIN ALGORITHM FOR GENERATING PARTITIONS STARTS HERE
C
7       IF (A(M) .EQ. 1) GOTO 11
        A(M) = A(M) -1
        IF (A(1) .LT. NMIN) GOTO 13
8       M = M + 1
        IF (M .GT. K) GOTO 12
        SUM = A(1)
        L = 2
9       IF (L .GE. M) GOTO 10
        SUM = SUM + A(L)
        L = L + 1
        GOTO 9
10      A(M) = N - SUM
        IF (A(M) .LE. A(M -1)) GOTO 2
        IF (M .GE. K) GOTO 12
        A(M) = A(M -1)
        GOTO 8
11      M = M -1
        GOTO 7
12      M = M -2
        GOTO 7
C
C         ALL PARTITIONS HAVE NOW BEEN GENERATED
C
13      DO 14 I = 1, NN
14      PROB(I) = PROB(I) / NUP
        RETURN
        END
```

```
      double precision function gammds (y,p,ifault)
c
c        Algorithm AS 147  Appl. Statist. (1980) Vol. 29, No. 1
c
c        Computes the incomplete gamma integral for positive
c        parameters y,p using an infinite series
c
c        Auxiliary function required: ALNGAM = CACM algorithm 291
c
c        AS239 should be considered as an alternative to AS147
c
      implicit double precision (a-h,o-z)
      data e/1.0d-9/, zero/0.0d0/, one/1.0d0/, uflo/1.0d-37/
c
c        Checks admissibility of arguments and value of f
c
      ifault = 1
      gammds = zero
      if(y.le.zero .or. p.le.zero) return
      ifault = 2
c
c        alngam is natural log of gamma function
c
      arg = p*log(y)-alngam(p+one)-y
      if(arg.lt.log(uflo)) return
      f = exp(arg)
      if(f.eq.zero) return
      ifault = 0
c
c          Series begins
c
      c = one
      gammds = one
      a = p
    1 a = a+one
      c = c*y/a
      gammds = gammds+c
      if (c/gammds.gt.e) goto 1
      gammds = gammds*f
      return
      end
```

```
      SUBROUTINE JKKN(X, XX, K, N, TH, THJ, LTH, COV, PS, NGRP, IGSIZE,
     *  IFAULT)
C
C        REMARK ASR62 ON AS 148 APPL. STATIST. (1980) VOL. 29 NO. 1.
C
C        REMOVAL OF BIAS BY THE JACKKNIFE PROCEDURE
C
      INTEGER K, N, LTH, NGRP, IGSIZE, IFAULT
      REAL X(K, N), TH(LTH), THJ(LTH), COV(LTH, LTH), PS(LTH, NGRP),
     *  XX(K, N)
C
      IFAULT = 1
      IF (NGRP .LE. 1) RETURN
      IFAULT = 0
      NGRP1 = NGRP - 1
      NN = NGRP1 * IGSIZE
      ENGRP = NGRP
      ENGRP1 = NGRP1
      ENN = ENGRP * ENGRP1
      IGSZ1 = IGSIZE + 1
      KI = K * IGSIZE
C
C        KI = NUMBER OF X COMPONENTS IN A GROUP
C
      LIN = NGRP1 * KI
C
C        FIRST CALCULATE NGRP*THETAHAT AND STORE IT IN TH
C
      CALL THTHT(X, K, N, TH, LTH)
      DO 10 I = 1, LTH
   10 TH(I) = ENGRP * TH(I)
C
C        CALCULATE THE NGRP PSEUDOVALUES
C
C     IG1 = 0
C     DO 12 J = 1, K
C       DO 11 IN = I, NGRP
C  11   XX(J, IN) = X(J, IN)
C  12 CONTINUE
      DO 50 I = 1, NGRP
C
C        MOVE THE I-TH GROUP OF OBSERVATIONS TO THE NGRP-TH GROUP
C        POSITION, THEN MOVE POSITION OF THE I+1-TH GROUP I-TH
C        POSITION, I+2-TH TO I+1-TH AND SO ON
C
        DO 20 IG = 1, IGSIZE
          DO 15 J = 1, K
            TEMP = XX(J, I)
            DO 14 IN = I, NN
              NUM = IN + 1
   14       XX(J, IN) = XX(J, NUM)
            XX(J, NGRP) = TEMP
   15     CONTINUE
   20   CONTINUE
        CALL THTHT(XX, K, NN, THJ, LTH)
        DO 22 J = 1, K
          DO 21 IN = I, NGRP
   21     XX(J, IN) = X(J, IN)
   22   CONTINUE
        DO 35 II = 1, LTH
   35   PS(II, I) = TH(II) - THJ(II) * ENGRP1
```

```
   50 CONTINUE
C
C        INITIALIZE
C
      DO 75 I = 1, LTH
        THJ(I) = 0.0
        DO 75 J = 1, LTH
          COV(I, J) = 0.0
   75 CONTINUE
C
C        CALCULATE JACKKNIFED VERSION OF THETAHAT AND STORE IN THJ
C
      DO 100 I = 1, NGRP
        DO 90 II = 1, LTH
   90   THJ(II) = THJ(II) + PS(II, I)
  100 CONTINUE
      DO 110 II = 1, LTH
  110 THJ(II) = THJ(II) / ENGRP
C
C        CALCULATE THE APPROXIMATE COVARIANCE MATRIC OF THE JACKKNIFED
C        THETAHAT AND STORE IT IN COV
C
      DO 125 II = 1, LTH
        THJII = THJ(II)
        DO 125 JJ = II, LTH
          THJJJ = THJ(JJ)
          DO 115 I = 1, NGRP
            COV(II, JJ) = COV(II, JJ) + (PS(II, I) - THJII) * (PS(JJ, I)
     *                     - THJJJ)
  115     CONTINUE
  125 CONTINUE
C
C        IF LTH .GT. 1, COPY THE UPPER-TRIANGULAR SECTION OF COV INTO
C        THE LOWER TRIANGULAR SECTION
C
      IF (LTH .EQ. 1) GOTO 200
      DO 175 II = 2, LTH
        II1 = II - 1
        DO 175 JJ = 1, II1
          COV(II, JJ) = COV(JJ, II)
  175 CONTINUE
  200 DO 210 II = 1, LTH
        DO 210 JJ = 1, LTH
  210 COV(II, JJ) = COV(II, JJ) / ENN
      RETURN
      END
```

```
      SUBROUTINE AMALGM(K, XO, WO, X, W, XA, IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C
C        ALGORITHM AS 149 APPL. STATIST. (1980) VOL.29, NO.2
C
C        AMALGAMATION  OF  MEANS  BY  THE  UP-AND-DOWN  BLOCKS  ALGORITHM
C        OF  KRUSKAL  (BARLOW  ET  AL. ,1972, P.72 )
C
      DIMENSION XO(K),WO(K),X(K),W(K),XA(K)
      DATA TOL /1.0E-6/
      IFAULT = 1
C
C        CHECK  THAT  K .GE. 2
C
      IF (K .LT. 2) RETURN
      IFAULT = 2
C
C        CHECK  THAT  THE  WEIGHTS  ARE  POSITIVE
C
      DO 1 I = 1,K
      IF (WO(I) .LE. 0.0) RETURN
    1 CONTINUE
      IFAULT = 0
      DO 2 I = 1,K
      X(I) = XO(I)
      W(I) = WO(I)
    2 CONTINUE
      M = K
      I = 1
    3 IF (I .EQ. M) GOTO 4
      IF (X(I) .GT. X(I+1)) GOTO 9
      IF (I .EQ. 1) GOTO 13
    4 IF (X(I-1) .GT. X(I)) GOTO 6
      IF (I .LT. M) GOTO 13
      GOTO 14
C
C        POOL  THE  ACTIVE  BLOCK  WITH  THE  NEXT  LOWER  BLOCK
C
    6 IM1 = I - 1
      WW = W(IM1) + W(I)
      X(IM1) = (W(IM1)*X(IM1) + W(I)*X(I))/WW
      W(IM1) = WW
      MM1 = M  - 1
      IF (I .EQ. M) GOTO 8
      DO 7 J = I,MM1
      J1 = J + 1
      X(J) = X(J1)
      W(J) = W(J1)
    7 CONTINUE
    8 I = IM1
      M = MM1
      IF (M .EQ. 1) GOTO 14
      GOTO 3
C
C        POOL  THE  ACTIVE  BLOCK  WITH  THE  NEXT  HIGHER  BLOCK
C
    9 I1 = I + 1
      WW = W(I) + W(I1)
```

```
         X(I) = (W(I)*X(I) + W(I1)*X(I1))/WW
         W(I) = WW
         MM1 = M - 1
         IF (I1 .EQ. M) GOTO 11
         DO 10 J = I1,MM1
         J1 = J + 1
         X(J) = X(J1)
         W(J) = W(J1)
      10 CONTINUE
      11 M = MM1
         IF (M .EQ. 1) GOTO 14
         IF (I .EQ. 1) GOTO 12
         IF (X(I-1) .GT. X(I)) GOTO 6
      12 IF (I .EQ. M) GOTO 14
         IF (X(I) .GT. X(I+1)) GOTO 9
      13 I = I + 1
         GO TO 12
C
C        OBTAIN  THE  AMALGAMATED  MEANS  XA(K) FROM  THE  WORKING  ARRAY  X(M)
C
      14 I1 = 1
         DO  17 I = 1,M
         S = 0.0
         DO 15 J = I1,K
         S = S + WO(J)
         XA(J) = X(I)
         IF (ABS(S - W(I)) .LT. TOL) GOTO 16
      15 CONTINUE
      16 I1 = J + 1
      17 CONTINUE
         RETURN
         END
```

```
      SUBROUTINE SCOUNT(NEVENT,TZERO,NF,NW,T,SPEC,FREQ,C,S,IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C     ALGORITHM AS 150 APPL. STATIST. (1980) VOL.29, NO.2
C
C     COMPUTES ESTIMATES OF THE SPECTRUM OF A POINT PROCESS
C     BY TAKING A CENTRED MOVING AVERAGE OF THE NORMALISED
C     PERIODOGRAM OF THE COUNTING PROCESS
C
C
      DIMENSION T(NEVENT),FREQ(NF),SPEC(NF),C(NF),S(NF)
      REAL TWOPI
      DATA TWOPI,NRECUR/6.283185307196E0,100/
C
C      TEST FOR ERRORS
C
      IFAULT = 0
      IF(NW.LE.0) GO TO 10
      IF(NW.GT.NF) GO TO 11
      IF(MOD(NW,2).EQ.0) GO TO 12
      IF(NEVENT.LE.0) GO TO 13
      IF(TZERO.LE.0.0) GO TO 14
      IF(T(1).LT.0.0 .OR. T(NEVENT).GT.TZERO) GO TO 15
      DO 1 I = 2,NEVENT
      IF(T(I).LE.T(I-1)) GO TO 15
    1 CONTINUE
C
C     END OF ERROR-TESTING SECTION
C
      FW = NW
      FN = NEVENT
      RTZERO = 1.0/TZERO
C
C     INITIALISE ARRAYS FOR SUMS OF SINES AND COSINES
C     AND CALCULATE FREQUENCIES
C
      DO 2 I = 1,NF
      C(I) = 0.0
      S(I) = 0.0
      SPEC(I) = I
      FREQ(I) = SPEC(I)*RTZERO
    2 CONTINUE
C
C     CALCULATE SUMS OF SINES AND COSINES USING THE DOUBLE-
C     ANGLE SINE AND COSINE FORMULAS - EXCEPT COMPUTE EVERY
C     (NRECUR)TH. DIRECTLY
C
      DO 6 I = 1,NEVENT
      ANGLE = TWOPI*T(I)*RTZERO
      COSINI = COS(ANGLE)
      SININI = SIN(ANGLE)
      COSREC = 1.0
      SINREC = 0.0
      DO 5 J = 1,NF
C
C     TEST WHETHER NEED TO COMPUTE SINE AND COSINE DIRECTLY
C
      IF(MOD(J,NRECUR).NE.0) GO TO 3
C
```

```
C     RECALL THAT SPEC(J) = J
C
      FMULT = SPEC(J)*ANGLE
      COSREC = COS(FMULT)
      SINREC = SIN(FMULT)
      GOTO 4
    3 TEMP = COSREC*COSINI - SINREC*SININI
      SINREC = COSREC*SININI + SINREC*COSINI
      COSREC = TEMP
    4 C(J) = C(J) + COSREC
      S(J) = S(J) + SINREC
    5 CONTINUE
    6 CONTINUE
C
C     S(.) AND C(.) NOW CONTAIN THE SUMS OF SINES AND
C            COSINES,RESPECTIVELY
C
C
C     NOW USE SPEC(.) FOR THE NORMALISED PERIODOGRAM
C
      CNORM = 2.0/FN
      DO 7 I = 1,NF
    7 SPEC(I) = CNORM*(C(I)*C(I) + S(I)*S(I))
C
C     NOW OBTAIN THE SMOOTHED SPECTRAL ESTIMATES -
C     RETURN IF WANT ONLY THE PERIODOGRAM
C
      IF(NW.EQ.1) RETURN
      KU = NF - NW + 1
      L = NW/2
      DO 9 I = 1,KU
      SUM = 0.0
      J = I + NW - 1
      DO 8 K = I,J
    8 SUM = SUM + SPEC(K)
C
C     STORE THE SMOOTHED ESTIMATES AND THE CORRESPONDING
C     FREQUENCIES IN THE FIRST (NF - NW + 1) ELEMENTS OF
C         SPEC(.) AND FREQ(.)
C
      SPEC(I) = SUM/FW
      M = I + L
      FREQ(I) = FREQ(M)
    9 CONTINUE
      RETURN
   10 IFAULT = 1
      RETURN
   11 IFAULT = 2
      RETURN
   12 IFAULT = 3
      RETURN
   13 IFAULT = 4
      RETURN
   14 IFAULT = 5
      RETURN
   15 IFAULT = 6
      RETURN
      END
```

```
      SUBROUTINE BIVCNT(N1,N2,TZERO,NSECT,T1,T2,NF,TTEMP,SPC1,SPC2,C1,
     * S1,C2,S2,NT1,NT2,NN1,NN2,SPEC1,SPEC2,COHERE,PHASE,FREQ,IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C     ALGORITHM AS 151 APPL. STATIST. (1980) VOL.29, NO.2
C
C     CALCULATES SMOOTHED SPECTRAL ESTIMATES FOR A BIVARIATE POINT
C     PROCESS BY SPLITTING THE PERIOD OF OBSERVATION INTO NONOVERLAPPING
C     SECTIONS OF EQUAL LENGTH
C
      DIMENSION T1(N1),T2(N2),FREQ(NF),SPEC1(NF),SPEC2(NF),COHERE(NF),
     1 PHASE(NF),C1(NF),S1(NF),C2(NF),S2(NF),TTEMP(NF),SPC1(NF),SPC2(NF)
      INTEGER NT1(NSECT),NT2(NSECT),NN1(NSECT),NN2(NSECT)
C
C     TEST FOR PARAMETER ERRORS
C
      IF(TZERO.LE.0.0) GO TO 9
      IF(N1.LE.0 .OR. N2.LE.0) GO TO 10
      IF(T1(N1).GT.TZERO .OR. T2(N2).GT.TZERO) GO TO 11
      IF(NSECT.LE.0) GO TO 12
      IF(NF.LT.NSECT) GO TO 13
      IFAULT = 0
C
C     INITIALISE ARRAYS FOR ACCUMULATING SPECTRAL ESTIMATES
C
      DO 1 I = 1,NF
      SPEC1(I) = 0.0
      SPEC2(I) = 0.0
      COHERE(I) = 0.0
      PHASE(I) = 0.0
    1 CONTINUE
C
C     UNLESS NSECT = 1,CALL SPLIT TO SECTION THE PERIOD OF OBSERVATION
C     AND CALCULATE THE EVENT TIMES RELATIVE TO THE SECTION ORIGINS
C
      NT1(1) = N1
      NT2(1) = N2
      NN1(1) = N1
      NN2(1) = N2
      IF(NSECT.EQ.1) GO TO 3
      CALL SPLIT(NSECT, N1, TZERO, T1, NT1, TTEMP, IFAULT)
      IF (IFAULT.NE.0) RETURN
      CALL SPLIT(NSECT, N2, TZERO, T2, NT2, TTEMP, IFAULT)
      IF(IFAULT.NE.0) RETURN
C
C     CALCULATE NO. OF EVENTS IN EACH SECTION OF BOTH SERIES
C
      NN1(1) = NT1(1)
      NN2(1) = NT2(1)
C
C     TEST WHETHER THE WORKSPACE ARRAY TTEMP(.) IS LARGE ENOUGH
C     TO STORE THE EVENT TIMES IN SECTION 1 OF EACH SERIES
C
      IF(NN1(1).GT.NF.OR.NN2(1).GT.NF) GO TO 14
      DO 2 I = 2,NSECT
      NN1(I) = NT1(I)-NT1(I-1)
      NN2(I) = NT2(I)-NT2(I-1)
C
C     DO THE SAME FOR THE OTHER SECTIONS
```

```
C
      IF(NN1(I).GT.NF.OR.NN2(I).GT.NF) GO TO 14
    2 CONTINUE
C
C     CALL SCOUNT SECTION BY SECTION,USING TTEMP(.) TO STORE
C     THE EVENT TIMES TEMPORARILY
C
    3 FN = 1.0/FLOAT(NSECT)
      TS = TZERO*FN
      KU1 = 0
      KU2 = 0
      DO 7 J = 1,NSECT
      KL1 = KU1+1
      KU1 = NT1(J)
      KL2 = KU2+1
      KU2 = NT2(J)
      DO 4 I = KL1,KU1
      K = I-KL1+1
      TTEMP(K) = T1(I)
    4 CONTINUE
C
C     CALL SCOUNT WITH NW-1 TO OBTAIN THE NORMALISED PERIODOGRAM AND
C     THE SUM OF SINES AND COSINES FOR SECTION J OF SERIES 1
C
      CALL SCOUNT(NN1(J), TS, NF, 1, TTEMP, SPC1, FREQ, C1, S1, IFAULT)
      IF(IFAULT.NE.0) RETURN
      DO 5 I = KL2, KU2
      K = I-KL2+1
      TTEMP(K) = T2(I)
    5 CONTINUE
C
C     DO THE SAME FOR SERIES 2
C
      CALL SCOUNT(NN2(J), TS, NF, 1, TTEMP, SPC2, FREQ, C2, S2, IFAULT)
      IF(IFAULT.NE.0) RETURN
      SQNN = 1.0/SQRT(FLOAT(NN1(J)*NN2(J)))
C
C     ACCUMULATE(OVER THE NSECT SECTIONS) THE SPECTRAL ESTIMATES
C                      AT EACH FREQUENCY
C
      DO 6 I = 1,NF
      SPEC1(I) = SPEC1(I) + SPC1(I)
      SPEC2(I) = SPEC2(I) + SPC2(I)
      COHERE(I) = COHERE(I) + (C1(I)*C2(I) + S1(I)*S2(I))*SQNN
      PHASE(I) = PHASE(I) + (C2(I)*S1(I) - C1(I)*S2(I))*SQNN
    6 CONTINUE
    7 CONTINUE
C
C     NOW FIND THE SMOOTHED ESTIMATES
C
      DO 8 I = 1,NF
      SPEC1(I) = SPEC1(I)*FN
      SPEC2(I) = SPEC2(I)*FN
      TEMP = COHERE(I)
      COHERE(I) = 4.0*FN*FN*(TEMP*TEMP + PHASE(I)*PHASE(I))/
     1  (SPEC1(I)*SPEC2(I))
      PHASE(I) = ATAN2(PHASE(I),TEMP)
    8 CONTINUE
      RETURN
    9 IFAULT = 7
      RETURN
```

```
     10 IFAULT = 8
        RETURN
     11 IFAULT = 9
        RETURN
     12 IFAULT = 10
        RETURN
     13 IFAULT = 11
        RETURN
     14 IFAULT = 12
        RETURN
        END
        SUBROUTINE SPLIT(NSECT,NEVENT,TZERO,T,NT,TSECT,IFAULT)
C
C     ALGORITHM AS 151.1 APPL. STATIST. (1980) VOL.29, NO.2
C
C     SPLITS A PERIOD OF OBSERVATION ON A POINT PROCESS INTO
C     NONOVERLAPPING SECTIONS OF EQUAL LENGTH AND COMPUTES
C     THE EVENT TIMES RELATIVE TO THE SECTION ORIGINS
C
C
        DIMENSION T(NEVENT),TSECT(NSECT)
        INTEGER NT(NSECT)
        IFAULT = 0
C
C     TEST FOR PARAMETER ERRORS
C
        IF (NSECT.LE.1.OR.NSECT.GT.NEVENT) GO TO 7
        IF(TZERO.LE.0.0) GO TO 8
        IF(T(NEVENT).GT.TZERO) GO TO 9
C
        INITIA = NEVENT/NSECT
        JJ = NSECT-1
        FN = TZERO/FLOAT(NSECT)
        DO 4 I = 1,JJ
C
C     COMPUTE THE TIMES AT WHICH THE SECTIONS END
C
        TSECT(I) = FLOAT(I)*FN
C
C     AS A STARTING POINT,ASSUME THAT ALL SECTIONS CONTAIN THE
C     SAME NUMBER OF EVENTS
C
        NT(I) = I*INITIA
        INDEX = NT(I)
C
C     TEST WHETHER THE FINISHING POINT FOR SECTION I IS TOO
C     SMALL,JUST RIGHT OR TOO HIGH
C
        IF(T(INDEX)-TSECT(I))1,4,3
     1  NT(I) = NT(I)+1
        INDEX = NT(I)
        IF(INDEX.GT.NEVENT) GO TO 2
        IF(T(INDEX)-TSECT(I))1,4,2
     2  NT(I) = NT(I)-1
        GOTO 4
     3  NT(I) = NT(I)-1
        INDEX = NT(I)
        IF(INDEX.LT.1) GO TO 4
        IF(T(INDEX).GT.TSECT(I)) GO TO 3
     4  CONTINUE
        NT(NSECT) = NEVENT
```

```
C
C     NT(I) IS NOW EQUAL TO THE TOTAL NUMBER OF EVENTS IN
C     THE FIRST I SECTIONS
C
C     NOW COMPUTE THE EVENT TIMES RELATIVE TO THE SECTION ORIGINS -
C     FIRST TEST WHETHER SECTION 1 CONTAINS NO EVENTS
C
      IF(NT(1).EQ.0) GO TO 10
      DO 6 I = 2,NSECT
      KK = I-1
      LK = NT(KK)+1
      LU = NT(I)
C
C     TEST WHETHER THE SECTION CONTAINS NO EVENTS
C
      IF(LK.GT.LU) GO TO 10
      DO 5 J = LK,LU
    5 T(J) = T(J)-TSECT(KK)
    6 CONTINUE
      RETURN
    7 IFAULT = 13
      RETURN
    8 IFAULT = 14
      RETURN
    9 IFAULT = 15
      RETURN
   10 IFAULT = 16
      RETURN
      END
```

```
      REAL FUNCTION CHYPER(POINT, KK, LL, MM, NN, IFAULT)
C
C     ALGORITHM AS R77  APPL. STATIST. (1989), VOL.38, NO.1
C     Replaces AS 59 and AS 152
C     Incorporates AS R86 from vol.40(2)
C
C     Auxiliary routines required: ALNFAC (AS 245), ALNORM (AS 66)
C
      INTEGER          KK, LL, MM, NN, IFAULT, K, L, M, N, I, J, NL, KL,
     *                 MNKL, MVBIG, MBIG
      REAL             ZERO, ONE, P, PT, HALF, ALNFAC, ELIMIT, MEAN,
     *                 SIG, ALNORM, SXTEEN, SCALE, ROOTPI, ARG, HUNDRD
      LOGICAL          POINT, DIR
      PARAMETER (ZERO = 0.0, HALF = 0.5, ONE = 1.0, MVBIG = 1000,
     *           MBIG = 600, ELIMIT = -88.0, SXTEEN = 16.0,
     *           SCALE = 1.0E35, ROOTPI = 2.50662 82746 31001,
     *           HUNDRD = 100.0)
C
      K = KK + 1
      L = LL + 1
      M = MM + 1
      N = NN + 1
      DIR = .TRUE.
C
C     Check arguments are within permitted limits
C
      IFAULT = 1
      CHYPER = ZERO
      IF (N .LT. 1 .OR. M .LT. N .OR. K .LT. 1 .OR. K .GT. M) RETURN
C
      IFAULT = 2
      IF (L .LT. 1 .OR. K-L .GT. M-N) RETURN
      IF (.NOT. POINT) CHYPER = ONE
      IF (L .GT. N .OR. L .GT. K) RETURN
      IFAULT = 0
      CHYPER = ONE
      IF (K .EQ. 1 .OR. K .EQ. M .OR. N .EQ. 1 .OR. N .EQ. M) RETURN
      IF (.NOT. POINT .AND. LL .EQ. MIN(KK, NN)) RETURN
C
      P = REAL(NN) / REAL(MM - NN)
      IF (REAL(MIN(KK, MM-KK)) .GT. SXTEEN * MAX(P, ONE/P) .AND.
     *   MM .GT. MVBIG .AND. ELIMIT .GT. -HUNDRD) THEN
C
C     Use a normal approximation
C
      MEAN = REAL(KK) * REAL(NN) / REAL(MM)
      SIG = SQRT(MEAN * (REAL(MM-NN) / REAL(MM)) * (REAL(MM-KK) /
     *            (REAL(MM-1))))
      IF (POINT) THEN
        ARG = -HALF * (((REAL(LL) - MEAN) / SIG)**2)
        CHYPER = ZERO
        IF (ARG .GE. ELIMIT) CHYPER = EXP(ARG) / (SIG * ROOTPI)
      ELSE
        CHYPER = ALNORM((REAL(LL) + HALF - MEAN) / SIG, .FALSE.)
      END IF
C
      ELSE
C
C     Calculate exact hypergeometric probabilities.
C     Interchange K and N if this saves calculations.
C
```

```
      IF (MIN(K-1, M-K) .GT. MIN(N-1, M-N)) THEN
        I = K
        K = N
        N = I
      END IF
      IF (M-K .LT. K-1) THEN
        DIR = .NOT. DIR
        L = N - L + 1
        K = M - K + 1
      END IF
C
      IF (MM .GT. MBIG) THEN
C
C     Take logarithms of factorials.
C
        P = ALNFAC(NN) - ALNFAC(MM) + ALNFAC(MM-KK) + ALNFAC(KK) +
     *        ALNFAC(MM-NN) - ALNFAC(LL) - ALNFAC(NN-LL) - ALNFAC(KK-LL)
     *      - ALNFAC(MM-NN-KK+LL)
        CHYPER = ZERO
        IF (P .GE. ELIMIT) CHYPER = EXP(P)
        ELSE
C
C     Use Freeman/Lund algorithm
C
        DO 3 I = 1, L-1
          CHYPER = CHYPER * REAL(K-I) * REAL(N-I) / (REAL(L-I) *
     *                     REAL(M-I))
    3     CONTINUE
        IF (L .NE. K) THEN
          J = M - N + L
          DO 5 I = L, K-1
            CHYPER = CHYPER * REAL(J-I) / REAL(M-I)
    5          CONTINUE
        END IF
C
        END IF
C
        IF (POINT) RETURN
        IF (CHYPER .EQ. ZERO) THEN
C
C     We must recompute the point probability since it has underflowed.
C
        IF (MM .LE. MBIG) P = ALNFAC(NN) - ALNFAC(MM) + ALNFAC(KK) +
     *      ALNFAC(MM-NN) - ALNFAC(LL) - ALNFAC(NN-LL) - ALNFAC(KK-LL) -
     *      ALNFAC(MM-NN-KK+LL) + ALNFAC(MM-KK)
        P = P + LOG(SCALE)
        IF (P .LT. ELIMIT) THEN
          IFAULT = 3
          IF (LL .GT. REAL(NN*KK + NN + KK +1)/(MM +2)) CHYPER = ONE
          RETURN
        ELSE
          P = EXP(P)
        END IF
        ELSE
C
C     Scale up at this point.
C
        P = CHYPER * SCALE
        END IF
C
        PT = ZERO
```

```
          NL = N - L
          KL = K - L
          MNKL = M - N - KL + 1
          IF (L .LE. KL) THEN
          DO 7 I = 1, L-1
            P = P * REAL(L-I) * REAL(MNKL-I) /
     *                          (REAL(NL+I) * REAL(KL+I))
            PT = PT + P
      7     CONTINUE
          IF (P .EQ. ZERO) IFAULT = 3
          ELSE
          DIR = .NOT. DIR
          DO 9 J = 0, KL-1
            P = P * REAL(NL-J) * REAL(KL-J) / (REAL(L+J) * REAL(MNKL+J))
              PT = PT + P
      9     CONTINUE
          IF (P .EQ. ZERO) IFAULT = 3
          END IF
C
          IF (DIR) THEN
          CHYPER = CHYPER + (PT / SCALE)
          ELSE
          CHYPER = ONE - (PT / SCALE)
          END IF
C
        END IF
C
        END
```

```
      DOUBLE PRECISION FUNCTION GRADSOL(A, M, C, N)
C
C     TRANSLATION OF AMENDED VERSION OF APPLIED STATISTICS ALGORITHM
C     AS 153 (AS R52), VOL. 33, 363-366, 1984.
C     BY R.W. FAREBROTHER  (ORIGINALLY NAMED GRADSOL OR PAN)
C
C     GRADSOL EVALUATES THE PROBABILITY THAT A WEIGHTED SUM OF
C     SQUARED STANDARD NORMAL VARIATES DIVIDED BY X TIMES THE UNWEIGHTED
C     SUM IS LESS THAN A GIVEN CONSTANT, I.E. THAT
C     A1.U1**2 + A2.U2**2 + ... + AM.UM**2 <
C     X*(U1**2 + U2**2 + ... + UM**2) + C
C     WHERE THE U'S ARE STANDARD NORMAL VARIABLES.
C     FOR THE DURBIN-WATSON STATISTIC, X = DW, C = 0, AND
C     A ARE THE NON-ZERO EIGENVALUES OF THE "M*A" MATRIX.
C
C     THE ELEMENTS A(I) MUST BE ORDERED.  A(0) = X
C     N = THE NUMBER OF TERMS IN THE SERIES.   THIS DETERMINES THE
C     ACCURACY AND ALSO THE SPEED.   NORMALLY N SHOULD BE ABOUT 10-15.
C     --------------
C     ORIGINALLY FROM STATLIB.  REVISED 5/3/1996 BY CLINT CUMMINS:
C     1. DIMENSION A STARTING FROM 0  (FORTRAN 77)
C        IF THE USER DOES NOT INITIALIZE A(0) = X,
C        THERE WOULD BE UNPREDICTABLE RESULTS, SINCE A(0) IS ACCESSED
C        WHEN J2=0 FOR THE FINAL DO 60 LOOP.
C     2. USE X VARIABLE TO AGREE WITH PUBLISHED CODE
C     3. FIX BUG 2 LINES BELOW  DO 60 L2 = J2, NU, D
C        PROD = A(J2)  -->  PROD = A(L2)
C        (PRIOR TO THIS FIX, ONLY THE TESTS WITH M=3 WORKED CORRECTLY)
C     4. TRANSLATE TO UPPERCASE AND REMOVE TABS
C     TESTED SUCCESSFULLY ON THE FOLLOWING BENCHMARKS:
C     1. FAREBROTHER 1984 TABLE (X=0):
C        A           C   PROBABILITY
C        1,3,6       1   .0542
C        1,3,6       7   .4936
C        1,3,6      20   .8760
C        1,3,5,7,9   5   .0544
C        1,3,5,7,9  20   .4853
C        1,3,5,7,9  50   .9069
C        3,4,5,6,7   5   .0405
C        3,4,5,6,7  20   .4603
C        3,4,5,6,7  50   .9200
C     2. DURBIN-WATSON 1951/71 SPIRITS DATASET, FOR X=.2,.3,...,3.8, C=0
C        COMPARED WITH BETA APPROXIMATION  (M=66), A SORTED IN REVERSE ORDER
C     3. JUDGE, ET AL 2ND ED. P.399 DATASET, FOR X=.2,.3,...,3.8, C=0
C        COMPARED WITH BETA APPROXIMATION  (M=8), A SORTED IN EITHER ORDER
C
      INTEGER M, N
      DOUBLE PRECISION A(0:M), C, X
C
C     LOCAL VARIABLES
C
      INTEGER D, H, I, J1, J2, J3, J4, K, L1, L2, NU, N2
      DOUBLE PRECISION NUM, PIN, PROD, SGN, SUM, SUM1, U, V, Y
      DOUBLE PRECISION ZERO, ONE, HALF, TWO
      DATA ZERO/0.D0/, ONE/1.D0/, HALF/0.5D0/, TWO/2.D0/
C
C     SET NU = INDEX OF 1ST A(I) >= X.
C     ALLOW FOR THE A'S BEING IN REVERSE ORDER.
C
      IF (A(1) .GT. A(M)) THEN
        H = M
```

```
          K = -1
          I = 1
        ELSE
          H = 1
          K = 1
          I = M
        ENDIF
        X = A(0)
        DO 10 NU = H, I, K
          IF (A(NU) .GE. X) GO TO 20
   10   CONTINUE
C
C       IF ALL A'S ARE -VE AND C >= 0, THEN PROBABILITY = 1.
C
        IF (C .GE. ZERO) THEN
          GRADSOL = ONE
          RETURN
        ENDIF
C
C       SIMILARLY IF ALL THE A'S ARE +VE AND C <= 0, THEN PROBABILITY = 0.
C
   20   IF (NU .EQ. H .AND. C .LE. ZERO) THEN
          GRADSOL = ZERO
          RETURN
        ENDIF
C
        IF (K .EQ. 1) NU = NU - 1
        H = M - NU
        IF (C .EQ. ZERO) THEN
          Y = H - NU
        ELSE
          Y = C * (A(1) - A(M))
        ENDIF
C
        IF (Y .GE. ZERO) THEN
          D = 2
          H = NU
          K = -K
          J1 = 0
          J2 = 2
          J3 = 3
          J4 = 1
        ELSE
          D = -2
          NU = NU + 1
          J1 = M - 2
          J2 = M - 1
          J3 = M + 1
          J4 = M
        ENDIF
        PIN = TWO * DATAN(ONE) / N
        SUM = HALF * (K + 1)
        SGN = K / DBLE(N)
        N2 = N + N - 1
C
C         FIRST INTEGRALS
C
        DO 70 L1 = H-2*(H/2), 0, -1
          DO 60 L2 = J2, NU, D
            SUM1 = A(J4)
C           FIX BY CLINT CUMMINS 5/3/96
```

```
C          PROD = A(J2)
           PROD = A(L2)
           U = HALF * (SUM1 + PROD)
           V = HALF * (SUM1 - PROD)
           SUM1 = ZERO
           DO 50 I = 1, N2, 2
             Y = U - V * DCOS(DBLE(I)*PIN)
             NUM = Y - X
             PROD = DEXP(-C/NUM)
             DO 30 K = 1, J1
               PROD = PROD * NUM / (Y - A(K))
30           CONTINUE
             DO 40 K = J3, M
               PROD = PROD * NUM / (Y - A(K))
40           CONTINUE
             SUM1 = SUM1 + DSQRT(DABS(PROD))
50           CONTINUE
           SGN = -SGN
           SUM = SUM + SGN * SUM1
           J1 = J1 + D
           J3 = J3 + D
           J4 = J4 + D
60    CONTINUE
C
C        SECOND INTEGRAL.
C
         IF (D .EQ. 2) THEN
           J3 = J3 - 1
         ELSE
           J1 = J1 + 1
         ENDIF
         J2 = 0
         NU = 0
70  CONTINUE
C
       GRADSOL = SUM
       RETURN
       END
```

```
CSTART OF AS 154
      SUBROUTINE STARMA(IP, IQ, IR, NP, PHI, THETA, A, P, V, THETAB,
     $  XNEXT, XROW, RBAR, NRBAR, IFAULT)
C
C        ALGORITHM AS 154  APPL. STATIST. (1980) VOL.29, P.311
C
C        INVOKING THIS SUBROUTINE SETS THE VALUES OF V AND PHI, AND
C        OBTAINS THE INITIAL VALUES OF A AND P.
C        THIS ROUTINE IS NOT SUITABLE FOR USE WITH AN AR(1) PROCESS.
C        IN THIS CASE THE FOLLOWING INSTRUCTIONS SHOULD BE USED FOR
C        INITIALISATION.
C        V(1) = 1.0
C        A(1) = 0.0
C        P(1) = 1.0 / (1.0 - PHI(1) * PHI(1))
C
      REAL PHI(IR), THETA(IR), A(IR), P(NP), V(NP), THETAB(NP),
     $  XNEXT(NP), XROW(NP), RBAR(NRBAR), VJ, PHII, PHIJ, SSQERR,
     $  RECRES, YNEXT, ZERO, ONE
C
      DATA ZERO, ONE /0.0, 1.0/
C
C        CHECK FOR FAILURE INDICATION.
C
      IFAULT = 0
      IF (IP .LT. 0) IFAULT = 1
      IF (IQ .LT. 0) IFAULT = IFAULT + 2
      IF (IP .EQ. 0 .AND. IQ .EQ. 0) IFAULT = 4
      K = IQ + 1
      IF (K .LT. IP) K = IP
      IF (IR .NE. K) IFAULT = 5
      IF (NP .NE. IR * (IR + 1) / 2) IFAULT = 6
      IF (NRBAR .NE. NP * (NP - 1) / 2) IFAULT = 7
      IF (IR .EQ. 1) IFAULT = 8
      IF (IFAULT .NE. 0) RETURN
C
C        NOW SET A(0), V AND PHI.
C
      DO 10 I = 2, IR
      A(I) = ZERO
      IF (I .GT. IP) PHI(I) = ZERO
      V(I) = ZERO
      IF (I .LE. IQ + 1) V(I) = THETA(I - 1)
   10 CONTINUE
      A(1) = ZERO
      IF (IP .EQ. 0) PHI(1) = ZERO
      V(1) = ONE
      IND = IR
      DO 20 J = 2, IR
      VJ = V(J)
      DO 20 I = J, IR
      IND = IND + 1
      V(IND) = V(I) * VJ
   20 CONTINUE
C
C        NOW FIND P(0).
C
      IF (IP .EQ. 0) GOTO 300
C
C        THE SET OF EQUATIONS S * VEC(P(0)) = VEC(V)
C        IS SOLVED FOR VEC(P(0)).
C        S IS GENERATED ROW BY ROW IN THE ARRAY XNEXT.
```

```
C        THE ORDER OF ELEMENTS IN P IS CHANGED, SO AS TO
C        BRING MORE LEADING ZEROS INTO THE ROWS OF S,
C        HENCE ACHIEVING A REDUCTION OF COMPUTING TIME.
C
      IRANK = 0
      SSQERR = ZERO
      DO 40 I = 1, NRBAR
   40 RBAR(I) = ZERO
      DO 50 I = 1, NP
      P(I) = ZERO
      THETAB(I) = ZERO
      XNEXT(I) = ZERO
   50 CONTINUE
      IND = 0
      IND1 = 0
      NPR = NP - IR
      NPR1 = NPR + 1
      INDJ = NPR1
      IND2 = NPR
      DO 110 J = 1, IR
      PHIJ = PHI(J)
      XNEXT(INDJ) = ZERO
      INDJ = INDJ + 1
      INDI = NPR1 + J
      DO 110 I = J, IR
      IND = IND + 1
      YNEXT = V(IND)
      PHII = PHI(I)
      IF (J .EQ. IR) GOTO 100
      XNEXT(INDJ) = -PHII
      IF (I .EQ. IR) GOTO 100
      XNEXT(INDI) = XNEXT(INDI) - PHIJ
      IND1 = IND1 + 1
      XNEXT(IND1) = -ONE
  100 XNEXT(NPR1) = -PHII * PHIJ
      IND2 = IND2 + 1
      IF (IND2 .GT. NP) IND2 = 1
      XNEXT(IND2) = XNEXT(IND2) + ONE
      CALL INCLU2(NP, NRBAR, ONE, XNEXT, XROW, YNEXT,
     $  P, RBAR, THETAB, SSQERR, RECRES, IRANK, IFAIL)
C
C        NO NEED TO CHECK IFAIL AS WEIGHT = 1.0
C
      XNEXT(IND2) = ZERO
      IF (I .EQ. IR) GOTO 110
      XNEXT(INDI) = ZERO
      INDI = INDI + 1
      XNEXT(IND1) = ZERO
  110 CONTINUE
      CALL REGRES(NP, NRBAR, RBAR, THETAB, P)
C
C        NOW RE-ORDER P.
C
      IND = NPR
      DO 200 I = 1, IR
      IND = IND + 1
      XNEXT(I) = P(IND)
  200 CONTINUE
      IND = NP
      IND1 = NPR
      DO 210 I = 1, NPR
```

```
      P(IND) = P(IND1)
      IND = IND - 1
      IND1 = IND1 - 1
  210 CONTINUE
      DO 220 I = 1, IR
  220 P(I) = XNEXT(I)
      RETURN
C
C        P(0) IS OBTAINED BY BACKSUBSTITUTION FOR
C        A MOVING AVERAGE PROCESS.
C
  300 INDN = NP + 1
      IND = NP + 1
      DO 310 I = 1, IR
      DO 310 J = 1, I
      IND = IND - 1
      P(IND) = V(IND)
      IF (J .EQ. 1) GOTO 310
      INDN = INDN - 1
      P(IND) = P(IND) + P(INDN)
  310 CONTINUE
      RETURN
      END
C
      SUBROUTINE KARMA(IP, IQ, IR, NP, PHI, THETA, A, P,
     $  V, N, W, RESID, SUMLOG, SSQ, IUPD, DELTA, E, NIT)
C
C        ALGORITHM AS 154.1  APPL. STATIST. (1980) VOL.29, P.311
C
C        INVOKING THIS SUBROUTINE UPDATES A, P, SUMLOG AND SSQ BY
C        INCLUSION OF DATA VALUES W(1) TO W(N). THE CORRESPONDING
C        VALUES OF RESID ARE ALSO OBTAINED.
C        WHEN FT IS LESS THAN (1 + DELTA), QUICK RECURSIONS ARE USED.
C
      REAL PHI(IR), THETA(IR), A(IR), P(NP), V(NP), W(N), RESID(N),
     $  E(IR), SUMLOG, SSQ, DELTA, WNEXT, A1, DT, ET, FT, UT, G,
     $  ZERO, ZLOG, ZSQRT
C
      DATA ZERO /0.0/
C
      ZLOG(G) = ALOG(G)
      ZSQRT(G) = SQRT(G)
C
      IR1 = IR - 1
      DO 10 I = 1, IR
   10 E(I) = ZERO
      INDE = 1
C
C        FOR NON-ZERO VALUES OF NIT, PERFORM QUICK RECURSIONS.
C
      IF (NIT .NE. 0) GOTO 600
      DO 500 I = 1, N
      WNEXT = W(I)
C
C        PREDICTION.
C
      IF (IUPD .EQ. 1 .AND. I .EQ. 1) GOTO 300
C
C        HERE DT = FT - 1.0
C
      DT = ZERO
```

```
          IF (IR .NE. 1) DT = P(IR + 1)
          IF (DT .LT. DELTA) GOTO 610
          A1 = A(1)
          IF (IR .EQ. 1) GOTO 110
          DO 100 J = 1, IR1
      100 A(J) = A(J + 1)
      110 A(IR) = ZERO
          IF (IP .EQ. 0) GOTO 200
          DO 120 J = 1, IP
      120 A(J) = A(J) + PHI(J) * A1
      200 IND = 0
          INDN = IR
          DO 210 L = 1, IR
          DO 210 J = L, IR
          IND = IND + 1
          P(IND) = V(IND)
          IF (J .EQ. IR) GOTO 210
          INDN = INDN + 1
          P(IND) = P(IND) + P(INDN)
      210 CONTINUE
C
C         UPDATING.
C
      300 FT = P(1)
          UT = WNEXT - A(1)
          IF (IR .EQ. 1) GOTO 410
          IND = IR
          DO 400 J = 2, IR
          G = P(J) / FT
          A(J) = A(J) + G * UT
          DO 400 L = J, IR
          IND = IND + 1
          P(IND) = P(IND) - G * P(L)
      400 CONTINUE
      410 A(1) = WNEXT
          DO 420 L = 1, IR
      420 P(L) = ZERO
          RESID(I) = UT / ZSQRT(FT)
          E(INDE) = RESID(I)
          INDE = INDE + 1
          IF (INDE .GT. IQ) INDE = 1
          SSQ = SSQ + UT * UT / FT
          SUMLOG = SUMLOG + ZLOG(FT)
      500 CONTINUE
          NIT = N
          RETURN
C
C         QUICK RECURSIONS
C
      600 I = 1
      610 NIT = I - 1
          DO 650 II = I, N
          ET = W(II)
          INDW = II
          IF (IP .EQ. 0) GOTO 630
          DO 620 J = 1, IP
          INDW = INDW - 1
          IF (INDW .LT. 1) GOTO 630
          ET = ET - PHI(J) * W(INDW)
      620 CONTINUE
      630 IF (IQ .EQ. 0) GOTO 645
```

```
      DO 640 J = 1, IQ
      INDE = INDE - 1
      IF (INDE .EQ. 0) INDE = IQ
      ET = ET - THETA(J) * E(INDE)
  640 CONTINUE
  645 E(INDE) = ET
      RESID(II) = ET
      SSQ = SSQ + ET * ET
      INDE = INDE + 1
      IF (INDE .GT. IQ) INDE = 1
  650 CONTINUE
      RETURN
      END
C
      SUBROUTINE KALFOR(M, IP, IR, NP, PHI, A, P, V, WORK)
C
C        ALGORITHM AS 154.2  APPL. STATIST. (1980) VOL.29, P.311
C
C        INVOKING THIS SUBROUTINE OBTAINS PREDICTIONS
C        OF A AND P, M STEPS AHEAD.
C
      REAL PHI(IR), A(IR), P(NP), V(NP), WORK(IR), DT,
     $  A1, PHII, PHIJ, PHIJDT, ZERO
C
      DATA ZERO /0.0/
C
      IR1 = IR - 1
      DO 300 L = 1, M
C
C        PREDICT A.
C
      A1 = A(1)
      IF (IR .EQ. 1) GOTO 110
      DO 100 I = 1, IR1
  100 A(I) = A(I + 1)
  110 A(IR) = ZERO
      IF (IP .EQ. 0) GOTO 200
      DO 120 J = 1, IP
  120 A(J) = A(J) + PHI(J) * A1
C
C        PREDICT P.
C
  200 DO 210 I = 1, IR
  210 WORK(I) = P(I)
      IND = 0
      IND1 = IR
      DT = P(1)
      DO 220 J = 1, IR
      PHIJ = PHI(J)
      PHIJDT = PHIJ * DT
      DO 220 I = J, IR
      IND = IND + 1
      PHII = PHI(I)
      P(IND) = V(IND) + PHII * PHIJDT
      IF (J .LT. IR) P(IND) = P(IND) + WORK(J + 1) * PHII
      IF (I .EQ. IR) GOTO 220
      IND1 = IND1 + 1
      P(IND) = P(IND) + WORK(I + 1) * PHIJ + P(IND1)
  220 CONTINUE
  300 CONTINUE
      RETURN
```

```
      END
C
      SUBROUTINE INCLU2(NP, NRBAR, WEIGHT, XNEXT, XROW, YNEXT, D, RBAR,
     $  THETAB, SSQERR, RECRES, IRANK, IFAULT)
C
C        ALGORITHM AS 154.3  APPL. STATIST. (1980) VOL.29, P.311
C
C        FORTRAN VERSION OF REVISED VERSION OF ALGORITHM AS 75.1
C        APPL. STATIST. (1974) VOL.23, P.448
C        SEE REMARK AS R17 APPL. STATIST. (1976) VOL.25, P.323
C
      REAL XNEXT(NP), XROW(NP), D(NP), RBAR(NRBAR), THETAB(NP),
     $  WEIGHT, YNEXT, SSQERR, RECRES, WT, Y, DI, DPI, XI, XK,
     $  CBAR, SBAR, RBTHIS, ZERO, ZSQRT
C
      DATA ZERO /0.0/
C
      ZSQRT(Y) = SQRT(Y)
C
C        INVOKING THIS SUBROUTINE UPDATES D, RBAR, THETAB, SSQERR
C        AND IRANK BY THE INCLUSION OF XNEXT AND YNEXT WITH A
C        SPECIFIED WEIGHT. THE VALUES OF XNEXT, YNEXT AND WEIGHT WILL
C        BE CONSERVED. THE CORRESPONDING VALUE OF RECRES IS CALCULATED.
C
      Y = YNEXT
      WT = WEIGHT
      DO 10 I = 1, NP
   10 XROW(I) = XNEXT(I)
      RECRES = ZERO
      IFAULT = 1
      IF (WT .LE. ZERO) RETURN
      IFAULT = 0
C
      ITHISR = 0
      DO 50 I = 1, NP
      IF (XROW(I) .NE. ZERO) GOTO 20
      ITHISR = ITHISR + NP - I
      GOTO 50
   20 XI = XROW(I)
      DI = D(I)
      DPI = DI + WT * XI * XI
      D(I) = DPI
      CBAR = DI / DPI
      SBAR = WT * XI / DPI
      WT = CBAR * WT
      IF (I .EQ. NP) GOTO 40
      I1 = I + 1
      DO 30 K = I1, NP
      ITHISR = ITHISR + 1
      XK = XROW(K)
      RBTHIS = RBAR(ITHISR)
      XROW(K) = XK - XI * RBTHIS
      RBAR(ITHISR) = CBAR * RBTHIS + SBAR * XK
   30 CONTINUE
   40 XK = Y
      Y = XK - XI * THETAB(I)
      THETAB(I) = CBAR * THETAB(I) + SBAR * XK
      IF (DI .EQ. ZERO) GOTO 100
   50 CONTINUE
      SSQERR = SSQERR + WT * Y * Y
      RECRES = Y * ZSQRT(WT)
```

```
      RETURN
  100 IRANK = IRANK + 1
      RETURN
      END
C
      SUBROUTINE REGRES(NP, NRBAR, RBAR, THETAB, BETA)
C
C        ALGORITHM AS 154.4  APPL. STATIST. (1980) VOL.29, P.311
C
C        REVISED VERSION OF ALGORITHM AS 75.4
C        APPL. STATIST. (1974) VOL.23, P.448
C        INVOKING THIS SUBROUTINE OBTAINS BETA BY BACKSUBSTITUTION
C        IN THE TRIANGULAR SYSTEM RBAR AND THETAB.
C
      REAL RBAR(NRBAR), THETAB(NP), BETA(NP), BI
      ITHISR = NRBAR
      IM = NP
      DO 50 I = 1, NP
      BI = THETAB(IM)
      IF (IM .EQ. NP) GOTO 30
      I1 = I - 1
      JM = NP
      DO 10 J = 1, I1
      BI = BI - RBAR(ITHISR) * BETA(JM)
      ITHISR = ITHISR - 1
      JM = JM - 1
   10 CONTINUE
   30 BETA(IM) = BI
      IM = IM - 1
   50 CONTINUE
      RETURN
      END
CEND OF AS 154
```

```
      REAL FUNCTION QF(ALB,ANC,N,IRR,SIGMA,CC,LIM1,ACC,ITH,TRACE,IFAULT)
C
C     ALGORITHM AS 155  APPL. STATIST. (1980) VOL.29, NO.3
C
C     Distribution of a linear combination of non-central chi-squared
C     random variables.
C
      INTEGER IRR,LIM1,IFAULT
      REAL SIGMA,CC,ACC
      REAL TRACE(7),ALB(IRR),ANC(IRR)
      INTEGER N(IRR),ITH(IRR)
      INTEGER J,NJ,NT,NTM
      REAL ACC1,ALMX,XLIM,XNT,XNTM
      REAL UTX,TAUSQ,SD,AINTV,AINTV1,X,UP,UN,D1,D2,ALJ,ANCJ
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,HALF,ONE,TWO,FOUR,
     1  SIXTN,FOURP5,PT07,PT2,QUART,TEN,PT33,PT67,PT75,ONEPT5,THREE,
     2  ONEPT1
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO /0.0/,HALF/0.5/,ONE/1.0/,TWO/2.0/,FOUR/4.0/,SIXTN/16.0/,
     1   FOURP5/4.5/,PT07/0.07/,PT2/0.2/,QUART/0.25/,TEN/10.0/,
     2   PT33/0.33/,PT67/0.67/,PT75/0.75/,ONEPT5/1.5/,THREE/3.0/,
     3   ONEPT1/1.1/
C
      IROUND(X) = INT(X + SIGN(HALF,X))
C
C     Setting constants in COMMON.   ALN28 = ln(2) / 8.
C
      PI = 3.14159265358979
      ALN28 = 0.0866
C
      C = CC
      IR = IRR
      LIM = LIM1
      DO 10 J = 1,7
        TRACE(J) = ZERO
 10   CONTINUE
      IFAULT = 0
      ICOUNT = 0
      AINTL = ZERO
      ERSM = ZERO
      QF = -ONE
      ACC1 = ACC
      NDTSRT = .TRUE.
      FAIL = .FALSE.
C
C     Find mean, sd, max & min of ALB.
C     Check that parameter values are valid.
C
      XLIM = LIM
      SIGSQ = SIGMA**2
      SD = SIGSQ
      ALMAX = ZERO
      ALMIN = ZERO
      AMEAN = ZERO
      J = 1
 20   IF (.NOT.(J.LE.IR)) GO TO 60
      NJ = N(J)
```

```
      ALJ = ALB(J)
      ANCJ = ANC(J)
      IF (.NOT.(NJ.LT.0.OR.ANCJ.LT.ZERO)) GO TO 30
      IFAULT = 3
      GO TO 260
 30   SD = SD + ALJ**2*(2*NJ + FOUR*ANCJ)
      AMEAN = AMEAN + ALJ*(NJ + ANCJ)
      IF (.NOT.(ALMAX.LT.ALJ)) GO TO 40
      ALMAX = ALJ
      GO TO 50
 40   IF (.NOT.(ALMIN.GT.ALJ)) GO TO 50
      ALMIN = ALJ
 50   J = J + 1
      GO TO 20
 60   IF (.NOT.(SD.EQ.ZERO)) GO TO 80
      IF (.NOT.(C.GT.ZERO)) GO TO 70
      QF = ONE
      GO TO 260
 70   QF = ZERO
      GO TO 260
 80   IF (.NOT.(ALMIN.EQ.ZERO.AND.ALMAX.EQ.ZERO.AND.SIGMA.EQ.ZERO))
     1  GO TO 90
      IFAULT = 3
      GO TO 260
 90   SD = SQRT(SD)
      IF (.NOT.(ALMAX.LT.-ALMIN)) GO TO 100
      ALMX = -ALMIN
      GO TO 110
 100  ALMX = ALMAX
C
C     Starting values for FINDU * CTFF.
C
 110  UTX = SIXTN/SD
      UP = FOURP5/SD
      UN = -UP
C
C     Truncation point with no convergence factor.
C
      CALL FINDU (N,ALB,ANC,UTX,HALF*ACC1)
C
C     Does convergence factor help ?
C
      IF (.NOT.(C.NE.ZERO.AND.ALMX.GT.PT07*SD)) GO TO 130
      TAUSQ = QUART*ACC1/CFE(N,ALB,ANC,ITH,C)
      IF (.NOT.(FAIL)) GO TO 120
      FAIL = .FALSE.
      GO TO 130
 120  IF (.NOT.(TRUNCN(N,ALB,ANC,UTX,TAUSQ).LT.PT2*ACC1)) GO TO 130
      SIGSQ = SIGSQ + TAUSQ
      CALL FINDU (N,ALB,ANC,UTX,QUART*ACC1)
      TRACE(6) = SQRT(TAUSQ)
 130  TRACE(5) = UTX
      ACC1 = HALF*ACC1
C
C     Find 'range' of distribution, quit if outside of this.
C
 140  D1 = CTFF(N,ALB,ANC,ACC1,UP) - C
      IF (.NOT.(D1.LT.ZERO)) GO TO 150
      QF = ONE
      GO TO 260
 150  D2 = C - CTFF(N,ALB,ANC,ACC1,UN)
```

```
      IF (.NOT.(D2.LT.ZERO)) GO TO 160
      QF = ZERO
      GO TO 260
C
C     Find integration interval.
C
 160  IF (.NOT.(D1.GT.D2)) GO TO 170
      AINTV = D1
      GO TO 180
 170  AINTV = D2
 180  AINTV = TWO*PI/AINTV
C
C     Calculate number of terms required for main & auxiliary
C     integrations.
C
      XNT = UTX/AINTV
      XNTM = THREE/SQRT(ACC1)
      IF (.NOT.(XNT.GT.XNTM*ONEPT5)) GO TO 220
      IF (.NOT.(XNTM.GT.XLIM)) GO TO 190
      IFAULT = 1
      GO TO 260
C
C     Parameters for auxiliary integration.
C
 190  NTM = IROUND(XNTM)
      AINTV1 = UTX/XNTM
      X = TWO*PI/AINTV1
      IF (.NOT.(X.LE.ABS(C))) GO TO 200
      GO TO 220
C
C     Calculate convergence factor.
C
 200  TAUSQ = CFE(N,ALB,ANC,ITH,C - X) + CFE(N,ALB,ANC,ITH,C + X)
      TAUSQ = PT33*ACC1/(ONEPT1*TAUSQ)
      IF (.NOT.(FAIL)) GO TO 210
      GO TO 220
 210  ACC1 = PT67*ACC1
C
C     Auxiliary integration.
C
      CALL INTEGR (N,ALB,ANC,NTM,AINTV1,TAUSQ,.FALSE.)
      XLIM = XLIM - XNTM
      SIGSQ = SIGSQ + TAUSQ
      TRACE(3) = TRACE(3) + 1
      TRACE(2) = TRACE(2) + NTM + 1
C
C     Find truncation point with new convergence factor.
C
      CALL FINDU (N,ALB,ANC,UTX,QUART*ACC1)
      ACC1 = PT75*ACC1
      GO TO 140
C
C     Main integration.
C
 220  TRACE(4) = AINTV
      IF (.NOT.(XNT.GT.XLIM)) GO TO 230
      IFAULT = 1
      GO TO 260
 230  NT = IROUND(XNT)
      CALL INTEGR (N,ALB,ANC,NT,AINTV,ZERO,.TRUE.)
      TRACE(3) = TRACE(3) + 1
```

```
      TRACE(2) = TRACE(2) + NT + 1
      QF = HALF - AINTL
      TRACE(1) = ERSM
      UP = ERSM
C
C     Test whether round-off error could be significant.
C     Allow for radix 8 or 16 machines.
C
      X = UP + ACC/TEN
      J = 1
 240  IF (.NOT.(J.LE.8)) GO TO 260
      IF (.NOT.(J*X.EQ.J*UP)) GO TO 250
      IFAULT = 2
 250  J = J*2
      GO TO 240
 260  TRACE(7) = ICOUNT
      RETURN
      END
C
      SUBROUTINE COUNTR
C
C     Count number of calls to ERRBD, TRUNCN & CFE.
C
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
C
      ICOUNT = ICOUNT + 1
      IF (.NOT.(ICOUNT.GT.LIM)) GO TO 20
      WRITE (6,10)
 10   FORMAT (' qf: cannot locate integration parameters'/)
      STOP
 20   RETURN
      END
C
      REAL FUNCTION ALOG1(X,FIRST)
C
C     If FIRST then return ln(1 + x) else ln(1 + x) - x.
C
      REAL X
      LOGICAL FIRST
      REAL S,S1,TERM,Y,AK,PT1,ONE,TWO,THREE
      DATA PT1/0.1/,ONE/1.0/,TWO/2.0/,THREE/3.0/
C
      F1(I) = S + TERM/AK
C
      IF (.NOT.(ABS(X).GT.PT1)) GO TO 20
      IF (.NOT.(FIRST)) GO TO 10
      ALOG1 = LOG(ONE + X)
      GO TO 70
 10   ALOG1 = LOG(ONE + X) - X
      GO TO 70
 20   Y = X/(TWO + X)
      TERM = TWO*Y**3
      AK = THREE
      IF (.NOT.(FIRST)) GO TO 30
      S = TWO
      GO TO 40
```

```
 30   S = -X
 40   S = S*Y
      Y = Y**2
      S1 = F1(0)
 50   IF (.NOT.(S1.NE.S)) GO TO 60
      AK = AK + TWO
      TERM = TERM*Y
      S = S1
      S1 = F1(0)
      GO TO 50
 60   ALOG1 = S
 70   RETURN
      END
C
      REAL FUNCTION EXP1(X)
      REAL X
      REAL ZERO,NEG50
      DATA ZERO/0.0/,NEG50/-50.0/
C
      IF (.NOT.(X.LT.NEG50)) GO TO 10
      EXP1 = ZERO
      GO TO 20
 10   EXP1 = EXP(X)
 20   RETURN
      END
C
      SUBROUTINE ORDER (ALB,ITH)
C
C     Find order of absolute values of ALB.
C
      REAL ALB(*)
      INTEGER ITH(*)
      INTEGER J,K,K1,ITHK
      REAL ALJ
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
C
      J = 1
 10   IF (.NOT.(J.LE.IR)) GO TO 60
      ALJ = ABS(ALB(J))
      K = J - 1
 20   IF (.NOT.(K.GT.0)) GO TO 40
      ITHK = ITH(K)
      K1 = K + 1
      IF (.NOT.(ALJ.GT.ABS(ALB(ITHK)))) GO TO 50
      ITH(K1) = ITHK
      GO TO 30
 30   K = K - 1
      GO TO 20
 40   K = 0
      K1 = 1
 50   ITH(K1) = J
      J = J + 1
      GO TO 10
 60   NDTSRT = .FALSE.
      RETURN
      END
```

```
C
      REAL FUNCTION ERRBD(N,ALB,ANC,UU,CX)
C
C     Find bound on tail probability using mgf.
C     Cut-off point returned to CX.
C
      REAL U,UU,CX
      INTEGER N(*)
      REAL ALB(*),ANC(*)
      REAL SUM1,ALJ,ANCJ,X,Y,CONST
      INTEGER J,NJ
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA HALF/0.5/,ONE/1.0/,TWO/2.0/
C
      CALL COUNTR
      U = UU
      CONST = U*SIGSQ
      SUM1 = U*CONST
      U = TWO*U
      J = IR
 10   IF (.NOT.(J.GT.0)) GO TO 20
      NJ = N(J)
      ALJ = ALB(J)
      ANCJ = ANC(J)
      X = U*ALJ
      Y = ONE - X
      CONST = CONST + ALJ*(ANCJ/Y + NJ)/Y
      SUM1 = SUM1 + ANCJ*(X/Y)**2
      SUM1 = SUM1 + NJ*(X**2/Y + ALOG1(-X,.FALSE.))
      J = J - 1
      GO TO 10
 20   ERRBD = EXP1(-HALF*SUM1)
      CX = CONST
      RETURN
      END
C
      REAL FUNCTION CTFF(N,ALB,ANC,ACCX,UPN)
C
C     Find CTFF so that P(QF > CTFF) < ACCX if UPN > 0;
C     P(QF < CTFF) < ACCX otherwise.
C
      REAL ACCX,UPN
      INTEGER N(*)
      REAL ALB(*),ANC(*)
      REAL U1,U2,U,RB,CONST,C1,C2
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0/,ONE/1.0/,TWO/2.0/
C
      F1(I) = U2/(ONE + U2*RB)
      F2(I) = (C1 - AMEAN)/(C2 - AMEAN)
C
```

```
      U2 = UPN
      U1 = ZERO
      C1 = AMEAN
      IF (.NOT.(U2.GT.ZERO)) GO TO 10
      RB = ALMAX
      GO TO 20
 10   RB = ALMIN
 20   RB = TWO*RB
      U = F1(0)
 30   IF (.NOT.(ERRBD(N,ALB,ANC,U,C2).GT.ACCX)) GO TO 40
      U1 = U2
      C1 = C2
      U2 = TWO*U2
      U = F1(0)
      GO TO 30
 40   U = F2(0)
 50   IF (.NOT.(U.LT.0.9)) GO TO 80
      U = (U1 + U2)/TWO
      IF (.NOT.(ERRBD(N,ALB,ANC,U/(ONE + U*RB),CONST).GT.ACCX)) GO TO 60
      U1 = U
      C1 = CONST
      GO TO 70
 60   U2 = U
      C2 = CONST
 70   U = F2(0)
      GO TO 50
 80   CTFF = C2
      UPN = U2
      RETURN
      END
C
      REAL FUNCTION TRUNCN(N,ALB,ANC,UU,TAUSQ)
C
C     Bound integration error due to truncation at U.
C
      REAL U,UU,TAUSQ
      INTEGER N(*)
      REAL ALB(*),ANC(*)
      REAL SUM1,SUM2,PROD1,PROD2,PROD3,ALJ,ANCJ,X,Y,ERR1,ERR2
      INTEGER J,NJ,NS
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,QUART,HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0/,QUART/0.25/,HALF/0.5/,ONE/1.0/,TWO/2.0/
C
      CALL COUNTR
      U = UU
      SUM1 = ZERO
      PROD2 = ZERO
      PROD3 = ZERO
      NS = 0
      SUM2 = (SIGSQ + TAUSQ)*U**2
      PROD1 = TWO*SUM2
      U = TWO*U
      J = 1
 10   IF (.NOT.(J.LE.IR)) GO TO 40
      ALJ = ALB(J)
      ANCJ = ANC(J)
```

```
      NJ = N(J)
      X = (U*ALJ)**2
      SUM1 = SUM1 + ANCJ*X/(ONE + X)
      IF (.NOT.(X.GT.ONE)) GO TO 20
      PROD2 = PROD2 + NJ*LOG(X)
      PROD3 = PROD3 + NJ*ALOG1(X,.TRUE.)
      NS = NS + NJ
      GO TO 30
 20   PROD1 = PROD1 + NJ*ALOG1(X,.TRUE.)
 30   J = J + 1
      GO TO 10
 40   SUM1 = HALF*SUM1
      PROD2 = PROD1 + PROD2
      PROD3 = PROD1 + PROD3
      X = EXP1(-SUM1 - QUART*PROD2)/PI
      Y = EXP1(-SUM1 - QUART*PROD3)/PI
      IF (.NOT.(NS.EQ.0)) GO TO 50
      ERR1 = ONE
      GO TO 60
 50   ERR1 = X*TWO/NS
 60   IF (.NOT.(PROD3.GT.ONE)) GO TO 70
      ERR2 = 2.5*Y
      GO TO 80
 70   ERR2 = ONE
 80   IF (.NOT.(ERR2.LT.ERR1)) GO TO 90
      ERR1 = ERR2
 90   X = HALF*SUM2
      IF (.NOT.(X.LE.Y)) GO TO 100
      ERR2 = ONE
      GO TO 110
 100  ERR2 = Y/X
 110  IF (.NOT.(ERR1.LT.ERR2)) GO TO 120
      TRUNCN = ERR1
      GO TO 130
 120  TRUNCN = ERR2
 130  RETURN
      END
C
      SUBROUTINE FINDU (N,ALB,ANC,UTX,ACCX)
C
C     Find U such that TRUNCN(U) < ACCX & TRUNCN(U / 1.2) > ACCX.
C
      REAL UTX,ACCX
      INTEGER N(*)
      REAL ALB(*),ANC(*)
      REAL U,UT
      REAL DIVIS(4)
      INTEGER I
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,FOUR
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA DIVIS/2.0,1.4,1.2,1.1/, FOUR/4.0/
C
      UT = UTX
      U = UT/FOUR
      IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).GT.ACCX)) GO TO 20
      U = UT
 10   IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).GT.ACCX)) GO TO 40
```

```
      UT = UT*FOUR
      U = UT
      GO TO 10
 20   UT = U
      U = U/FOUR
 30   IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).LE.ACCX)) GO TO 40
      UT = U
      U = U/FOUR
      GO TO 30
 40   DO 50 I = 1,4
        U = UT/DIVIS(I)
        IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).LE.ACCX)) GO TO 50
        UT = U
 50   CONTINUE
      UTX = UT
      RETURN
      END
C
      SUBROUTINE INTEGR (N,ALB,ANC,NTERM,AINTRV,TAUSQ,MAIN)
C
C     Carry out integration with NTERM terms, at interval AINTRV.
C     If not MAIN then multiply integrand by  1 - exp(-0.5 * TAUSQ *
C     U**2).
C
      INTEGER NTERM
      REAL AINTRV,TAUSQ
      LOGICAL MAIN
      INTEGER N(*)
      REAL ALB(*),ANC(*)
      REAL AINPI,U,SUM1,SUM2,SUM3,X,Y,Z
      INTEGER K,J,NJ
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,QUART,HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA QUART/0.25/,HALF/0.5/,ONE/1.0/,TWO/2.0/
C
      AINPI = AINTRV/PI
      K = NTERM
 10   IF (.NOT.(K.GE.0)) GO TO 50
      U = (K + HALF)*AINTRV
      SUM1 = -TWO*U*C
      SUM2 = ABS(SUM1)
      SUM3 = -HALF*SIGSQ*U**2
      J = IR
 20   IF (.NOT.(J.GT.0)) GO TO 30
      NJ = N(J)
      X = TWO*ALB(J)*U
      Y = X**2
      SUM3 = SUM3 - QUART*NJ*ALOG1(Y,.TRUE.)
      Y = ANC(J)*X/(ONE + Y)
      Z = NJ*ATAN(X) + Y
      SUM1 = SUM1 + Z
      SUM2 = SUM2 + ABS(Z)
      SUM3 = SUM3 - HALF*X*Y
      J = J - 1
      GO TO 20
 30   X = AINPI*EXP1(SUM3)/U
      IF (.NOT.(.NOT.MAIN)) GO TO 40
```

```
      X = X*(ONE - EXP1(-HALF*TAUSQ*U**2))
 40   SUM1 = SIN(HALF*SUM1)*X
      SUM2 = HALF*SUM2*X
      AINTL = AINTL + SUM1
      ERSM = ERSM + SUM2
      K = K - 1
      GO TO 10
 50   RETURN
      END
C
      REAL FUNCTION CFE(N,ALB,ANC,ITH,X)
C
C     Coefficient of TAUSQ in error when convergence factor of
C     exp(-0.5 * TAUSQ * U**2) is used when df is evaluated at X.
C
      REAL X
      INTEGER N(*),ITH(*)
      REAL ALB(*),ANC(*)
      REAL AXL,AXL1,AXL2,SXL,SUM1,ALJ
      INTEGER J,K,IT,ITK
      DOUBLE PRECISION AINTL,ERSM
      REAL PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,ONE,FOUR,HUNDRD
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0/,ONE/1.0/,FOUR/4.0/,HUNDRD/100.0/
C
      CALL COUNTR
      IF (.NOT.(NDTSRT)) GO TO 10
      CALL ORDER (ALB,ITH)
 10   AXL = ABS(X)
      SXL = SIGN(ONE,X)
      SUM1 = ZERO
      J = IR
 20   IF (.NOT.(J.GT.0)) GO TO 70
      IT = ITH(J)
      IF (.NOT.(ALB(IT)*SXL.GT.ZERO)) GO TO 60
      ALJ = ABS(ALB(IT))
      AXL1 = AXL - ALJ*(N(IT) + ANC(IT))
      AXL2 = ALJ/ALN28
      IF (.NOT.(AXL1.GT.AXL2)) GO TO 30
      AXL = AXL1
      GO TO 60
 30   IF (.NOT.(AXL.GT.AXL2)) GO TO 40
      AXL = AXL2
 40   SUM1 = (AXL - AXL1)/ALJ
      K = J - 1
 50   IF (.NOT.(K.GT.0)) GO TO 70
      ITK = ITH(K)
      SUM1 = SUM1 + (N(ITK) + ANC(ITK))
      K = K - 1
      GO TO 50
 60   J = J - 1
      GO TO 20
 70   IF (.NOT.(SUM1.GT.HUNDRD)) GO TO 80
      CFE = ONE
      FAIL = .TRUE.
      GO TO 90
 80   CFE = 2**(SUM1/FOUR)/(PI*AXL**2)
 90   RETURN
```

```
END
```

```
      SUBROUTINE UDRUNS(X, N, UV, DV, IFAULT)
C
C        ALGORITHM AS 157 APPL. STATIST. (1981), VOL. 30, NO. 1
C
C        THE RUNS-UP AND FUNS-DOWN TEST
C
      INTEGER UCOUNT, DCOUNT, RU, RD
      DIMENSION X(N), A(6, 6), B(6), UCOUNT(6), DCOUNT(6)
C
C        SET-UP THE A AND B MATRICES
C
      DATA
     *  A(1, 1), A(1, 2), A(1, 3), A(1, 4), A(1, 5), A(1, 6), A(2, 2),
     *  A(2, 3), A(2, 4), A(2, 5), A(2, 6), A(3, 3), A(3, 4), A(3, 5),
     *  A(3, 6), A(4, 4), A(4, 5), A(4, 6), A(5, 5), A(5, 6), A(6, 6)
     *  /4529.4, 9044.9, 13568.0, 18091.0, 22615.0, 27892.0, 18097.0,
     *  27139.0, 36187.0, 45234.0, 55789.0, 40721.0, 54281.0, 67852.0,
     *  83685.0, 72414.0, 90470.0, 111580.0, 113262.0, 139476.0,
     *  172860.0/
      IFAULT = 0
      IF (N .LT. 4000) GOTO 500
      DO 1 J = 2, 6
        J1 = J - 1
        DO 1 I = 1, J1
          A(J, I) = A(I, J)
    1 CONTINUE
      B(1) = 1.0 / 6.0
      B(2) = 5.0 / 24.0
      B(3) = 11.0 / 120.0
      B(4) = 19.0 / 720.0
      B(5) = 29.0 / 5040.0
      B(6) = 1.0 / 840.0
C
      DO 100 I = 1, 6
        UCOUNT(I) = 0
        DCOUNT(I) = 0
  100 CONTINUE
C
C        THE LOOP THAT ENDS AT LINE 300 DETERMINES THE NUMBER OF
C        RUNS-UP AND RUNS-DOWN OF LENGTH I FOR I=1(1)5 AND THE NUMBER
C        OF RUNS-UP AND RUNS-DOWN OF LENGTH GREATER OR EQUAL TO 6
C
      RU = 1
      RD = 1
      DO 300 J = 2, N
C
C        THE FOLLOWING STATEMENT TESTS FOR BOTH RUNS-UP AND
C        RUNS-DOWN BREAK-POINTS.  IF A RUN-DOWN BREAK-POINT IS
C        DETECTED - GOTO 200, OTHERWISE - GOTO 150.
C        ( RU AND RD ACT AS LOCAL COUNTERS FOR THE NUMBER OF
C        RUNS-UP AND RUNS-DOWN RESPECTIVELY.)
C        A TEST IS ALSO MADE FOR DATA TIES BEWTEEN ADJACENT
C        ELEMENTS.  IF A TIE IS DETECTED - GOTO 600.
C
        IF (X(J) - X(J - 1)) 150, 600, 200
  150   UCOUNT(RU) = UCOUNT(RU) + 1
        RU = 1
        IF (RD .LT. 6) RD = RD + 1
        GOTO 300
  200   DCOUNT(RD) = DCOUNT(RD) + 1
        RD = 1
```

```
            IF (RU .LT. 6) RU = RU + 1
      300 CONTINUE
          UCOUNT(RU) = UCOUNT(RU) + 1
          DCOUNT(RD) = DCOUNT(RD) + 1
C
C         CALCULATE THE TEST STATISTICS UV AND DV.
C
          UV = 0.0
          DV = 0.0
          RN = FLOAT(N)
          DO 400 I = 1, 6
            DO 400 J = 1, 6
              UV = UV + (FLOAT(UCOUNT(I)) - RN * B(I)) *
         *       (FLOAT(UCOUNT(J)) - RN * B(J)) * A(I, J)
              DV = DV + (FLOAT(DCOUNT(I)) - RN * B(I)) *
         *       (FLOAT(DCOUNT(J)) - RN * B(J)) * A(I, J)
      400 CONTINUE
          UV = UV / RN
          DV = DV / RN
          GOTO 700
      500 IFAULT = N
          GOTO 700
      600 IFAULT = 1
      700 RETURN
          END
```

```
      SUBROUTINE PROBS(K, W, P, IFAULT)
C
C          ALGORITHM AS 158 APPL. STATIST. (1981) VOL.30, NO.1
C
C          CALCULATION OF THE PROBABILITIES P(L,K) FOR
C          THE CASE OF SIMPLE ORDER,
C          FOR EQUAL WEIGHTS, K .GE. 3 AND .LE. 10 .
C          FOR UNEQUAL WEIGHTS, K .GE. 3 AND .LE. 6
C
      DIMENSION W(10), P(10), Q(10, 10)
      REAL ZERO, HALF, ONE, THREE, SIX
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/, THREE/3.0/,
     *     SIX/6.0/, C1 /1.0E-6/
C
C        CHECK THAT WEIGHTS ARE POSITIVE
C
      IFAULT = 0
      DO 1 I = 1, K
        IF (W(I) .LE. ZERO) GOTO 101
    1 CONTINUE
C
C        CHECK THAT K .GE. 3 AND .LE. 10
C
      IF (K .LT. 2 .OR. K .GT. 10) GOTO 102
      WW = W(1)
      DO 2 I = 2, K
        IF (ABS(WW - W(I)) .GT. C1) GOTO 7
    2 CONTINUE
C
C        EQUAL WEIGHTS
C
      Q(1, 3) = ONE / THREE
      Q(2, 3) = HALF
      Q(3, 3) = ONE / SIX
      IF (K .EQ. 3) GOTO 5
      DO 4 J = 4, K
        AJ = J
        A1 = ONE /AJ
        A2 = (AJ - ONE) * A1
        Q(1, J) = A1
        J1 = J - 1
        DO 3 L = 2, J1
          L1 = L- 1
    3   Q(L, J) = A1 * Q(L1, J1) + A2 * Q(L, J1)
        Q(J, J) = ONE / FACT(J, IFAULT)
        IF (IFAULT .NE. 0) RETURN
    4 CONTINUE
    5 CONTINUE
      DO 6 J = 1, K
    6 P(J) = Q(J, K)
      RETURN
C
C        UNEQUAL WEIGHTS - CHECK THAT K .LE. 6
C
    7 IF (K .GT. 6) GOTO 102
      K2 = K - 2
      GOTO (8, 9, 10, 11), K2
    8 P(1) = PR1(1, 3, W)
      P(2) = HALF
      P(3) = HALF - P(1)
      RETURN
```

```
      9 P(1) = PR1(1, 4, W)
        P(4) = PR1(4, 4, W)
        P(2) = HALF - P(4)
        P(3) = HALF - P(1)
        RETURN
     10 P(5) = PR1(5, 5, W)
        P(4) = PR1(4, 5, W)
        P(2) = HALF - P(4)
        P(1) = PR1(1, 5, W)
        P(3) = HALF - P(1) - P(5)
        RETURN
     11 SUM = ZERO
        DO 12 I = 2, 6
          PP = PR2(I, 2)
          P(I) = PP
          SUM = SUM + PP
     12 CONTINUE
        P(1) = ONE - SUM
        IF (P(1) .LT. ZERO) P(1) = ZERO
        RETURN
    101 IFAULT = 1
        RETURN
    102 IFAULT = 2
        RETURN
        END
C
        FUNCTION PR1(I, J, W)
C
C          ALGORITHM AS 158.1 APPL. STATIST. (1981) VOL.30, NO.1
C
C          EXPLICIT CALCULATION OF PROBABILITIES FOR K .LE. 5
C          ALSO CALLED BY FUNCTION F2
C
        DIMENSION W(10)
        REAL ZERO, HALF, QTR, EIGHTH, PT0625, PT375, PII
        DATA ZERO/0.0/, HALF/0.5/, QTR/0.25/, EIGHTH/0.125/,
       *     PT0625/0.0625/, PT375/0.375/, PII/0.318309886/
C
        IF (J .NE. 3) GOTO 40
        C = HALF * PII * F1(W(1), W(2), W(3))
        IF (I .EQ. 3) GOTO 30
        PR1 = QTR - C
        RETURN
     30 PR1 = QTR + C
        RETURN
     40 W1 = W(1)
        W2 = W(2)
        W3 = W(3)
        W4 = W(4)
        W12 = W1 + W2
        W23 = W2 + W3
        W34 = W3 + W4
        S12 = F1(W1, W2, W3)
        S23 = F1(W2, W3, W4)
        IF (J .EQ. 5) GOTO 50
        IF (I .EQ. 4) GOTO 41
        C1 = QTR * PII *
       *   (F1(W1, W2, W34) + F1(W1, W23, W4) + F1(W12, W3, W4))
        PR1 = EIGHTH - C1
        RETURN
     41 C2 = QTR *PII * (S12 + S23)
```

```
      PR1 = EIGHTH + C2
      RETURN
   50 W5 = W(5)
      W45 = W4 + W5
      W123 = W12 + W3
      W234 = W23 + W4
      W345 = W34 + W5
      S34 = F1(W3, W4, W5)
      IF (I .EQ. 4) GOTO 52
      C5 = PT0625 + EIGHTH * PII * (S12 + S23 + S34) +
     *   QTR * PII * PII * S12 * S34
      IF (I .EQ. 1) GOTO 51
      PR1 = C5
      IF (PR1 .LT. ZERO) PR1 = ZERO
      RETURN
   51 S113 = F1(W1, W2, W345)
      S131 = F1(W1, W234, W5)
      S311 = F1(W123, W4, W5)
      C3 = PT375 + EIGHTH * PII * (S113 + S131 +S311 + F1(W1, W23, W45)
     *  + F1(W12, W3, W45) + F1(W12, W34, W5) - S12 - S23 - S34)
     *  - QTR * PII * PII * (S12 * S311 + S23 * S131 + S34 * S113)
      PR1 = HALF - C3 - C5
      RETURN
   52 C2 = EIGHTH * PII * (S12 + S34 + F1(W1, W2, W34) + F1(W12, W3, W4)
     *  + F1(W1, W23, W4) + F1( W2, W3, W45) + F1(W23, W4, W5) +
     *  F1(W2, W34, W5))
      PR1 = QTR + C2
      RETURN
      END
C
C
      FUNCTION F1(V1, V2, V3)
C
C      ALGORITHM AS 158.2 APPL. STATIST. (1981) VOL.30, NO.1
C
      RHO = -SQRT(V1 * V3 / ((V1 + V2) * (V2 + V3)))
      F1 = ASIN(RHO)
      RETURN
      END
C
      FUNCTION PR2(I, W)
C
C        ALGORITHM AS 158.3 APPL. STATIST. (1981) VOL.30, NO.1
C
C        CALCULATION OF PROBABILITIES FOR K .EQ. 6 USING
C        RECURRENCE RELATION
C
      DIMENSION W(10)
      REAL ZERO, HALF, EIGHTH, PT0625, P03125, QTR, PII
      DATA ZERO/0.0/, HALF/0.5/, EIGHTH/0.125/, PT0625/0.0625/,
     *     P03125/0.03125/, QTR/0.25/
      DATA PII /0.318309886/
C
      W1 = W(1)
      W2 = W(2)
      W3 = W(3)
      W4 = W(4)
      W5 = W(5)
      W6 = W(6)
      W12 = W1 + W2
      W23 = W2 + W3
```

```
      W34 = W3 + W4
      W45 = W4 + W5
      W56 = W5 + W6
      W123 = W12 + W3
      W234 = W23 + W4
      W345 = W34 + W5
      W456 = W45 + W6
      W1234 = W123 + W4
      W2345 = W234 + W5
      W3456 = W345 + W6
      D4 = ZERO
      D5 = ZERO
      I1 = I - 1
      GOTO (2, 3, 4, 5, 6), I1
    2 PR2 = HALF * (F2(1, 5, W2, W3, W4, W5, W6) +
     *  F2(1, 5, W1, W2, W3, W4, W5) + F2(1, 3, W1, W2, W3, D4, D5) *
     *  F2(1, 3, W4, W5, W6, D4, D5)) + QTR *
     *  (F2(1, 4, W3, W4, W5, W6, D5) + F2(1, 4, W1, W2, W3, W4, D5))
      RETURN
    3 PR2 = F2(3, 3, W1, W2, W3456, D4, D5) *
     *  F2(1, 4, W3, W4, W5, W6, D5) + F2(3, 3, W1, W2345, W6, D4, D5) *
     *  F2(1, 4, W2, W3, W4, W5, D5) + F2(3, 3, W1234, W5, W6, D4, D5) *
     *  F2(1, 4, W1, W2, W3, W4, D5) + HALF *
     *  (F2(3, 3, W1, W23, W456, D4, D5) * F2(1, 3, W4, W5, W6, D4, D5)
     * + F2(3, 3, W1, W234, W56, D4, D5) * F2(1, 3, W2, W3, W4, D4, D5)
     * + F2(3, 3, W12, W3, W456, D4, D5) * F2(1, 3, W4, W5, W6, D4, D5)
     * + F2(3, 3, W12, W345, W6, D4, D5) * F2(1, 3, W3, W4, W5, D4, D5)
     * + F2(3, 3, W123, W4, W56, D4, D5) * F2(1, 3, W1, W2, W3, D4, D5)
     * + F2(3, 3, W123, W45, W6, D4, D5) * F2(1, 3, W1, W2, W3, D4, D5))
     * + EIGHTH * F2(3, 3, W12, W34, W56, D4, D5)
      RETURN
    4 PR2 = F2(4, 4, W1, W2, W3, W456, D5) *
     *  F2(1, 3, W4, W5, W6, D4, D5) + F2(4, 4, W1, W2, W345, W6, D5) *
     *  F2(1, 3, W3, W4, W5, D4, D5) + F2(4, 4, W1, W234, W5, W6, D5) *
     *  F2(1, 3, W2, W3, W4, D4, D5) + F2(4, 4, W123, W4, W5, W6, D5) *
     *  F2(1, 3, W1, W2, W3, D4, D5) + QTR *
     *  (F2(4, 4, W1, W2, W34, W56, D5) + F2(4, 4, W1, W23, W4, W56, D5)
     *  + F2(4, 4, W1, W23, W45, W6, D5) +
     *  F2(4, 4, W12, W3, W4, W56, D5) + F2(4, 4, W12, W3, W45, W6, D5)
     *  + F2(4, 4, W12, W34, W5, W6, D5))
      RETURN
    5 PR2 = HALF * (F2(5, 5, W1, W2, W3, W4, W56) +
     *  F2(5, 5, W1, W2, W3, W45, W6) + F2(5, 5, W1, W2, W34, W5, W6) +
     *  F2(5, 5, W1, W23, W4, W5, W6) + F2(5, 5, W12, W3, W4, W5, W6))
      RETURN
    6 S12 = F1(W1, W2, W3)
      S23 = F1(W2, W3, W4)
      S34 = F1(W3, W4, W5)
      S45 = F1(W4, W5, W6)
      PR2 = P03125 + PT0625 * PII * (S12 + S23 + S34 + S45) +
     *  EIGHTH * PII * PII * (S12 * S34 + S12 * S45 + S23 * S45)
      IF (PR2 .LT. ZERO) PR2 = ZERO
      RETURN
      END
C
      FUNCTION F2(I, J, V1, V2, V3, V4, V5)
C
C        ALGORITHM AS 158.4 APPL. STATIST. (1981) VOL.30, NO.1
C
      DIMENSION VV(10)
      VV(1) = V1
```

```
      VV(2) = V2
      VV(3) = V3
      VV(4) = V4
      VV(5) = V5
      F2 = PR1(I, J, VV)
      RETURN
      END
C
      FUNCTION FACT(M, IFAULT)
C
C        ALGORITHM AS 158.5 APPL. STATIST. (1981) VOL.30, NO.1
C
C        CALCULATION OF M FACTORIAL
C
      REAL ONE
      DATA ONE/1.0/, MAXM /50/
C
      IFAULT = 3
      IF (M .LT. 0 .OR. M .GT. MAXM) RETURN
      IFAULT = 0
      FACT = ONE
      IF (M .LE. 1) RETURN
      A1 = ONE
      DO 1 I = 2, M
        AI = I
        A1 = A1 * AI
    1 CONTINUE
      FACT = A1
      RETURN
      END
```

```
      SUBROUTINE RCONT2(NROW, NCOL, NROWT, NCOLT, JWORK, MATRIX,
     *    KEY, IFAULT)
C
C     ALGORITHM AS 159  APPL. STATIST. (1981) VOL.30, NO.1
C
C     Generate random two-way table with given marginal totals
C
C     N.B. The call to the NAG random number function G05AAF has been
C     replaced by the use of the random number generator from AS 183.
C
C     N.B. See the journal article for the significance of the lines
C     which start with C*
C
      INTEGER NROWT(NROW), NCOLT(NCOL), MATRIX(NROW, NCOL),
     *     JWORK(NCOL)
      REAL FACT(5001), DUMMY, ZERO, ONE, HALF
      LOGICAL KEY
      LOGICAL LSP, LSM
      COMMON /B/ NTOTAL, NROWM, NCOLM, FACT
C
C*      COMMON /TEMPRY/ HOP
C
      DATA MAXTOT /5000/, ZERO/0.0/, ONE/1.0/, HALF/0.5/
C
      IFAULT = 0
      IF (KEY) GO TO 103
C
C     Set KEY for subsequent calls
C
      KEY = .TRUE.
C
C     Check for faults and prepare for future calls
C
      IF (NROW .LE. 1) GO TO 212
      IF (NCOL .LE. 1) GO TO 213
      NROWM = NROW - 1
      NCOLM = NCOL - 1
      DO 100 I = 1, NROW
        IF (NROWT(I) .LE. 0) GO TO 214
  100 CONTINUE
      NTOTAL = 0
      DO 101 J = 1, NCOL
        IF (NCOLT(J) .LE. 0) GO TO 215
        NTOTAL = NTOTAL + NCOLT(J)
  101 CONTINUE
      IF (NTOTAL .GT. MAXTOT) GO TO 216
C
C     Calculate log-factorials
C
      X = ZERO
      FACT(1) = ZERO
      DO 102 I = 1, NTOTAL
        X = X + LOG(FLOAT(I))
        FACT(I+1) = X
  102 CONTINUE
C
C     -----------------------
C     Construct random matrix
C     -----------------------
C
  103 DO 105 J = 1, NCOLM
```

```
  105 JWORK(J) = NCOLT(J)
      JC = NTOTAL
C
C*      HOP = ONE
C
      DO 190 L = 1, NROWM
        NROWTL = NROWT(L)
        IA = NROWTL
        IC = JC
        JC = JC - NROWTL
        DO 180 M = 1, NCOLM
          ID = JWORK(M)
          IE = IC
          IC = IC - ID
          IB = IE - IA
          II = IB - ID
C
C     Test for zero entries in MATRIX
C
          IF (IE .NE. 0) GO TO 130
          DO 121 J = M, NCOL
  121     MATRIX(L,J) = 0
          GO TO 190
C
C     Generate pseudo-random number
C
  130     DUMMY = RAND()
C
C     Compute conditional expected value of MATRIX(L, M)
C
  131     NLM = IA * ID / FLOAT(IE) + HALF
          IAP = IA + 1
          IDP = ID + 1
          IGP = IDP - NLM
          IHP = IAP - NLM
          NLMP = NLM + 1
          IIP = II + NLMP
          X = EXP(FACT(IAP) + FACT(IB+1) + FACT(IC+1) + FACT(IDP) -
     *      FACT(IE+1) - FACT(NLMP) - FACT(IGP) - FACT(IHP) - FACT(IIP))
          IF (X .GE. DUMMY) GO TO 160
          SUMPRB = X
          Y = X
          NLL = NLM
          LSP = .FALSE.
          LSM = .FALSE.
C
C     Increment entry in row L, column M
C
  140     J = (ID - NLM) * (IA - NLM)
          IF (J .EQ. 0) GO TO 156
          NLM = NLM + 1
          X = X * J / FLOAT(NLM * (II + NLM))
          SUMPRB = SUMPRB + X
          IF (SUMPRB .GE. DUMMY) GO TO 160
  150     IF (LSM) GO TO 155
C
C     Decrement entry in row L, column M
C
          J = NLL * (II + NLL)
          IF (J .EQ. 0) GO TO 154
          NLL = NLL - 1
```

```
            Y = Y * J / FLOAT((ID - NLL) * (IA - NLL))
            SUMPRB = SUMPRB + Y
            IF (SUMPRB .GE. DUMMY) GO TO 159
            IF (.NOT. LSP) GO TO 140
            GO TO 150
  154       LSM = .TRUE.
  155       IF (.NOT. LSP) GO TO 140
            DUMMY = SUMPRB * RAND()
            GO TO 131
  156       LSP = .TRUE.
            GO TO 150
  159       NLM = NLL
C
C*            HOP = HOP * Y
C*            GO TO 161
C*160         HOP = HOP * X
C*161         MATRIX(L,M) = NLM
C
  160       MATRIX(L,M) = NLM
            IA = IA - NLM
            JWORK(M) = JWORK(M) - NLM
  180     CONTINUE
          MATRIX(L,NCOL) = IA
  190   CONTINUE
C
C       Compute entries in last row of MATRIX
C
        DO 192 M = 1, NCOLM
  192   MATRIX(NROW,M) = JWORK(M)
        MATRIX(NROW,NCOL) = IB - MATRIX(NROW,NCOLM)
        RETURN
C
C       Set faults
C
  212   IFAULT = 1
        RETURN
  213   IFAULT = 2
        RETURN
  214   IFAULT = 3
        RETURN
  215   IFAULT = 4
        RETURN
  216   IFAULT = 5
        RETURN
        END
```

```
      SUBROUTINE SCREEN(NVAR, MP, MM, NTAB, TABLE, DIM, GSQ, DGFR,
     *   PART, MARG, DFS, IP, IM, ISET, JSET, CONFIG, FIT, SIZE,
     *   COORD, X, Y, IFAULT)
C
C     ALGORITHM AS 160 APPL. STATIST. (1981) VOL.30, NO.1
C
C     Screen all efects for partial and marginal association.
C
      INTEGER NVAR, MP, MM, NTAB, IP(NVAR,MP), IM(NVAR,MM), DGFR(NVAR),
     *   DFS(NVAR,MP), ISET(NVAR), JSET(NVAR), CONFIG(NVAR,MP),
     *   DIM(NVAR), DF, SIZE(NVAR), COORD(NVAR)
      REAL GSQ(NVAR), PART(NVAR,MP), MARG(NVAR,MP), TABLE(NTAB),
     *   FIT(NTAB), X(NTAB), Y(NTAB), ZERO
      DATA ZERO /0.0/
C
C     Check for input errors
C
      IFAULT = 1
      IF (NVAR .LE. 1) RETURN
      ISZ = 1
      DO 100 I = 1, NVAR
        IF (DIM(I) .LE. 1) IFAULT = 2
        ISZ = ISZ * DIM(I)
  100 CONTINUE
      IF (ISZ .NE. NTAB) IFAULT = 2
      MAX = 1
      LIM = NVAR / 2
      DO 110 I = 1, LIM
  110 MAX = MAX * (NVAR - I + 1) / I
      IF (MP .LT. MAX) IFAULT = 3
      MAX = 1
      LIM = (NVAR - 1) / 2
      DO 120 I = 1, LIM
  120 MAX = MAX * (NVAR - I) / I
      MAX = MAX * NVAR
      IF (MM .LT. MAX) IFAULT = 4
      IF (IFAULT .GT. 1) RETURN
C
C     Fit the no effect model
C
      DGFR(NVAR) = NTAB - 1
      AVG = ZERO
      IFAULT = 5
      DO 130 I = 1, NTAB
        IF (TABLE(I) .LT. ZERO) RETURN
        AVG = AVG + TABLE(I)
  130 CONTINUE
      IFAULT = 0
      AVG = AVG / NTAB
      CALL RESET(FIT, NTAB, AVG)
      CALL LIKE(GSQ, FIT, TABLE, NTAB)
C
C     Begin fitting effects
C
      NV1 = NVAR - 1
      DO 200 M = 1, NV1
C
C     Set up the arrays IP and IM
C
        CALL CONF(NVAR, M, MP, MM, ISET, JSET, IP, IM, NP)
C
```

```
C     Fit the saturated model
C
        CALL RESET(FIT, NTAB, AVG)
        CALL EVAL(IP, NP, M, 1, NVAR, MP, CONFIG, DIM, DGFR(M))
        CALL LOGFIT(NVAR, NTAB, NP, DIM, CONFIG, TABLE, FIT, SIZE,
     *     COORD, X, Y)
        CALL LIKE(GSQ(M+1), FIT, TABLE, NTAB)
C
C     Move the first column of IP to the last
C
        DO 150 I = 1, M
          ITP = IP(I,1)
          NP1 = NP - 1
          DO 140 J = 1, NP1
  140     IP(I,J) = IP(I,J+1)
          IP(I,NP) = ITP
  150   CONTINUE
        L3 = -M + 1
        DO 190 J = 1, NP
C
C     Fit the effects in IP ignoring the last column
C
          CALL RESET(FIT, NTAB, AVG)
          CALL EVAL(IP, NP-1, M, 1, NVAR, MP, CONFIG, DIM, DF)
          CALL LOGFIT(NVAR, NTAB, NP-1, DIM, CONFIG, TABLE, FIT, SIZE,
     *         COORD, X, Y)
          CALL LIKE(G21, GIT, TABLE, NTAB)
          DFS(M,J) = DGFR(M) - DF
          PART(M,J) = G21 - GSQ(M+1)
C
C     For M = 1, partials and marginals are equal
C
          IF (M .GT. 1) GO TO 160
          MARG(1,J) = PART(1,J)
          GO TO 170
C
C     Fit the last column alone
C
  160     CALL RESET(FIT, NTAB, AVG)
          CALL EVAL(IP, 1, M, NP, NVAR, MP, CONFIG, DIM, DF)
          CALL LOGFIT(NVAR, NTAB, 1, DIM, CONFIG, TABLE, FIT, SIZE,
     *         COORD, X, Y)
          CALL LIKE(G22, FIT, TABLE, NTAB)
C
C     Locate the appropriate columns of IM and fit them
C
        L3 = L3 + M
          CALL RESET(FIT, NTAB, AVG)
          CALL EVAL(IM, M, M-1, L3, NVAR, MM, CONFIG, DIM, DF)
          CALL LOGFIT(NVAR, NTAB, M, DIM, CONFIG, TABLE, FIT, SIZE,
     *         COORD, X, Y)
          CALL LIKE(G23, FIT, TABLE, NTAB)
          MARG(M,J) = G23 - G22
C
C     Move the next effect to be ignored to the last in IP
C
  170     DO 180 I = 1, M
            ITP = IP(I,NP)
            IP(I,NP) = IP(I,J)
            IP(I,J) = ITP
  180     CONTINUE
```

```
  190   CONTINUE
C
        DGFR(NVAR) = DGFR(NVAR) - DGFR(M)
        GSQ(M) = GSQ(M) - GSQ(M+1)
  200 CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE CONF(N, M, MP, MM, ISET, JSET, IP, IM, NP)
C
C     ALGORITHM AS 160.1 APPL. STATIST. (1981) VOL.30, NO.1
C
C     Set up the arrays IP and IM for a given N and M.   Essentially
C     IP contains all possible combinations of (N choose M).   For each
C     combination found IM contains all combinations of degree M-1.
C
      INTEGER ISET(N), JSET(N), IP(N,MP), IM(N,MM)
      LOGICAL ILAST, JLAST
C
      ILAST = .TRUE.
      NP = 0
      NM = 0
C
C     Get IP
C
  100 CALL COMBO(ISET, N, M, ILAST)
      IF (ILAST) RETURN
      NP = NP + 1
      DO 110 I = 1, M
  110 IP(I,NP) = ISET(I)
      IF (M .EQ. 1) GO TO 100
C
C     Get IM
C
      JLAST = .TRUE.
      L = M - 1
  120 CALL COMBO(JSET, M, L, JLAST)
      IF (JLAST) GO TO 100
      NM = NM + 1
      DO 130 I = 1, L
        JS = JSET(I)
        IM(I,NM) = ISET(JS)
  130 CONTINUE
      GO TO 120
C
      END
C
C
      SUBROUTINE COMBO(ISET, N, M, LAST)
C
C     ALGORITHM AS 160.2  APPL. STATIST. (1981) VOL.30, NO.1
C
C     Subroutine to generate all possible combinations of M of the
C     integers from 1 to N in a stepwise fashion.   Prior to the first
C     call, LAST should be set to .FALSE.   Thereafter, as long as LAST
C     is returned .FALSE., a new valid combination has been generated.
C     When LAST goes .TRUE., there are no more combinations.
C
      LOGICAL LAST
```

```
      INTEGER N, M, ISET(M)
C
      IF (LAST) GO TO 110
C
C     Get next element to increment
C
      K = M
100   L = ISET(K) + 1
      IF (L + M - K .LE. N) GO TO 150
      K = K - 1
C
C     See if we are done
C
      IF (K .LE. 0) GO TO 130
      GO TO 100
C
C     Initialize first combination
C
110   DO 120 I = 1, M
120   ISET(I) = I
130   LAST = .NOT. LAST
      RETURN
C
C     Fill in remainder of combination.
C
150   DO 160 I = K, M
        ISET(I) = L
        L = L + 1
160   CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE EVAL(IAR, NC, NV, IBEG, NVAR, MAX, CONFIG, DIM, DF)
C
C     ALGORITHM AS 160.3 APPL. STATIST. (1981) VOL.30, NO.1
C
C     IAR  = array containing the effects to be fitted
C     NC   = number of columns of IAR to be used
C     NV   = number of variables in each effect
C     IBEG = gebinning column
C     DF   = degrees of freedom
C
C     CONFIG is in a format compatible with algorithm AS 51
C
      INTEGER IAR(NVAR,MAX), CONFIG(NVAR,NC), DIM(NVAR), DF
C
      DF = 0
      DO 110 J = 1, NC
        KK = 1
        DO 100 I = 1, NV
          L = IBEG + J - 1
          K = IAR(I,L)
          KK = KK * (DIM(K) - 1)
          CONFIG(I,J) = K
  100   CONTINUE
        CONFIG(NV+1,J) = 0
        DF = DF + KK
  110 CONTINUE
C
```

```
      RETURN
      END
C
C
      SUBROUTINE RESET(FIT, NTAB, AVG)
C
C     ALGORITHM AS 160.4 APPL. STATIST. (1981) VOL.30, NO.1
C
C     Initialize the fitted values to the average entry
C
      REAL FIT(NTAB)
C
      DO 100 I = 1, NTAB
        FIT(I) = AVG
  100 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE LIKE(GSQ, FIT, TABLE, NTAB)
C
C     ALGORITHM AS 160.5 APPL. STATIST. (1981) VOL.30, NO.1
C
C     Compute the likelihood-ration chi-square
C
      REAL FIT(NTAB), TABLE(NTAB), ZERO, TWO
      DATA ZERO /0.0/, TWO /2.0/
C
      GSQ = ZERO
      DO 100 I = 1, NTAB
        IF (FIT(I) .EQ. ZERO .OR. TABLE(I) .EQ. ZERO) GO TO 100
        GSQ = GSQ + TABLE(I) * LOG(TABLE(I) / FIT(I))
  100 CONTINUE
      GSQ = TWO * GSQ
      RETURN
      END
C
C
      SUBROUTINE LOGFIT(NVAR, NTAB, NCON, DIM, CONFIG, TABLE, FIT, SIZE,
     *     COORD, X, Y)
C
C     ALGORITHM AS 160.6 APPL. STATIST. (1981) VOL.30, NO.1
C
C     Iterative proportional fitting of the marginals of a contingency
C     table.   Relevant code from AS 51 is used.
C
      REAL TABLE(NTAB), FIT(NTAB), MAXDEV, X(NTAB), Y(NTAB), ZERO
      INTEGER CONFIG(NVAR,NCON), DIM(NVAR), SIZE(NVAR), COORD(NVAR)
      LOGICAL OPTION
      DATA MAXDEV /0.25/, MAXIT /25/, ZERO /0.0/
C
      DO 230 KK = 1, MAXIT
C
C     XMAX is the maximum deviation between fitted and true marginal
C
        XMAX = ZERO
        DO 220 II = 1, NCON
          OPTION = .TRUE.
C
C     Initialize arrays
C
```

```
            SIZE(1) = 1
            NV1 = NVAR - 1
            DO 100 K = 1, NV1
              L = CONFIG(K,II)
              IF (L .EQ. 0) GO TO 110
              SIZE(K+1) = SIZE(K) * DIM(L)
  100       CONTINUE
            K = NVAR
  110       N = K - 1
            ISZ = SIZE(K)
            DO 120 J = 1, ISZ
              X(J) = ZERO
              Y(J) = ZERO
  120       CONTINUE
C
C     Initialize co-ordinates
C
  130       DO 140 K = 1, NVAR
  140       COORD(K) = 0
C
C     Find locations in tables
C
            I = 1
  150       J = 1
            DO 160 K = 1, N
              L = CONFIG(K,II)
              J = J + COORD(L) * SIZE(K)
  160       CONTINUE
            IF (.NOT. OPTION) GO TO 170
C
C     Compute marginals
C
            X(J) = X(J) + TABLE(I)
            Y(J) = Y(J) + FIT(I)
            GO TO 180
C
C     Make adjustments
C
  170       IF (Y(J) .LE. ZERO) FIT(I) = ZERO
            IF (Y(J) .GT. ZERO) FIT(I) = FIT(I) * X(J) / Y(J)
C
C     Update co-ordinates
C
  180       I = I + 1
            DO 190 K = 1, NVAR
              COORD(K) = COORD(K) + 1
              IF (COORD(K) .LT. DIM(K)) GO TO 150
              COORD(K) = 0
  190       CONTINUE
            IF (.NOT. OPTION) GO TO 200
            OPTION = .FALSE.
            GO TO 130
C
C     Find the largest deviation
C
  200       DO 210 I = 1, ISZ
              E = ABS(X(I) - Y(I))
              IF (E .GT. XMAX) XMAX = E
  210       CONTINUE
  220     CONTINUE
C
```

```
C     Test convergence
C
        IF (XMAX .LT. MAXDEV) RETURN
  230 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE TABLE(N1, N2, ALPHA, MM, A1, PMAX, FSUM, A2, PMAX2,
     * FSUM2, IFAULT)
C
C        ALGORITHM AS 161 APPL. STATIST. (1981) VOL.30, NO.2
C
      DIMENSION MM(101, 2), FSUM(201), X(101, 101), COM(101),
     * FSUM2(201), COMB(101), COMBO(201)
C
      IFAULT = 0
      IF (N2 .GT. 100) IFAULT = IFAULT + 1
      IF (N1 .GT. N2) IFAULT = IFAULT + 2
      IF (ALPHA .LE. 0.0 .OR. ALPHA .GE. 0.3) IFAULT = IFAULT + 4
      IF (IFAULT .NE. 0) RETURN
      N11 = N1 + 1
      N21 = N2 + 1
      CALL COMBIN(N11, COM)
      CALL COMBIN(N21, COMB)
      CALL COMBIN(N1 + N2 + 1, COMBO)
      DO 10 I = 1, N11
        MM(I, 1) = 999
        MM(I, 2) = -999
   10 CONTINUE
      DO 15 I = 1, N11
        DO 15 J = 1, N21
        X(I, J) = -1.0
   15 CONTINUE
C
C        INCREASE CHECK UNTIL SIG, THE LEAST UPPER BOUND ON
C        THE SIZE OF CRITICAL REGION, IS .GE. ALPHA + 0.0005
C
      DEL = 0.1
      CHECK = 3.0 * ALPHA
   20 IF (CHECK .GT. 1.0) GOTO 30
      CALL UMPUAL(N1, N2, CHECK, MM, X, SIG, FSUM, PMAX,
     * COM, COMB, COMBO)
      IF (SIG .GE. ALPHA) GOTO 40
      CHECK = CHECK + DEL
      GOTO 20
   30 CHECK = 1.0
      CALL UMPUAL(N1, N2, CHECK, MM, X, SIG, FSUM, PMAX,
     * COM, COMB, COMBO)
   40 IF (SIG .LE. ALPHA + 0.0005) GOTO 50
C
C        BACKUP SIG UNTIL LESS THAN ALPHA BY REPEATEDLY
C        ELIMINATING THE POINT WITH LARGEST PROBABILITY
C        FROM THE CRITICAL REGION
C
      J = 1
      IF (MM(J, 1) .EQ. 999) GOTO 50
      JOLD = MM(J, 1)
      DO 45 I = 1, N11
        IF (MM(I, 1) .EQ. 999) GOTO 47
        INEW = MM(I, 1)
        IF (X(I, INEW) .LE. X(J, JOLD)) GOTO 45
        J = I
        JOLD = MM(J, 1)
   45 CONTINUE
C
C        REMOVE ALL X VALUES CLOSE TO THE MAXIMUM VALUE
C
   47 XVAL = X(J, JOLD)
```

```
      DO 49 I = 1, N11
        IF (MM(I, 1) .EQ. 999) GOTO 49
        INEW = MM(I, 1)
        IF (ABS(X(I, INEW) - XVAL) .GT. 0.00005) GOTO 49
        MM(I, 1) = MM(I, 1) + 1
        IF (MM(I, 1) .GT. N21) MM(I, 1) = 999
   49 CONTINUE
      CALL TMAX(MM, N1, N2, SIG, PMAX, FSUM, COM, COMB)
      GOTO 40
   50 A1 = SIG
      DO 60 I = 1, N11
        IF (MM(I, 1) .EQ. 999) GOTO 60
        IREV = N11 - I + 1
        MM(IREV, 2) = N21 + 1 - MM(I, 1)
   60 CONTINUE
      CALL TMAX(MM, N1, N2, A2, PMAX2, FSUM2, COM, COMB)
      DO 70 J = 1, N11
        IF (MM(J, 1) .NE. 999) MM(J, 1) = MM(J, 1) - 1
        IF (MM(J, 2) .NE. -999) MM(J, 2) = MM(J, 2) - 1
   70 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE COMBIN(NP1, C)
C
C        ALGORITHM AS 161.1 APPL. STATIST. (1981) VOL.30, NO.2
C
C        COMPUTE THE COMBINATIONS OF N = NP1 - 1 ITEMS
C        TAKING J AT A TIME AND STORE THEM IN THE ARRAY
C        C(J + 1), J = 0, 1, 2, ..., N
C
      DIMENSION C(NP1)
      N = NP1 - 1
      C(1) = 1.0
      C(2) = FLOAT(N)
      C(N) = FLOAT(N)
      C(N + 1) = 1.0
      IF (N .LE. 3) RETURN
      M = N / 2 + 1
      DO 10 J = 3, M
        NMJP2 = N - J + 2
        C(J) = C(J - 1) * (FLOAT(NMJP2) / FLOAT(J - 1))
        C(NMJP2) = C(J)
   10 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE UMPUAL(N1, N2, CHECK, MM, X, SIG, FSUM, PMAX, COM,
     * COMB, COMBO)
C
C        ALGORITHM AS 161.2 APPL. STATIST. (1981) VOL.30, NO.2
C
      DIMENSION MM(101, 2), COMBO(201), COMB(101), COM(101), FSUM(201)
      DIMENSION X(101, 101)
      N11 = N1 + 1
      N21 = N2 + 1
      BETA0 = CHECK
C
C        ADD POINTS TO CRITICAL REGION WITH X1 + X2 CONSTANT, UNTIL
```

```
C        THE SUM OF THE PROBABILITIES (BY THE HYPERGEOMETRIC) IS AS
C        LARGE AS POSSIBLE WITHOUT EXCEEDING CHECK
C
C        THE CRITICAL REGION IS REPRESENTED BY MM AND HAS BEEN
C        INITIALIZED IN SUBROUTINE TABLE.
C
C        ADJUST THE CURRENT MM VALUES ROW BY ROW (J1 = 1, N11).
C
      DO 100 J1 = 1, N11
C
C        IF NO ENTRY IN ROW IS CURRENTLY IN THE CRITICAL REGION,
C        ASSUME THE LAST ENTRY (N21) IS.
C

      IF (MM(J1, 1) .EQ. 999) MM(J1, 1) = N2
C
C        INITIALIZE BETAU AND START J2 INCREASED BY 1 FOR THE LOOP
C
      J2 = MM(J1, 1) + 1
      BETAU = -1.0
C
C        WHILE WE ARE STILL IN THE ROW AND THE POSITION IS IN THE
C        CRITICAL REGION, CALCULATE BETAU BY THE HYPERGEOMETRIC
C        ON THE DIAGONAL. THE SUM OF THE PROBABILITIES IS STORED
C        IN X, AN N11 * N21 ARRAY. THE COMPUTATION IS BYPASSED
C        IF DONE EARLIER.
C
   10 IF (J2 .LE. 1 .OR. BETAU .GT. BETA0) GOTO 40
      J2 = J2 - 1
      IF (X(J1, J2) .NE. -1.0) GOTO 30
      J1J2M1 = J1 + J2 - 1
      BETAU = COM(J1) * COMB(J2) / COMBO(J1J2M1)
      IF (J1 .NE. 1 .AND. J2 .NE. N21) BETAU = BETAU + X(J1 - 1, J2 + 1)
      X(J1, J2) = BETAU
      GOTO 10
C
C        RETRIEVE BETAU FROM X ARRAY
C
   30 BETAU = X(J1, J2)
      GOTO 10
C
C        IF WE STOPPED BECAUSE J2 = 1, CONTINUE WITH J1 LOOP
C        OTHERWISE, BACKUP J2 FIRST.
C
   40 IF (BETAU .GT. BETA0) J2 = J2 + 1
C
C        CHECK TO SEE IF ANY POINT IN ROW IS IN THE CRITICAL REGION
C
      IF (J2 .GT. N21) J2 = 999
      MM(J1, 1) = J2
  100 CONTINUE
C
C        FOR THE CRITICAL REGION WITH SIZE LESS THAN OR
C        EQUAL TO CHECK UNDER THE NONRANDOMIZED UMPU TEST,
C        FIND SIG THE LEAST UPPER BOUND ON ITS SIZE UNDER
C        THE UNCONDITIONAL NON-RANDOMIZED TEST
C
      CALL TMAX(MM, N1, N2, FMAX, PMAX, FSUM, COM, COMB)
      SIG = FMAX
      RETURN
      END
```

```
C
C
      SUBROUTINE TMAX(MM, N1, N2, FMAX, PMAX, FSUM, COM, COMB)
C
C        ALGORITHM AS 161.3 APPL. STATIST. (1981) VOL.30, NO.2
C
      DIMENSION FSUM(201), MM(101, 2), P(201), DFSUM(201), COM(101),
     *  COMB(101)
      DATA EPS /0.00001/
C
C        EVALUATE THE PROBABILITY, FSUM(P), OF THE CRITICAL REGION
C        UNDER THE HYPOTHESIS P1 = P2 = P FOR P = 0.0, ..., 1.0
C
      F = 0.0
      KSTEP = N1 + N2 + 1
      DEL = 1.0 / FLOAT(N1 + N2)
      DO 10 K = 1, KSTEP
        P(K) = F
        CALL SRCH(MM, F, N1, N2, FSUM(K), DFSUM(K), COM, COMB)
        F = F + DEL
   10 CONTINUE
      FMAX = 0.0
      PMAX = 0.0
      IF (MM(1, 1) .EQ. 999 .AND. MM(N1 + 1, 2) .EQ. -999) RETURN
C
C        APPROXIMATE THE LEAST UPPER BOUND, FMAX, FOR THE PROBABILITY
C        OF THE CRITICAL REGION ON THE INTERVAL 0.0 TO 1.0
C
      J = 1
      DO 20 I = 2, KSTEP
        IF (FSUM(I) .GT. FSUM(J)) J = I
   20 CONTINUE
      FMAX = FSUM(J)
      PMAX = P(J)
C
C        IF FMAX OCCURS AT EITHER ENDPOINT OF (0, 1) BRANCH OUT
C
      PJMID = P(J)
      SUMID = FSUM(J)
      DSUMID = DFSUM(J)
      IF (J .EQ. 1) GOTO 60
      IF (J .EQ. KSTEP) GOTO 70
C
C        BRANCH OUT IF THE FUNCTION FSUM(P) IS FLAT ENOUGH IN THE
C        NEIGHBORHOOD OF FMAX
C
   25 IF (DFSUM(J - 1) * DFSUM(J + 1) .GT. 0.0) GOTO 40
      SUMLO = FSUM(J - 1)
      PJLO = P(J - 1)
      SUMHI = FSUM(J + 1)
      PJHI = P(J + 1)
   30 IF (ABS(SUMID - SUMLO) .LE. EPS .AND.
     * ABS(SUMID - SUMHI) .LE. EPS) GOTO 40
      IF (DSUMID) 34, 40, 32
   32 PJLO = PJMID
      SUMLO = SUMID
      GOTO 38
   34 PJHI = PJMID
      SUMHI = SUMID
   38 PJMID = (PJLO + PJHI) * 0.5
      CALL SRCH(MM, PJMID, N1, N2, SUMID, DSUMID, COM, COMB)
```

```
      GOTO 30
   40 FMAX = SUMID
      PMAX = PJMID
      RETURN
   60 IF (FMAX .GE. 0.9999) RETURN
      G = 0.0001
      CALL SRCH(MM, G, N1, N2, RUM, DRUM, COM, COMB)
C
C        IF FMAX OCCURS AT 0, RETURN.
C
      IF (RUM .LT. FMAX) RETURN
      J = 2
      GOTO 25
   70 IF (FMAX .GE. 0.9999) RETURN
      G = 0.9999
      CALL SRCH(MM, G, N1, N2, RUM, DRUM, COM, COMB)
C
C        IF FMAX OCCURS AT 1, RETURN.
C
      IF (RUM .LT. FMAX) RETURN
      J = KSTEP - 1
      GOTO 25
      END
C
C
      SUBROUTINE SRCH(MM, PP, N1, N2, SUMS, DSUMS, COM, COMB)
C
C        ALGORITHM AS 161.4 APPL. STATIST. (1981) VOL.30, NO.2
C
C        FIND THE SIZE OF THE CRITICAL REGION UNDER THE UNCONDITIONAL
C        NONRANDOMIZED TEST WITH PROBABILITY P1 = P2 = PP
C
      DIMENSION MM(101, 2), COM(101), COMB(101)
C
      N11 = N1 + 1
      N21 = N2 + 1
      SUM = 0.0
      SUMM = 0.0
      DSUM = 0.0
      DSUMM = 0.0
      IF (PP .GE. 0.99999) GOTO 30
      IF (PP .LE. 0.00001) GOTO 40
C
C        SEARCH THE ONE-SIDED CRITICAL REGION
C
      PP1 = 1.0 - PP
      DO 20 I = 1, N11
      IF (MM(I, 1) .EQ. 999) GOTO 50
      NE = N21
      DP1EXP = PP1 ** (I + N2)
      PPEXP = PP ** (I + N2)
      DPPEXP = PP ** (N1 - I)
      PP1EXP = PP1 ** (N1 - I)
      DO 10 J = 1, N21
      IF (MM(I, 1) .GT. NE) GOTO 20
      PPEXP = PPEXP / PP
      PP1EXP = PP1EXP * PP1
C
C         EVALUATE THE PROBABILITY OF A POINT IN THE CRITICAL REGION
C         BY THE PRODUCT OF INDEPENDENT BINOMIAL TERMS AND ACCUMULATE
C         IN SUM. ALSO EVALUATE THE DERIVATIVE OF SUM AT P1 = P2 = PP.
```

```
C
      IF (MM(1, 1) .EQ. 999) GOTO 4
      SUM = SUM + COM(I) * COMB(NE) * PPEXP * PP1EXP
      DSUM = DSUM + COM(I) * COMB(NE) * (FLOAT(I + N2 - J) * PPEXP /
     * PP * PP1EXP - FLOAT(N1 + J - I) * PPEXP * PP1EXP / PP1)
    4 IF (MM(N11, 2) .EQ. -999) GOTO 5
      DPPEXP = DPPEXP * PP
      DP1EXP = DP1EXP / PP1
      SUMM = SUMM + COM(I) * COMB(NE) * DPPEXP * DP1EXP
      DSUMM = DSUMM + COM(I) * COMB(NE) * (FLOAT(N1 + J - I) * DPPEXP /
     * PP * DP1EXP - FLOAT(N2 + I - J) * DPPEXP * DP1EXP / PP1)
    5 NE = NE - 1
   10 CONTINUE
   20 CONTINUE
      GOTO 50
   30 IF (MM(N11, 1) .NE. 999) SUM = 1.0
      SUMS = SUM + SUMM
      RETURN
C
   40 IF (MM(1, 1) .EQ. 1) SUM = 1.0
      SUMS = SUM
      RETURN
C
   50 SUMS = SUM + SUMM
      DSUMS = DSUM + DSUMM
      RETURN
      END
```

```
      subroutine logccs(ns, nca, nct, nimax, nmax2, nvmax, nv, nv1, z,
     +          ivar, covi, w, ww, dl, iw, is, b, cov, chi2, st, ifault)
c
c     ALGORITHM AS 162  APPL. STATIST. (1981) VOL. 30, NO. 2
c
c     Logistic analysis of case-control studies
c
c     Auxiliary routine required: SYMINV from AS 7.
c     N.B. The description in the journal states that AS 6 is also
c          needed, but where ?
c
      integer   nca(ns), nct(ns), ivar(nv), iw(nmax2), is(ns)
      real      z(nvmax, nimax), covi(nv1), w(nv), ww(nv), dl(nv),
     +          b(nv), cov(nv1)
      real      one, zero, two
      logical   ifg, id
      data maxit /20/, eps /1.e-6/, one/1.0/, zero/0.0/, two/2.0/
c
c     Initial settings
c
      rlikp = one
      ifault = 0
      if (nvmax .lt. nv) go to 27
      do 1 i = 1, ns
        if (nmax2 .lt. nca(i) + nct(i) + 2) go to 27
    1 continue
      is(1) = 0
      IF(NS .EQ. 1) GOTO 28
      do 2 j = 2, ns
        j1 = j - 1
        is(j) = is(j1) + nca(j1) + nct(j1)
    2 continue
   28 if (nimax .lt. is(ns) + nca(ns) + nct(ns)) go to 27
      its = 0
c
c     Start of iteration
c
    3 its = its + 1
      if (its .gt. maxit) go to 24
      rlik = zero
      k = 0
      do 4 j = 1, nv
        dl(j) = zero
        do 4 jj = 1, j
          k = k + 1
          covi(k) = zero
    4 continue
c
c     Loop through strata
c
      do 17 i = 1, ns
        if (nca(i) * nct(i) .eq. 0) go to 17
        ifg = .false.
        sx = zero
        k = 0
        do 5 j = 1, nv
          w(j) = zero
          ww(j) = zero
          do 5 jj = 1, j
            k = k + 1
            cov(k) = zero
```

```
     5   continue
         m = nca(i)
         n = m + nct(i)
         xx = one
         x = zero
         iw(1) = n + 1
         iw(n + 2) = -2
         kk = nct(i) + 1
         do 6 j = 2, kk
     6   iw(j) = 0
         do 7 j = 1, m
           jj = kk + j
           iw(jj) = j
     7   continue
c
c      Calculator numerator of terms of likelihood
c
         do 9 k = 1, nv
           l = ivar(k)
           bk = b(k)
           wk = zero
           do 8 j = 1, m
             ji = is(i) + j
             wk = wk + z(l, ji)
             x = x + bk * z(l, ji)
     8     continue
           w(k) = wk
     9   continue
c
c      Go through all possible combinations to calculate denominator
c      terms of likelihood
c
    10   xx = exp(x)
         sx = sx + xx
         if (ifg) go to 12
         rlik = rlik + log(sx)
         do 11 k = 1, nv
    11   dl(k) = dl(k) + w(k)
         ifg = .true.
    12   l = 0
         do 13 k = 1, nv
           ww(k) = ww(k) + xx * w(k)
           do 13 kk = 1, k
             l = l + 1
             cov(l) = cov(l) + xx * w(k) * w(kk)
    13   continue
         call twidl(ips, im, iz, id, iw, nmax2)
         if (id) go to 15
c
c      Use the special features of TWIDL that only one element is altered
c      at a time, to calculate contribution of succeeding combinations
c      with minimal arithmetic.
c
         ips1i = is(i) + n - ips + 1
         im1i = is(i) + n - im + 1
         do 14 k = 1, nv
           l = ivar(k)
           zc = z(l, ips1i) - z(l, im1i)
           w(k) = w(k) + zc
           x = x + b(k) * zc
    14   continue
```

```
            go to 10
c
   15       rlik = rlik - log(sx)
            l = 0
            do 16 j = 1, nv
              dl(j) = dl(j) - ww(j) / sx
              do 16 k = 1, j
                l = l + 1
                covi(l) = covi(l) + (sx * cov(l) - ww(j) * ww(k)) / sx**2
   16       continue
   17    continue
c
         if (its .eq. 1) rliks = rlik
         call syminv(covi, nv, cov, w, nullty, ifault, nv1)
         if (ifault .ne. 0) go to 25
c
c        Calculate new parameter estimates.
c
         do 20 i = 1, nv
           w(i) = zero
           i2 = i * (i - 1) / 2
           do 18 j = 1, i
             k = i2 + j
             w(i) = w(i) + dl(j) * cov(k)
   18      continue
           i1 = i + 1
           if (i1 .gt. nv) go to 20
           do 19 k = i1, nv
             j = k * (k - 1) / 2 + i
             w(i) = w(i) + dl(k) * cov(j)
   19      continue
   20    continue
         do 21 i = 1, nv
   21    b(i) = b(i) + w(i)
         if (its .ne. 1) go to 23
c
c        Calculate score test
c
         st = zero
         do 22 i = 1, nv
   22    st = st + w(i) * dl(i)
c
c        Test for convergence
c
   23    rlik = rlik - rliks
         if (abs(rlikp - rlik) .le. eps) go to 26
         rlikp = rlik
         go to 3
   24    ifault = 1
         return
   25    ifault = 2
         return
   26    chi2 = two * rlik
         return
   27    ifault = 3
         return
         end
c
c
c
         subroutine twidl(x, y, z, done, p, n2)
```

```
      integer x, y, z, n2, p(n2)
      logical done
c
c       Algorithm AS 162.1  Appl. Statist. (1981) Vol.30, No. 2
c
c       This subroutine is a fortran version of
c       CACM Algorithm 382 for generating all combinations
c       of M out of N objects.  All subscripts in the array P have
c       been increased by unity to avoid reference to P(0).
c
c       Ref:- P.J. Chase (1970), Comm. ACM, Vol.13, No.6, p368.
c
c       Parameters:-
c
c       done         logical     input: Initially set to .false.
c                                output: If all combinations have been
c                                        found then done is set to .true.
c                                        Otherwise done=.false.
c
c       p            integer array  input: Initially, p(1)=N+1, p(N+2)=-2
c                    of length             p(2,...,N-M+1)=0,
c                    N+2                   p(N-M+2,...,N+1)=1,...,M
c                                          If M=0, set p(2)=1.
c
c      n2            integer       input: n2=n+2
c
c      x, y, z       integers      output: See below.
c
c      Initially, let A(1,...,N)=1,...,N
c                 let C(1,...,M)=A(N-M+1),...,A(N) = N-M+1,...,N
c                     be the initial combination.
c      Then the next combination is given by setting C(Z)=A(X)
c
c      Alternatively, initially let B(1,...,N-M)=0 and B(N-M+1,...,N)=1
c                                be the initial sequence of zeros and ones.
c      Then the next sequence of zeros and ones is obtained by setting
c      B(X)=1 and B(Y)=0.
c

      j = 0
    1 j = j + 1
      if (p(j + 1) .le. 0) goto 1
      if (p(j) .ne. 0) goto 4
      if (j .lt. 3) goto 3
      do 2 i = 3, j
    2 p(i) = -1
    3 p(j + 1) = 0
      p(2) = 1
      x = 1
      z = 1
      y = j
      goto 10
    4 if (j .gt. 1) p(j) = 0
    5 j = j + 1
      j1 = j + 1
      if (p(j1) .gt. 0) goto 5
      i = j - 1
      k = i
    6 i = i + 1
      i1 = i + 1
      if (p(i1) .ne.0) goto 7
```

```
        p(i1) = -1
        goto 6
    7 if (p(i1) .ne. -1) goto 8
        z = p(k + 1)
        p(i1) = z
        x = i
        y = k
        p(k + 1) = -1
        goto 10
    8 if (i .ne. p(1)) goto 9
        done = .true.
        goto 10
    9 z = p(i1)
        p(j1) = z
        p(i1) = 0
        x = j
        y = i
   10 return
        end
c
c
c*****************************************************************
c
c  The routine below initializes TWIDL.   It is NOT part of AS 162.
c  It enables TWIDL to be used for other applications.
c
      subroutine initp(n, m, p, n2, done)
      integer p(n2)
      logical done
c
      done = .false.
      n1 = n + 1
      nm1 = n - m + 1
      p(1) = n1
      p(n+2) = -2
      do 1 i = 2, nm1
    1 p(i) = 0
      do 2 i = 1,m
      i1 = nm1 + i
    2 p(i1) = i
      return
      end
```

```
      subroutine gtrans(r, nr, m, n, ifault)
c
c       Algorithm AS163 Applied Statistics (1981) vol.30, no. 2
c
c       Transposes rows m and m+1 in the standardized lower-triangular
c       working matrix r and restores the triangular structure by plane
c       rotations.   r is of length nr = n(n+1)/2 and is stored row by
c       row, shortest row first.
c
      double precision r(nr), a, b, c, d, zero, one
c
      data zero, one/0.d0, 1.d0/
c
      ifault = 1
      if (n .lt. 2 .or. m .lt. 1 .or. m .ge. n) return
      ifault = 0
      ma = m * (m + 1) / 2
      mb = ma + m
c
c       Exchange columns
c
      i = mb
   20 i = i - 1
      if (i .eq. ma) go to 30
      j = i - m
      a = r(i)
      r(i) = r(j)
      r(j) = a
      go to 20
   30 mc = mb + 1
c
c       a will now be set equal to element (m,m)
c       b = element (m,m+1), and c = element (m+1,m+1).
c       If a and c are zero, there is nothing to do.
c
      a = r(ma)
      c = r(mc)
      if (a .le. zero .and. c .le. zero) return
c
c       Other special cases where one or more of a, b or c is zero.
c
      b = r(mb)
      j = m + 2
      mi = mb + j
      if (b .ne. zero) go to 50
      r(mc) = a
      r(ma) = c
      if (j .gt. n) return
      do 40 i = j, n
        b = r(mi-1)
        r(mi-1) = r(mi)
        r(mi) = b
        mi = mi + i
   40 continue
      return

   50 if (c .gt. zero) go to 70
      r(ma) = a * b * b
      r(mb) = one / b
      if (j .gt. n) return
      do 60 i = j, n
```

```
            r(mi-1) = r(mi-1) / b
            mi = mi + i
   60     continue
          return
c
c         General case - a, b, c all non-zero
c
   70     d = a * b * b + c
          r(ma) = d
          c = c / d
          r(mc) = a * c
          a = a * b / d
          r(mb) = a
          if (j .gt. n) return
          do 80 i = j, n
            d = r(mi-1)
            r(mi-1) = a * d + c * r(mi)
            r(mi) = d - b * r(mi)
            mi = mi + i
   80     continue
          return
          end
c
c
          subroutine update(r, x, w, n, nr)
c
c         Algorithm AS163.1 Applied Statistics (1981) vol.30, no.2
c
c         Uses plane rotations to update the standardized square root
c         (Cholesky factor) of a matrix of sums of squares and products
c         when another data vector x is to be included.   r is of length
c         nr = n(n+1)/2 and is stored row by row, shortest row first.
c
          double precision x(n), r(nr), w, y, d, c, s, eps
c
          data eps/0.d0/
c
          l = 0
          do 20 i = 1, n
            if (w .le. eps) return
            l = l + i
            y = x(i)
            if (abs(y) .le. eps) go to 20
            d = r(l) + w * y * y
            if (abs(d) .le. eps) go to 20
            c = r(l) / d
            r(l) = d
            if (i .eq. n) return
            s = w * y / d
            w = c * w
            k = l
            m = i + 1
            do 10 j = m, n
              k = k + j - 1
              d = x(j)
              r(k) = c * r(k) + s * d
   10       continue
   20     continue
          return
          end
```

```
      SUBROUTINE GIVENC(R, IR, NVARS, X, V, IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C       ALGORITHM AS 164  APPL. STATIST. (1981) VOL.30, NO.2
C
C         APPLIES GIVENS TRANSFORMS TO INCLUDE A NEW ROW OF DATA
C         OR CONSTRAINT
C
C         The corrections detailed on p.357 of 'Applied Statistics', Vol.30,
C         have been incorporated.
C
      REAL R(IR), X(NVARS)
      DATA ZERO/0.0/
C
C         SMALL CONSTANTS THAT THE USER CAN MODIFY
C
      DATA EPS0/0.0/, EPS1/0.0/
C
C         CHECK FOR VALID PARAMETERS
C
      IFAULT = 0
      IRUSED = NVARS*(NVARS+1)/2
      IF (IR.LT.IRUSED) GO TO 1003
      IF (V.LT.ZERO) GO TO 1002
      VLOCAL = V
C
C         FOR EACH ROW OF UPPER TRIANGULAR R
C
      II = 0
      DO 60 I = 1, NVARS
        II = II + I
        XI = X(I)
        XI2 = XI*XI
        IF (XI2.LE.ABS(VLOCAL)*EPS0) GO TO 60
        CTEMP = R(II)
        IJ = II
        IPLUS = I + 1
C
C         IF ZERO WEIGHT ON ROW OF R, SIMPLE PIVOT
C
        IF (CTEMP.GE.ZERO) GO TO 20
        R(II) = VLOCAL/XI2
        IF (I.EQ.NVARS) GO TO 70
        DO 10 J = IPLUS, NVARS
          IJ = IJ + J - 1
          R(IJ) = X(J)/XI
   10   CONTINUE
        RETURN
C
C         IF INFINITE WEIGHT ON ROW OF R, SIMPLE PIVOT
C
   20   IF (CTEMP.GT.EPS1) GO TO 40
        DO 30 J = IPLUS, NVARS
          IJ = IJ + J - 1
          X(J) = X(J) - XI*R(IJ)
   30   CONTINUE
        GO TO 60
C
C         OTHERWISE ORDINARY GIVENS ROTATION
```

```
C
   40   VNEW = VLOCAL + CTEMP*XI2
        C = VLOCAL/VNEW
        S = CTEMP*XI/VNEW
        VLOCAL = VNEW
        R(II) = CTEMP*C
        IF (I.EQ.NVARS) GO TO 70
        DO 50 J = IPLUS, NVARS
          IJ = IJ + J - 1
          RTEMP = C*R(IJ) + S*X(J)
          X(J) = X(J) - XI*R(IJ)
          R(IJ) = RTEMP
   50   CONTINUE
C
   60 CONTINUE
C
C         CHECK FOR INCONSISTENT OR DUPLICATED CONSTRAINTS
C
   70 IF (ABS(R(IRUSED)).LE.EPS1) GO TO 1001
      IF (VLOCAL.LE.EPS1) IFAULT = -1
      RETURN
C
C         ERROR FLAG SET
C
 1001 IFAULT = IFAULT + 1
 1002 IFAULT = IFAULT + 1
 1003 IFAULT = IFAULT + 1
      RETURN
      END
C
      SUBROUTINE BSUB(R, IR, IDEP, COEFF, IC, IFAULT)
C
C        ALGORITHM AS 164.1  APPL. STATIST. (1981) VOL.30, NO.2
C
C         PERFORM BACK SUBSTITUTION TO GET PARAMETER ESTIMATES
C
      REAL R(IR), COEFF(IC)
      DATA ZERO/0.0/
C
C         CHECK FOR VALID PARAMETERS
C
      IFAULT = 0
      II = IDEP*(IDEP+1)/2
      NXVARS = IDEP - 1
      IF (IR.LT.II.OR.IC.LT.NXVARS) GO TO 1001
      IF (NXVARS.LT.1) RETURN
C
C         BACK SUBSTITUTION
C
      K = II
      NX = IDEP
      DO 30 I = 1, NXVARS
        II = II - NX
        K = K - 1
        TEMP = R(K)
        IF (R(II).LT.ZERO) IFAULT = IFAULT - 1
        IF (I.EQ.1) GO TO 20
        IJ = II
        DO 10 J = NX, NXVARS
          IJ = IJ + J - 1
          TEMP = TEMP - R(IJ)*COEFF(J)
```

```
      10   CONTINUE
      20   NX = NX - 1
           COEFF(NX) = TEMP
      30 CONTINUE
         RETURN
C
 1001 IFAULT = 1
      RETURN
      END
C
      SUBROUTINE SSCOMP(R, IR, IDEP, NOBS, ICOMP, SSQ, IDF, IFAULT)
C
C        ALGORITHM AS 164.2  APPL. STATIST. (1981) VOL.30, NO.2
C
C         FINDS (ICOMP)TH COMPONENT OF TOTAL SUM OF SQRS
C         ZERO-TH COMPONENT IS RESIDUAL SSQ
C
      DIMENSION R(IR)
      DATA ZERO/0.0/, ONE/1.0/
C
C         SMALL CONSTANT THAT THE USER CAN MODIFY
C
      DATA EPS1/0.0/
C
C         CHECK FOR VALID PARAMETERS
C
      IFAULT = 0
      IRUSED = IDEP*(IDEP + 1)/2
      IF (IR.LT.IRUSED) IFAULT = IFAULT + 1
      IF (ICOMP.LT.0.OR.ICOMP.GE.IDEP) IFAULT = IFAULT + 2
      IF (IFAULT.GT.0) RETURN
C
C         TEST IF RESIDUAL SSQ REQUIRED
C
      IF (ICOMP.GE.1) GO TO 20
      NXVARS = IDEP - 1
      IDF = NOBS - NXVARS
      II = 0
      DO 10 I = 1, NXVARS
        II = II + I
        IF (R(II).LE.EPS1) IDF = IDF + 1
   10 CONTINUE
      SSQ = ZERO
      IF (R(IRUSED).GT.EPS1) SSQ = ONE/R(IRUSED)
      RETURN
C
C         ORDINARY COMPONENT
C
   20 IDF = 0
      SSQ = ZERO
      II = ICOMP*(ICOMP+1)/2
      IF (R(II).LE.EPS1) RETURN
      IDF = 1
      IJ = IRUSED - IDEP + ICOMP
      SSQ = R(IJ)*R(IJ)/R(II)
      RETURN
      END
C
      SUBROUTINE VAR(R, IR, S, IS, IDEP, NOBS, IFAULT)
C
C        ALGORITHM AS 164.3  APPL. STATIST. (1981) VOL.30, NO.2
```

```
C
C         FINDS ESTIMATE OF VAR/COVAR MATRIX OF ESTIMATES
C
      REAL R(IR), S(IS)
      DATA ZERO/0.0/, ONE/1.0/
C
C         SMALL CONSTANT THAT THE USER CAN MODIFY
C
      DATA EPS1/0.0/
C
C         CHECK FOR VALID PARAMETERS
C
      IFAULT = 0
      IRUSED = IDEP*(IDEP+1)/2
      IF (IR.LT.IRUSED.OR.IS.LT.(IRUSED-IDEP)) GO TO 1002
      NXVARS = IDEP - 1
C
C         INVERT UNIT UPPER TRIANGULAR MATRIX
C
      NCONS = 0
      IJ = 0
      DO 50 I = 1, NXVARS
        JJ = 0
        J = 0
   10   J = J + 1
        IJ = IJ + 1
        JJ = JJ + J
        IF (J.LT.I) GO TO 20
        IF (R(IJ).LE.EPS1) NCONS = NCONS + 1
        GO TO 50
   20   STEMP = -R(IJ)
        IK = IJ
        KJ = JJ
        KMAX = I - 1
        KMIN = J + 1
        IF (KMAX.LT.KMIN) GO TO 40
        DO 30 K = KMIN, KMAX
          IK = IK + 1
          KJ = KJ + K - 1
          STEMP = STEMP - R(IK)*S(KJ)
   30   CONTINUE
   40   S(IJ) = STEMP
        GO TO 10
   50 CONTINUE
C
C         ESTIMATE VARIANCE AND APPLY IDENTIFIABILITY CONSTRAINTS
C
      IDF = NOBS - NXVARS + NCONS
      IF (IDF.LE.0) GO TO 1001
      SIGMA = ZERO
      IF (R(IRUSED).GT.ZERO) SIGMA = ONE/(R(IRUSED)*FLOAT(IDF))
      II = 0
      DO 60 I = 1, NXVARS
        II = II + I
        S(II) = SIGMA*R(II)
        IF (R(II).LT.ZERO) S(II) = ZERO
   60 CONTINUE
C
C         MULTIPLY MATRICES TOGETHER TO FORM EST OF VAR
C
      II = 0
```

```
      IJ = 0
      DO 90 I = 1, NXVARS
        II = II + I
        DO 90 J = 1, I
          KK = II
          IJ = IJ + 1
          KI = IJ
          KJ = II
          STEMP = S(KK)
          IF (I.NE.J) STEMP = STEMP*S(IJ)
          K = I
   70     K = K + 1
          IF (K.GT.NXVARS) GO TO 80
          KK = KK + K
          KI = KI + K - 1
          KJ = KJ + K - 1
          STEMP = STEMP + S(KI)*S(KJ)*S(KK)
          GO TO 70
C
   80     S(IJ) = STEMP
   90 CONTINUE
      RETURN
C
 1001 IFAULT = IFAULT + 1
 1002 IFAULT = IFAULT + 1
      RETURN
      END
C
      SUBROUTINE ALIAS(R, IR, NVARS, EPS, WORKSP, IFAULT)
C
C        ALGORITHM AS 164.4  APPL. STATIST. (1981) VOL.30, NO.2
C
C        ASSUMES ANY DIAGONAL ELEMENTS OF D LESS THAN EPS ARE
C        ROUNDING ERRORS AND REDUCES THEM TO ZERO
C
      REAL R(IR), WORKSP(NVARS)
      DATA ZERO/0.0/, ONE/1.0/, ONENEG/-1.0/
C
C         CHECK FOR VALID PARAMETERS
C
      IFAULT = 0
      IRUSED = NVARS*(NVARS+1)/2
      IF (IR.LT.IRUSED) GO TO 1001
      NXVARS = NVARS - 1
C
C         FOR EACH ROW OF TRIANGULAR R
C
      II = 0
      DO 20 I = 1, NXVARS
        II = II + I
        WORKSP(I) = ZERO
C
C         CHECK FOR WEIGHT OF ROW NEAR ZERO
C
        IF (ABS(R(II))*EPS.LE.ONE) GO TO 20
        IFAULT = IFAULT - 1
        V = R(II)
        R(II) = ONENEG
        IJ = II
        IPLUS = I + 1
C
```

```
C         ROTATE MODIFIED ROW WITH GIVENS
C
        DO 10 J = IPLUS, NVARS
          IJ = IJ + J - 1
          WORKSP(J) = R(IJ)
          R(IJ) = ZERO
   10   CONTINUE
        CALL GIVENC(R, IR, NVARS, WORKSP, V, IFAIL)
   20 CONTINUE
      RETURN
C
C         SET FAULT INDICATOR
C
 1001 IFAULT = 1
      RETURN
      END
```

```
      SUBROUTINE KKEY(IDATA,ICLSS,ILCSE,ILSYM,ISMSB,NCASE,NCLAS,NSCOR,
     1      NSYM,ISIG,NUM,NKEY,MAXLEV,NPRINT,NPUNCH,IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C        ALGORITHM AS 165  APPL. STATIST. (1981) VOL.30, NO.3
C
C        Constructs a discriminant function in Fortran for categorical data.
C
      DIMENSION IDATA(NSYM,NCASE),ICLSS(NCASE),ILCSE(NCASE),
     1          ILSYM(3,NKEY),ISMSB(NSYM)
      DIMENSION ITABL(10,10),MTAB(10,10),IFST(10),IT(10),ITERM(10),
     1          ITMIN(10),MEST(10),MINC(10),MSC(10),NOC(10)
      DATA ITWO/1/
C
C        CHECK INPUT VALUES
C
      IF(NCASE.LT.3) GO TO 52
      IF(NSYM.LT.3) GO TO 53
      IF(NSCOR.LT.2 .OR. NSCOR.GT.10) GO TO 54
      IF(NCLAS.LT.2 .OR. NCLAS.GT.10) GO TO 55
      DO 1 J = 1,NCASE
      IF(ICLSS(J).LT.1 .OR. ICLSS(J).GT.NCLAS) GO TO 56
      DO 1 I = 1,NSYM
      IF(IDATA(I,J).LT.0 .OR. IDATA(I,J).GE.NSCOR) GO TO 57
 1    CONTINUE
      IF(ISIG.LT.0 .OR. ISIG.GT.2) GO TO 58
      IF(NPRINT.LT.1 .OR. NPUNCH.LT.1) GO TO 59
C
C        CONSTRUCT KEY AND WRITE FORTRAN FUNCTION
C
C        INDIVIDUAL LABELS STORED IN ILCSE - INITIALISE TO ONES
C
      IFAULT = 0
      IFLAG = 2
      LEVEL = 1
      LEV10 = 10
      ILSYM(3,1) = 0
      ILSYM(3,2) = 1
      DO 2 I = 1,NCASE
 2    ILCSE(I) = 1
      DO 3 I = 1,NCLAS
 3    ITERM(I) = 1
      WRITE(NPUNCH,80)NSYM
C
C        VARIABLE LABELS STORED IN ILSYM - NSYMU GIVES NUMBER STORED
C
      NSYMU = 0
C
C        NEW LEVEL
C        LABEL IS CURRENT INDIVIDUAL LABEL - LABNO GIVES FIRST
C         INDIVIDUAL FOR INSPECTION
C
 4    LABEL = -1
      LABNO = 1
C
C        TAKE NEXT INDIVIDUAL LABEL - SAME LEVEL
C
 5    DO 6 I = LABNO,NCASE
C
```

```
C        IF INDIVIDUAL LABEL NEGATIVE MOVE ON
C
      IF(ILCSE(I).GT.0) GO TO 7
 6    CONTINUE
C
C        NO POSITIVE INDIVIDUAL LABELS - END OF KEY
C
      GO TO 50
C
C        FIND NEXT INDIVIDUAL LABEL IGNORING INDIVIDUALS ALREADY
C        PROCESSED AT THIS LEVEL
C
 7    DO 8 J = I,NCASE
      IF(ILCSE(J).LT.LEV10.AND.ILCSE(J).GE.0) GO TO 9
 8    CONTINUE
C
C        ALL INDIVIDUALS DONE - GO ON TO NEXT LEVEL IF ANY VARIABLES
C        LEFT - OTHERWISE END KEY
C
      LEVEL = LEVEL+1
      LEV10 = LEV10*10
      IF(LEVEL.GT.NSYM ) GO TO 49
      GO TO 4
C
C        TEST WHETHER VARIABLE HAS BEEN USED ON THESE INDIVIDUALS
C        PICK OUT VARIABLES THAT MATCH THE INDIVIDUAL LABEL OR THE
C        FIRST N DIGITS OF IT
C        THESE VARIABLES HAVE 1 PLACED IN ISMSB - OTHERS HAVE ZERO.
C
 9    LABEL = ILCSE(J)
      LABNO = J
      DO 10 I = 1,NSYM
 10   ISMSB(I) = 0
      IF(NSYMU.EQ.0) GO TO 13
      LAB = LABEL
      DO12 I = 1,LEVEL
      DO 11 J = 1,NSYMU
      ISM = ILSYM(1,J)
      IF(ILSYM(2,J).EQ.LAB)ISMSB(ISM) = 1
 11   CONTINUE
      IF(LEVEL.NE.I)LAB = LABEL/10
 12   CONTINUE
C
C        FIND THE VARIABLE OF THE SUBSET THAT IS MOST DISCRIMINATING
C        FOR THESE INDIVIDUALS
C
 13   MAXP = 0
      DO 16 I = 1,NSYM
      IF(ISMSB(I).EQ.1) GO TO 16
      CALL IPOWR(I,LABEL,IP,IT,MSC,NOC,ITABL,IDATA,ICLSS,ILCSE,NCASE,
     1      NCLAS,NSCOR,NSYM,IFAULT)
      IF(IFAULT.NE.0) RETURN
      IF(IP.LE.MAXP) GO TO 16
      MAXP = IP
      IPN = I
      DO 15 J = 1,10
      MEST(J) = MSC(J)
      MINC(J) = NOC(J)
      ITMIN(J) = IT(J)
      DO 15 J1 = 1,10
      MTAB(J1,J) = ITABL(J1,J)
```

```
 15   CONTINUE
 16   CONTINUE
C
C        APPLY ERROR-RATE STOP RULE
C
      MAXV = 0
      DO 20 L = 1,NCLAS
      IT(L) = 0
      DO 17 M = 1,NSCOR
 17   IT(L) = IT(L)+MTAB(L,M)
      IF(IT(L)-MAXV)20,18,19
 18   ITWO = 3-ITWO
      IF(ITWO.EQ.2) GO TO 20
 19   MAXV = IT(L)
      MCLAS = L
 20   CONTINUE
      NMIN = 0
      DO 21 L = 1,NSCOR
 21   NMIN = NMIN+MINC(L)
      IIP = IT(MCLAS)
      IF(MAXP - IIP.GE.NUM) GO TO 27
      WRITE(NPRINT,91)
      GO TO 28
C
C        TEST FOR SIGNIFICANCE FOR ADDITION OF VARIABLE TO KEY
C
 27   IF(LEVEL-1.GE.MAXLEV) GO TO 35
      CALL TRUNC(MTAB,NO,IFAULT)
      IF(IFAULT.EQ.1)RETURN
      IF(NO + ISIG.LT.3) GO TO 36
      IFAULT = 0
      WRITE(NPRINT,92)
C
C        STOP THAT BRANCH
C
 28   IIP = NMIN-IIP
      WRITE(NPRINT,95)LABEL,MCLAS,NMIN,IIP
      DO 30 II = 1,IFLAG
      IF(ILSYM(3,II).EQ.LABEL) GO TO 31
 30   CONTINUE
 31   IF(ITERM(MCLAS).EQ.1) GO TO 32
      WRITE(NPUNCH,85)II,ITERM(MCLAS)
      GO TO 33
 32   ITERM(MCLAS) = II
      WRITE(NPUNCH,81)II,MCLAS
 33   DO 34 II = LABNO,NCASE
 34   IF(ILCSE(II).EQ.LABEL) ILCSE(II) = -MCLAS
      GO TO 48
C
C        MAXIMUM LEVELS REACHED
C
 35   WRITE(NPRINT,93)
      GO TO 28
C
C        WRITE BRANCH STATEMENT
C
 36   WRITE(NPRINT,96)LABEL,IPN,NMIN
      DO 37 II = 1,IFLAG
      IF(ILSYM(3,II).EQ.LABEL) GO TO 38
 37   CONTINUE
 38   DO 41 J = 1,10
```

```
 41   IFST(J) = 1
      WRITE(NPUNCH,82)II,IPN
      DO 43 J = 1,NSCOR
      J1 = J-1
      IF(MINC(J).EQ.0) GO TO 43
      WRITE(NPRINT,97)J1,LABEL,J1,ITMIN(J)
      IFLAG = IFLAG+1
      IF(IFLAG.GT.NKEY) GO TO 60
      ILSYM(3,IFLAG) = LABEL*10+J1
      IFST(J) = IFLAG
 43   CONTINUE
      WRITE(NPUNCH,83)IFST
C
C        WRITE BRANCH TERMINAL  IE.  CLASS  =
C
      DO 46 J = 1,NSCOR
      IF(MINC(J).EQ.0 .OR. ITMIN(J).GT.0) GO TO 46
      LAB = LABEL*10+J-1
      WRITE(NPRINT,95)LAB,MEST(J),MINC(J),ITMIN(J)
      MEJ = MEST(J)
      IMJ = ITERM(MEJ)
      IF(IMJ.EQ.1) GO TO 44
      WRITE(NPUNCH,85)IFST(J),IMJ
      GO TO 46
 44   ITERM(MEJ) = IFST(J)
      WRITE(NPUNCH,81)IFST(J),MEST(J)
 46   CONTINUE
C
C        LABEL THE VARIABLE, CREATING NEW ROW IN ILSYM
C
      NSYMU = NSYMU+1
      IF(NSYMU.GT.NKEY) GO TO 61
      ILSYM(1,NSYMU) = IPN
      ILSYM(2,NSYMU) = LABEL
C
C        ADD A DIGIT TO INDIVIDUAL LABEL
C
      DO 47 I = LABNO,NCASE
      IF(ILCSE(I).NE.LABEL) GO TO 47
      IDA = IDATA(IPN,I) +1
      LAB = ILCSE(I)*10+IDA-1
      IF(ITMIN(IDA).EQ.0) LAB = -MEST(IDA)
      ILCSE(I) = LAB
 47   CONTINUE
 48   LABNO = LABNO+1
      IF(LABNO.LE.NCASE) GO TO 5
      LEVEL = LEVEL+1
      LEV10 = LEV10*10
      IF(LEVEL.LE.NSYM ) GO TO 4
C
C        TERMINATE PROCEDURE - SUCCESSFUL RETURN
C
 49   WRITE(NPRINT,94)
 50   LREJ = NCLAS+1
      DO 51 I = 1,NCASE
      ILCSE(I) = -ILCSE(I)
 51   CONTINUE
      WRITE(NPUNCH,84)LREJ
      RETURN
C
C        FAULT RETURNS
```

```
C
 52   IFAULT = 6
      RETURN
 53   IFAULT = 7
      RETURN
 54   IFAULT = 8
      RETURN
 55   IFAULT = 9
      RETURN
 56   IFAULT = 10
      RETURN
 57   IFAULT = 11
      RETURN
 58   IFAULT = 12
      RETURN
 59   IFAULT = 13
      RETURN
 60   IFAULT = 14
      RETURN
 61   IFAULT = 15
      RETURN
C
C        FORMATS FOR PUNCHED CARD OUTPUT
C
 80   FORMAT(6X,21H FUNCTION ICLAS(ISYM)/6X,16H DIMENSION ISYM(,I5,2H ))
 81   FORMAT(1X,I4,1X,7H ICLAS = ,I2/6X,7H RETURN)
 82   FORMAT(1X,I4,1X,8H I = ISYM(,I3,3H)+1)
 83   FORMAT(6X,8H GO TO (,9(I3,1H,),I3,3H),I)
 84   FORMAT(3X,2H 1,2X,6HICLAS = ,I2/6X,7H RETURN/6X,4H END)
 85   FORMAT(1X,I4,1X,7H GO TO ,I3)
C
C        FORMATS FOR PRINTED OUTPUT
C
 91   FORMAT(1H0,47X,61H * REMAINING VARIABLES DO NOT SUFFICIENTLY IMPRO
     1VE ERROR RATE)
 92   FORMAT(1H0,47X,38H * REMAINING VARIABLES NOT SIGNIFICANT)
 93   FORMAT(1H0,47X,34H * SPECIFIED MAXIMUM LEVEL REACHED)
 94   FORMAT(1H0,47X,31H * ALL VARIABLES HAVE BEEN USED)
 95   FORMAT(1H ,I8,7H CLASS = ,I2,30X,26H * THIS RULE DERIVED USING,I4,
     113H INDIVIDUALS.,I6)
 96   FORMAT(1H0,I8,13H IF VARIABLE ,I3,9H SCORES -,14X,26H * THIS RULE
     1DERIVED USING,I4,20H INDIVIDUALS. ERRORS)
 97   FORMAT(32X,I1,6H GO TO,I7,I1,2X,1H*,43X,I4)
      END
C
      SUBROUTINE IPOWR(I,J,IP,IT,MAXSC,NOC,ITABL,IDATA,ICLSS,ILCSE,NCASE
     1,NCLAS,NSCOR,NSYM,IFAULT)
C
C        ALGORITHM AS 165.1  APPL. STATIST. (1981) VOL.30, NO.3
C
C        COMPUTE DISCRIMINATING POWER, IP, OF VARIABLE I FOR A SUBSET OF THE
C        DATA DEFINED BY INDIVIDUAL LABEL J.
C
      DIMENSION IDATA(NSYM,NCASE),ICLSS(NCASE),ILCSE(NCASE)
      DIMENSION ITABL(10,10),IT(10),MAXSC(10),NOC(10)
      DATA ITWO/1/
C
C        CHECK INPUT VALUES
C
      IF(I.GT.NSYM .OR. I.LT.1) GO TO 9
      IF(NCASE.LT.3) GO TO 10
```

```
      IF(NSYM.LT.3) GO TO 11
      IF(NSCOR.LT.2 .OR. NSCOR.GT.10) GO TO 12
      IF(NCLAS.LT.2 .OR. NCLAS.GT.10) GO TO 13
      DO 1 L = 1,NCASE
      IF(ICLSS(L).LT.1 .OR. ICLSS(L).GT.NCLAS) GO TO 14
      IF(IDATA(I,L).LT.0 .OR. IDATA(I,L).GE.NSCOR) GO TO 15
 1    CONTINUE
C
C        TABULATE CLASS BY VARIABLE SCORE FOR THE APPROPRIATE
C        SUBSET OF THE DATA
C
      DO 2 L = 1,10
      DO 2 M = 1,10
 2    ITABL(L,M) = 0
      DO 3 L = 1,NCASE
      IF(ILCSE(L).NE.J) GO TO 3
      IND = IDATA(I,L)+1
      ICS = ICLSS(L)
      ITABL(ICS,IND) = ITABL(ICS,IND)+1
 3    CONTINUE
      IP = 0
      DO 8 L = 1,NSCOR
      MAX = 1
      NOC(L) = ITABL(1,L)
      DO 7 M = 2,NCLAS
      NOC(L) = NOC(L)+ITABL(M,L)
      IF(ITABL(MAX,L)-ITABL(M,L))5,4,7
 4    ITWO = 3-ITWO
      IF(ITWO.EQ.2) GO TO 7
 5    MAX = M
 7    CONTINUE
      IT(L) = NOC(L) - ITABL(MAX,L)
      MAXSC(L) = MAX
      IP = IP+ITABL(MAX,L)
 8    CONTINUE
      IFAULT = 0
      RETURN
C
C        FAULT RETURNS
C
 9    IFAULT = 5
      RETURN
 10   IFAULT = 6
      RETURN
 11   IFAULT = 7
      RETURN
 12   IFAULT = 8
      RETURN
 13   IFAULT = 9
      RETURN
 14   IFAULT = 10
      RETURN
 15   IFAULT = 11
      RETURN
      END
C
      SUBROUTINE TRUNC(ITABL,NO,IFAULT)
C
C        ALGORITHM AS 165.2  APPL. STATIST. (1981) VOL.30, NO.3
C
      DIMENSION CRIT(36,2),ITABL(10,10),IRT(10),ICT(10)
```

```
      DATA CRIT(1,1),CRIT(2,1),CRIT(3,1),CRIT(4,1),CRIT(5,1),CRIT(6,1),
     1CRIT( 7,1),CRIT( 8,1),CRIT( 9,1),CRIT(10,1),CRIT(11,1),CRIT(12,1),
     1CRIT(13,1),CRIT(14,1),CRIT(15,1),CRIT(16,1),CRIT(17,1),CRIT(18,1),
     1CRIT(19,1),CRIT(20,1),CRIT(21,1),CRIT(22,1),CRIT(23,1),CRIT(24,1),
     1CRIT(25,1),CRIT(26,1),CRIT(27,1),CRIT(28,1),CRIT(29,1),CRIT(30,1),
     1CRIT(31,1),CRIT(32,1),CRIT(33,1),CRIT(34,1),CRIT(35,1),CRIT(36,1)/
     1      3.84,      5.99,      7.81,      9.49,     11.07,     12.59,
     1     14.07,     15.51,     16.92,     18.31,     19.68,     21.03,
     1     22.36,     23.68,     25.00,     26.30,     27.59,     28.87,
     1     30.14,     31.41,     32.67,     33.92,     35.17,     36.42,
     1     37.65,     38.89,     40.11,     41.34,     42.56,     43.77,
     1     55.76,     67.50,     79.08,     90.53,    101.9 ,    113.1/
      DATA CRIT(1,2),
     1CRIT( 2,2),CRIT( 3,2),CRIT( 4,2),CRIT( 5,2),CRIT( 6,2),CRIT( 7,2),
     1CRIT( 8,2),CRIT( 9,2),CRIT(10,2),CRIT(11,2),CRIT(12,2),CRIT(13,2),
     1CRIT(14,2),CRIT(15,2),CRIT(16,2),CRIT(17,2),CRIT(18,2),CRIT(19,2),
     1CRIT(20,2),CRIT(21,2),CRIT(22,2),CRIT(23,2),CRIT(24,2),CRIT(25,2),
     1CRIT(26,2),CRIT(27,2),CRIT(28,2),CRIT(29,2),CRIT(30,2),CRIT(31,2),
     1CRIT(32,2),CRIT(33,2),CRIT(34,2),CRIT(35,2),CRIT(36,2)/
     16.63, 9.21,     11.34,     13.28,     15.09,     16.81,     18.48,
     1     20.09,     21.67,     23.21,     24.73,     26.22,     27.69,
     1     29.14,     30.58,     32.00,     33.41,     34.81,     36.19,
     1     37.57,     38.93,     40.29,     41.64,     42.98,     44.31,
     1     45.64,     46.96,     48.28,     49.59,     50.89,     63.69,
     1     76.15,     88.38,    100.4 ,    112.3 ,    124.1/
      NO = 0
      IFAULT = 0
C
C        COMPUTE ROW AND COLUMN TOTALS, NOTING EMPTY ROWS AND COLUMNS
C
      NROWS = 0
      DO 2 I = 1,10
      IR = 0
      DO 1 J = 1,10
      IF(ITABL(I,J).LT.0) GO TO 13
      IR = IR+ITABL(I,J)
 1    CONTINUE
      IF(IR.GT.0)NROWS = NROWS+1
      IRT(I) = IR
 2    CONTINUE
      IF(NROWS.LT.2) GO TO 15
      NCOLS = 0
      DO 4 I = 1,10
      IC = 0
      DO 3 J = 1,10
      IC = IC+ITABL(J,I)
 3    CONTINUE
      IF(IC.GT.0)NCOLS = NCOLS+1
      ICT(I) = IC
 4    CONTINUE
      IF(NCOLS.LT.2) GO TO 16
      IGT = 0
      DO 5 I = 1,10
 5    IGT = IGT+IRT(I)
      IF(IGT.EQ.0) GO TO 14
C
C        COMPUTE G STATISTIC (LIKELIHOOD RATIO TEST)
C
      G = 0.0
      DO 8 J = 1,10
      IF(ICT(J).EQ.0) GO TO 8
```

```
      FAC = FLOAT(IGT)/FLOAT(ICT(J))
      DO 7 I = 1,10
      IF(IRT(I).EQ.0) GO TO 7
      IJOBS = ITABL(I,J)
      IF(IJOBS.EQ.0) GO TO 7
      OBSIJ = FLOAT(IJOBS)
      AL = OBSIJ*FAC/FLOAT(IRT(I))
      G = G + OBSIJ*ALOG(AL)
 7    CONTINUE
 8    CONTINUE
      G = 2.0*G
      NDF = (NROWS - 1)*(NCOLS - 1)
C
C        COMPARE G WITH TABLE OF CRITICAL VALUES OF CHI-SQUARE,
C        OR INTERPOLATE
C
      IF(NDF.LE.30) GO TO 11
      NC = NDF/10
      NCRITL = NC*10
      IF(NCRITL.EQ.NDF) GO TO 10
      CL1 = CRIT(NC+27,1)
      CL2 = CRIT(NC+27,2)
      CH1 = CRIT(NC+28,1)
      CH2 = CRIT(NC+28,2)
      F = FLOAT(NDF - NCRITL)/10.0
      CRIT1 = CL1 + (CH1 - CL1)*F
      CRIT2 = CL2 + (CH2 - CL2)*F
      GO TO 12
 10   NDF = NC+27
 11   CRIT1 = CRIT(NDF,1)
      CRIT2 = CRIT(NDF,2)
 12   IF(G.LT.CRIT2)NO = 1
      IF(G.LT.CRIT1)NO = 2
      RETURN
C
C        FAULT RETURNS
C
 13   IFAULT = 1
      RETURN
 14   IFAULT = 2
      RETURN
 15   IFAULT = 3
      RETURN
 16   IFAULT = 4
      RETURN
      END
```

```
      SUBROUTINE ENTNGL(IP, L, M, NPTS, IDES, NP, NINTER, ITERMS, NTP,
     *   A, NA, X, NX, W, NW, IFAULT)
C
C        ALGORITHM AS 166  APPL. STATIST. (1981) VOL.30, NO.3
C
C        CALCULATES THE ENTANGLEMENT MATRIX FOR A SPECIFIED DESIGN
C
C     *** Warning ***
C     A scanner was used to enter this text - it may contain errors!
C
      INTEGER L(IP), M(IP), IDES(NP), ITERMS(NTP)
      REAL W(NW), A(NA), X(NX)
      LOGICAL XXONLY
      DATA ZERO, POINT4, ONE /0.0, 0.4, 1.0/
C
C        CHECK PARAMETERS FOR ERRORS
C
      IFAULT = 0
      ID = 0
      XXONLY = NINTER .EQ. 0
      IF (IP .LE. 0 .OR. NPTS .LE. 0 .OR. NINTER .LT. 0 .OR.
     *   NTP .LT. IP * NINTER) IFAULT = 5
      IF (NP .LT. NPTS * IP) IFAULT = 6
      LSUM = 0
      MSUM = 0
      LMSUM = 0
      MAXL = 0
      DO 10 JA = 1, IP
        LJA = IABS(L(JA))
        MJA = M(JA)
        IF (MJA .LE. 0 .OR. MJA .GE. LJA .OR. LJA .EQ. 0 .OR.
     *     LJA .GT. NPTS) IFAULT = 7
        IF (XXONLY) GOTO 3
        IT = JA
        DO 2 JB = 1, NINTER
          J = ITERMS(IT)
          IF (J .LT. 0 .OR. J .GT. MJA) IFAULT = 13
          IT = IT + IP
    2   CONTINUE
    3   IF (LJA .GT. MAXL) MAXL = LJA
        LSUM = LSUM + LJA
        MSUM = MSUM + MJA
        LMSUM = LMSUM + LJA * MJA
        DO 5 I = 1, NPTS
          ID = ID + 1
          LIJ = IDES(ID)
          IF (LIJ .LT. 0 .OR. LIJ .GE. LJA) IFAULT = 1
    5   CONTINUE
   10 CONTINUE
      MSIZE = MSUM + NINTER
      LASTA = MSIZE * (MSIZE + 1) / 2
      IF (NA .LT. LASTA) IFAULT = 8
      XXONLY = NX .EQ. 1
      IF (NX .LT. MSIZE * NPTS .AND. .NOT. XXONLY) IFAULT = 9
      LASTW = LSUM + MAXL + MAXL + 2 + LMSUM + MSUM + NINTER + NINTER
      IF (NW .LT. LASTW) IFAULT = 10
      IF (IFAULT .NE. 0) RETURN
C
C        INITIALIZE LAST PART OF W TO ZERO
C
      IWG = LASTW - NINTER + 1
```

```
      DO 15 IW = IWG, LASTW
   15 W(IW) = ZERO
C
C        CALCULATE ORTHOGONAL POLYNOMIAL COEFFICIENTS FOR EACH FACTOR
C
      LIMWD = LSUM + MAXL + MAXL + 2
      IWD = LIMWD
      ID = 0
      LIMWB = LSUM + 1
      LIMWC = LIMWB + MAXL
      DO 100 JA = 1, IP
        KR = 0
        LJA = IABS(L(JA))
        MJA = M(JA)
        IF (LJA .EQ. 2) GOTO 52
C
C         (1)  FACTORS WITH THREE OR MORE LEVELS
C
        DO 16 J = 1, LJA
          IW = LSUM + J
          W(IW) = ZERO
   16   CONTINUE
        DO 18 I = 1, NPTS
          ID = ID + 1
          IW = IDES(ID) + LIMWB
          W(IW) = W(IW) + ONE
   18   CONTINUE
        CONST = FLOAT(NPTS / LJA)
        IFAULT = 12
        DO 19 J = 1, LJA
          IW = LSUM + J
          IF (ABS(W(IW) - CONST) .GT. POINT4) RETURN
   19   CONTINUE
        CONST = ONE / SQRT(CONST)
        IFAULT = 0
        IF (L(JA) .LT. 0) GOTO 40
      DO 20 J = 1, LJA
        IW = IWA + J
        W(IW) = FLOAT(J)
   20 CONTINUE
   40 IW = IWA + 1
      CALL ORTHON(0, W(IW), W(LIMWB), W(LIMWC), LJA, LJA+2, IFAULT)
      IF (IFAULT .NE. 0 .AND. IFAULT .NE. 2) RETURN
      IF (IFAULT .EQ. 2) IFAIL = 2
   50 KR = KR + 1
      IF (KR .GT. MJA) GOTO 80
      CALL ORTHON(KR, W(IW), W(LIMWB), W(LIMWC), LJA, LJA+2, IFAULT)
      IF (IFAULT .NE. 0 .AND. IFAULT . NE . 2) RETURN
      GOTO 58
C
C        (2) FACTORS WITH TWO LEVELS ONLY (POSSIBLY OCCURRING UNEQUALLY)
C
   52 ISUM = 0
      DO 54 I = 1, NPTS
        ID = ID + 1
        ISUM = ISUM + IDES(ID)
   54 CONTINUE
      CONST = FLOAT(ISUM) / FLOAT(NPTS)
      W(LIMWB) = -CONST
      W(LIMWB + 1) = ONE - CONST
      CONST = 1.0 / SQRT(CONST * FLOAT(NPTS - ISUM))
```

```
C
C        COPY COEFFICIENTS FROM WORKSPACE AREA TO STORAGE AREA
C
   58 DO 60 J = 1, LJA
        IWD = IWD + 1
        IWB = LSUM + J
        W(IWD) = W(IWB) + CONST
   60 CONTINUE
      IF (LJA .NE. 2) GOTO 50
   80 IWA = IWA + LJA
  100 CONTINUE
C
C        PREPARE TO SET UP A
C
      LIMWE = IWD
      DO 120 IA = 1, LASTA
  120 A(IA) = ZERO
C
C       ADD EACH DESIGN POINT INTO A
C
      LIMWD = LIMWD + 1
      DO 240 I = 1, NPTS
      ID = 0
C
C        (1) MAIN EFFECTS
C
      IWD = LIMWD
      IWE = LIMWE
      DO 140 JA = 1, IP
        LJA = IABS(L(JA))
        MJA = M(JA)
        LIJ = ID + I
        LIJ = IDES(LIJ)
        DO 130 J = 1, MJA
          IWE = IWE + 1
          IW = IWD + LIJ
           W(IWE) = W(IW)
           IWD = IWD + LJA
  130   CONTINUE
        ID = ID + NPTS
  140 CONTINUE
C
C       (2)  INTERACTIONS
C
      IF  (NINTER .EQ. 0)  GOTO  200
      IT = 0
      IWF = IWE
      IWG = IWF + NINTER
      DO 180 JB = 1, NINTER
        IWF = IWF + 1
        IWG = IWG + 1
        WW = ONE
        IWE = LIMWE
        DO 160 JA = 1, IP
          IT = IT + 1
          IF (ITERMS(IT) .EQ. 0) GOTO 150
          IW = IWE + ITERMS(IT)
          WW = WW * W(IW)
  150     IWE = IWE + M(JA)
  160   CONTINUE
        W(IWF) = WW
```

```
          W(IWG) = W(IWG) + WW
  180 CONTINUE
C
C        CALCULATE THE TERMS IN THE SSP MATRIX
C
  200 IA = 1
      DO 220 JA = 1, MSIZE
        IW = LIMWE + JA
        DO 210 JB = 1, JA
          IW2 = LIMWE + JB
          A(IA) = A(IA) + W(IW) * W(IW2)
          IA = IA + 1
  210   CONTINUE
  220 CONTINUE
C
C
C        STORE CONTRASTS IN X
C
      IF (XXONLY) GOTO 240
      I X  =  I
      DO 230 JA = 1, MSIZE
        IW = LIMWE + JA
        X(IX) = W(IW)
        IX = IX + NPTS
  230 CONTINUE
  240 CONTINUE
C
C        ADJUST INT X INT TERMS IN A ABOUT THEIR MEANS
C        (TERMS INVOLVING MAIN EFFECTS ARE ALREADY CENTRED)
C
      IF (NINTER .EQ. 0)  RETURN
      J = MSUM + 1
      IA = J * MSUM / 2
      IA2 = IA
      IWG = IWG - MSIZE
      CONST = ONE / FLOAT(NPTS)
      DO 260 JA = J, MSIZE
        IA = IA + J - 1
        IW = IWG + JA
        DO 250 JB = J, JA
          IA = IA + 1
          IW2 = IWG + JB
          A(IA) = A(IA) - W(IW) * W(IW2) * CONST
  250   CONTINUE
  260 CONTINUE
C
C        CALCULATE 1 / SQRT(DIAG(A)) AND STORE
C
      IWE = LIMWE
      IA = 0
      DO 320 JA = 1, MSIZE
        IA = IA + JA
        IWE = IWE + 1
        W(IWE) = 1.0 / SQRT(LA(IA))
  320 CONTINUE
C
C        CONVERT A FROM A SSP-TYPE MATRIX TO A CO
C
      DO 360 JA = J, MSIZE
        IW = LIMWE + JA
        WW = W(IW)
```

```
          DO 350 JB = 1, JA
            IW2 = LIMWE + JB
            IA2 = IA2 + 1
            A(IA2) = A(IA2) * WW * W(IW2)
  350     CONTINUE
  360   CONTINUE
C
C         STANDARDIZE INTERACTION CONTRASTS
C
        IF (XXONLY) RETURN
        IWG = IWE
        IWE = IWE - NINTER
        IX = (J - 1) * NPTS
        DO 400 JA = 1, NINTER
          IWE = IWE +  1
          IWG = IWG + 1
          XBAR = W(IWG) + 1
          WW = W(IWE)
          DO 380 I = 1, NPTS
            IX = IX + 1
            X(IX) = WW * (X(IX) - XBAR)
  380     CONTINUE
  400   CONTINUE
        RETURN
        END
```

```
      SUBROUTINE FIT(A, NA, NTERMS, MSIZE, IE, INC, NINC, NADD,
     *  GVAR, KEY, EFF, EPS, IFAULT)
C
C        ALGORITHM AS 167  APPL. STATIST. (1981)  VOL.30, NO.3
C
C        CALCULATES EFFICIENCIES OF ESTIMATIOM OF TERMS IN
C        THE CURRENT MODEL AND ITS GENERALIZED VARIANCE
C
C     *** Warning: This text has been read using a scanner and may
C                  contain errors
C
      REAL A(NA), EFF(NTERMS)
      INTEGER INC(NINC), IE(NTERMS), KEY(NTERMS)
      DATA ONE /1.0/
      IFAULT = 0
      IA = NTERMS * (NTERMS + 1) / 2
      IF (NA .LT. IA) IFAULT = 2
      IF (MSIZE .LT. 0 .OR. NADD .LE. 0 .OR. NADD .GT. NINC)
     +    IFAULT = 3
      IF (MSIZE .NE. 0 .OR. IFAULT .NE. 0) GOTO 5
      ONEPLS = ONE + EPS
      ONEMIN = ONE - EPS
      IA = 0
      DO 3 J1 = 1, NTERMS
      DO 1 J2 = 1, J1
      IA = IA + 1
      R = A(IA)
      IF (R .LT. -ONEPLS .OR. R .GT. ONEPLS) IFAULT = 1
    1 CONTINUE
      IF (R .LT. ONEMIN) IFAULT = 1
      IE(J1) = -1
    3 CONTINUE
      GVAR = ONE
    5 IF (IFAULT .NE. 0) RETURN
      DO 20 J = 1, NADD
      JP = INC(J)
      IF (JP .LE. 0 .OR. JP .GT. NTERMS) GOTO 50
      IF (IABS(IE(JP)) .NE. 1) GOTO 50
      IA = JP * (JP + 1) / 2
      IF (ABS(A(IA)) .LE. EPS) GOTO 19
      IF (IE(JP) .LT. 0) GOTO 16
C
C        DROP TERM FROM MODEL
C
      K = MSIZE - 1
      DO 12 J1 = 1, K
      IF (JP .NE. KEY(J1)) GOTO 12
      DO 8 J2 = J1, K
    8 KEY(J2) = KEY(J2 + 1)
      GOTO 13
   12 CONTINUE
   13 MSIZE = K
      GVAR = -GVAR / A(IA)
      CALL PIVOT(A, NA, NTERMS, JP, IE)
      GOTO 20
C
C        ADD TERM TO MODEL
C
   16 MSIZE = MSIZE + 1
      KEY(MSIZE) = JP
      CALL PIVOT(A, NA, NTERMS, JP, IE)
```

```
      GVAR = -GVAR * A(IA)
      GOTO 20
C
C        LEAVE TERM IN OR OUT OF MODEL AS PIVOT TOO SMALL
C
   19 IFAULT = -JP
   20 CONTINUE
C
C        CALCULATE EFFICIENCIES OF EACH TERM IN MODEL
C
      DO 40 J = 1, MSIZE
      IA = IA * (IA + 1) / 2
      EFF(J) = -ONE / A(IA)
   40 CONTINUE
      RETURN
   50 IFAULT = 4
      RETURN
      END
C
      SUBROUTINE PIVOT(A, NA, K, IP, IE)
C
C        ALGORITHM AS 167.1  APPL. STATIST. (1981) VOL.30, NO.3
C
C        ROW BY ROW INVERSION IN SITU OF A SYMMETRIC MATRIX STORED AS
C        LOWCR TRIANGLE. A FORTRAN TRANSLATION OF ALGORITHM AS 37
C        (GARSIDE, 1971)
      REAL A(NA)
      INTEGER IE(K)
C
      DATA ONE /1.0/
      JP = IABS(IP)
      JPP = JP * (JP + 1) / 2
      JOP = JPP - JP
      AA = ONE / A(JPP)
      A(JPP) = -AA
      NIJ = 0
      IF (JP .EQ. 1) GOTO 100
      JP1 = JP - 1
      DO 80 I = I, JP1
      NIP = JOP + I
      AIP = -A(NIP) * FLOAT(IE(JP))
      A(NIP) = AIP * AA
      DO 60 J = 1, I
      NIJ = NIJ + 1
      NJP = JOP + J
      A(NIJ) = A(NIJ) - AIP * A(NJP)
   60 CONTINUE
   80 CONTINUE
  100 IF (JP .EQ. K) GOTO 190
      NIJ = NIJ + JP
      JP2 = JP + 1
      NIP = JPP
      DO 150 I = JP2, K
      NIP = NIP + I - 1
      AIP = -A(NIP) * FLOAT(IE(JP))
      A(NIP) = AIP * AA
      IF (JP .EQ. 1) GOTO 130
      DO 110 J = 1, JP1
      NIJ = NIJ + 1
      NJP = JOP + J
      A(NIJ) = A(NIJ) - AIP * A(NJP)
```

```
  110 CONTINUE
  130 NIJ = NIJ + 1
      NJP = JPP + JP
      DO 140 J = JP2, I
      NIJ = NIJ + 1
      A(NIJ) = A(NIJ) - AIP * A(NJP)
      NJP = NJP + J
  140 CONTINUE
  150 CONTINUE
  190 IE(JP) = -IE(JP)
      RETURN
      END
```

```
      subroutine scale(fmn, fmx, n, mpv, valmin, step, nvals,
     *  ir, ifault)
c
c        algorithm as 168  appl. statist. (1981) vol.30, no.3
c
c        calculates neat values for lowest print position and
c        step between printed values on a scale to include fmn
c        and fmx
c
      real unit(12)
      real tol, bias
      data nunit /12/
      data unit(1), unit(2), unit(3), unit(4), unit(5), unit(6),
     *   unit(7), unit(8), unit(9), unit(10), unit(11), unit(12)
     *  /12.0, 15.0, 20.0, 25.0, 30.0, 40.0,
     *  50.0, 60.0, 80.0, 100.0, 120.0, 150.0/
      data tol /5.0e-6/, bias /1.0e-5/
      data minn /2/, maxn /10000/, cover /0.7/
c
      fmax = fmx
      fmin = fmn
c
c        test for valid parameters
c
      ifault = 0
      if (fmax .lt. fmin) ifault = ifault + 1
      if (n .lt. minn .or. n .gt. maxn) ifault = ifault + 2
      if (mpv .le. 0 .or. mpv .ge. n) ifault = ifault + 4
      if (ifault .ne. 0) return
      nvals = (n - 1) / mpv + 1
c
c        test for values effectively equal
c
      if (fmax - fmin .gt. tol * amax1(abs(fmax), abs(fmin))) goto 20
      ifault = -1
      if (fmax) 5, 10, 15
    5 fmax = 0.0
      goto 20
   10 fmax = 1.0
      goto 20
   15 fmin = 0.0
c        find neat trial step size
c
   20 finter = float(n) / float(mpv)
      s = (fmax - fmin) * (1.0 + 2.0 * bias) / finter
      ir = 0
   25 if (s .gt. 10.0) goto 30
      s = s * 10.0
      ir = ir + 1
      goto 25
   30 if (s .le. 100.0) goto 35
      s = s / 10.0
      ir = ir - 1
      goto 30
   35 do 40 i = 1, nunit
      if (s .le. unit(i)) goto 45
   40 continue
   45 step = 10.0 ** (-ir) * unit(i)
c
c        find neat trial start value
c
```

```
      aj = 0.0
   50 aj = aj + 1.0
      if (unit(i) - 0.1 .gt. aint((unit(i) + 0.1) / aj) * aj) goto 50
      tstep = step / aj
      temp = fmin / tstep + aj * (0.5 / float(mpv) - finter * bias)
      valmin = aint(temp) * tstep
      if (temp .lt. 0.0 .and. temp .ne. aint(temp))
     * valmin = valmin - tstep
c
c        test whether fmax is in scale
c
      if (fmax .lt. valmin + step *
     *  (finter * (1.0 - bias) - 0.5 / float(mpv))) goto 55
c
c        try new step or start value
c
      if (unit(i) / unit(i + 1) * (1.0 - 1.0 / (aj * finter)) .lt.
     *  cover) goto 50
      i = i + 1
      goto 45
c
c        get position of least significant figure on scale
c
   55 do 60 j = 1, 2
        aj = aj * 10.0
        if (unit(i) - 0.1 .lt. aint((unit(i) + 0.1) / aj) * aj)
     *          ir = ir - 1
   60 continue
      return
      end
c
      subroutine axis(valmin, step, nvals, maxpr, ir, irprin, offset,
     *  ifact, vals, iv, ifault)
c
c        algorithm as 168.1 appl. statist. (1981) vol.30, no.3
c
c        sets up values and formats for printing on an axis
c
      real vals(iv)
      data irmax /20/, mprmax /20/
c
c        check for valid parameters
c
      ifault = 0
      if (nvals .lt. 2) ifault = ifault + 1
      fmax = valmin + step * float(nvals - 1)
      if (nvals .ge. 2 .and. fmax .le. valmin) ifault = ifault + 2
      if (maxpr .lt. 2 .or. maxpr .gt. mprmax) ifault = ifault + 4
      if (nvals .gt. iv) ifault = ifault + 8
      if (ir .gt. irmax) ifault = ifault + 16
      if (ifault .gt. 0) return
c        find position of most significant digit(il) and number of
c        significant digits overall(is) and varying(it)
c
      tmax = 10.0 ** maxpr
      fl = abs(fmax)
      fs = abs(valmin)
      il = 0
   10 if (fl .lt. 1.0 .and. fs .lt. 1.0) goto 20
      fl = fl / 10.0
      fs = fs / 10.0
```

```
      il = il + 1
      goto 10
c
   20 if (fl .ge. 0.1 .or. fs .ge. 0.1) goto 30
      fl = fl * 10.0
      fs = fs * 10.0
      il = il - 1
      goto 20
c
   30 is = il + ir
      it = is
      if (valmin .le. 0.0 .and. fmax .ge. 0.0) goto 50
   40 fl = mod(fl, 1.0) * 10.0
      fs = mod(fs, 1.0) * 10.0
      if (it .le. 0) goto 1016
      if (int(fl) .ne. int(fs)) goto 50
      it = it - 1
      goto 40
c
c        decide on printing format
c
   50 ifact = 0
      offset = 0.0
      irprin = max(ir, 0)
      ilprin = max(il, 0)
c
      if (irprin + ilprin .le. maxpr) goto 70
      if (is .le. maxpr) goto 60
      irprin = maxpr - 1
      ifact = max(it, maxpr) - 1 - ir
      goto 70
   60 ifact = il - 1
      irprin = is - 1
   70 fs = 10.0 ** (-ifact)
      vstep = step * fs
      vmin = valmin * fs
      if (is .le. maxpr) goto 80
      offset = aint(vmin / 10.0) * 10.0
      vmin = vmin - offset
c
c        write values for axis
c
   80 do 90 i = 1, nvals
        vals(i) = vmin
        vmin = vmin + vstep
   90 continue
c
c        check that all values can be printed
c
      fs = 0.1 ** irprin
      if (abs(vals(1)) * fs + 0.5 .lt. tmax .and.
     *  abs(vals(nvals)) * fs + 0.5 .lt. tmax) return
      il = il + 1
      is = is + 1
      it = it + 1
      goto 50
c
c        error indicator
c
 1016 ifault = 16
      return
```

```
end
```

```
      subroutine scatpl(a, n, m, icy, ncy, icx, ny, nx, scaley,
     *  scalex, istand, ifault, iwrite)
c
c       algorithm as 169  appl. statist. (1981) vol.30, no.3
c
c       produces a scatter plot of one variable against several
c
      character*1 iout,intch,markch,iblank,idot,icolon
      character*1 icomma,iapost,isemi,itwo,idash,form21*14,form22*6
      character iform1*20,iform2*21,form11*8,form12*11
      dimension iout(161), vals(20), a(n, m), icy(ncy), scalex(2),
     *  scaley(2), intch(11), markch(5)
      data maxwid /132/, maxht /50/, maxy /5/
      data mpvx /10/, mpvy /5/
      data intch(1), intch(2), intch(3), intch(4), intch(5), intch(6),
     *  intch(7), intch(8), intch(9), intch(10), intch(11)
     *  /'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '9'/
      data markch(1), markch(2), markch(3), markch(4), markch(5)
     *  /'*', '0', '+', 'x', '='/
      data iblank /' '/, idot /'.'/, icolon /':'/, icomma /','/,
     *  iapost /''''/, isemi /';'/, itwo /'2'/, idash /'-'/
c
c       formats for printing
c
      data iform1 /'(1h ,f8.0,1x,152a1) '/
      data iform2 /'(1h ,5x,16(f8.0,2x)) '/
c
    1 format(' ', 10x, ':', 151a1)
    2 format(' times 10**', i3)
    3 format(' offset', f10.0)
    4 format(' ', 14x, 'times 10**', i3)
    5 format(' ', 14x, 'offset', f10.0)
    6 format(' ', 2x, 16(9x, a1))
c
c       test for valid parameters
c
      ifault = 0
      if (n .lt. 1) ifault = ifault + 1
      if (m .lt. 2) ifault = ifault + 2
      if (icx .lt. 1 .or. icx .gt. m) ifault = ifault + 4
      if (ncy .le. 0 .or. ncy .gt. maxy) ifault = ifault + 8
      if (ifault .gt. 0) return
      do 10 i = 1, ncy
        if (icy(i) .lt. 1 .or. icy(i) .gt. m) goto 1016
   10 continue
c
c       set plot size
c
      nly = maxht - 5
      if (nly .gt. ny) nly = ny
      if (nly .le. mpvy) nly = mpvy + 1
      nlx = maxwid - 11
      if (nlx .gt. nx) nlx = nx
      if (nlx .le. mpvx) nlx = mpvx + 1
c
c       calculate horizontal scale
c
      xmin = scalex(1)
      xmax = scalex(2)
      if (xmax .gt. xmin) goto 30
      xmin = a(1, icx)
```

```
      xmax = xmin
      if (n .eq. 1) goto 30
      do 20 i = 2, n
        ai = a(i, icx)
        if (ai .lt. xmin) xmin = ai
        if (ai .gt. xmax) xmax = ai
   20 continue
c
   30 call scale(xmin, xmax, nlx, mpvx, temp, xvstep,
     *  nxvals, irx, ifail)
      if (ifail .gt. 0) goto 1032
      xmin = temp
      xstep = xvstep / float(mpvx)
c
c        calculate vertical scale
c
      ymin = scaley(1)
      ymax = scaley(2)
      if (ymax .gt. ymin) goto 50
      k = icy(1)
      ymin = a(1, k)
      ymax = ymin
      do 40 j = 1, ncy
        k = icy(j)
        do 40 i = 1, n
          ai = a(i, k)
          if (ai .lt. ymin) ymin = ai
          if (ai .gt. ymax) ymax = ai
   40 continue
c
   50 call scale(ymin, ymax, nly, mpvy, temp, yvstep,
     *  nyvals, iry, ifail)
      if (ifail .gt. 0) goto 1064
      ymin = temp
      ystep = yvstep / float(mpvy)
c
c        calculate printed values and set mark for 2 points
c
      call axis(ymin, yvstep, nyvals, 6, iry, irpr, offset,
     *  ifact, vals, 20, ifail)
      if (ifail .gt. 0) goto 1064
      read(iform1,'(a8,1x,a11)')form11,form12
      write(iform1,'(a8,a1,a11)')form11,intch(irpr + 1),form12
      if (ifact .ne. 0) write (iwrite, 2) ifact
      if (offset .ne. 0.0)  write (iwrite, 3) offset
      if (istand .eq. 0) intch(3) = isemi
c
c        for each line of output
c
      iplted = 0
      do 140 i = 1, nly
        iy = nly - i
        do 60 ix = 1, nlx
   60   iout(ix) = iblank
c
c        scan data for points on current line
c
        do 120 l = 1, n
          indx = (a(l, icx) - xmin) / xstep + 1.5
          if (indx .lt. 1 .or. indx .gt. nlx) goto 120
          do 110 j = 1, ncy
```

```
            k = icy(j)
            y = (a(l, k) - ymin) / ystep
            indy = y + 0.5
            if (indy .ne. iy) goto 110
            iplted = iplted + 1
            if (iout(indx) .ne. iblank) goto 80
c
c        single point
c
            if (istand .eq. 0) goto 70
            iout(indx) = markch(j)
            goto 110
   70       iout(indx) = icomma
            if (int(y) .eq. iy) iout(indx) = iapost
            goto 110
c
c        multiple points
c
   80       do 90 ic = 3, 10
              if (iout(indx) .eq. intch(ic)) goto 100
   90       continue
            ic = 2
  100       iout(indx) = intch(ic + 1)
  110     continue
  120   continue
c
c        print line
c
        if (mod(iy, mpvy) .eq. 0) goto 130
        write (iwrite, 1) (iout(ix), ix = 1, nlx)
        goto 140
  130   write (iwrite, iform1) vals(nyvals), idash, icolon,
     *       (iout(ix), ix = 1, nlx)
        nyvals = nyvals - 1
  140 continue
c
c        print horizontal axis using variable formats
c
      write (iwrite, 1) (idot, i = 1, nlx)
      call axis(xmin, xvstep, nxvals, 6, irx, irpr, offset,
     *  ifact, vals, 20, ifail)
      intch(3) = itwo
      read(iform2,'(a14,1x,a6)')form21,form22
      write(iform2,'(a14,a1,a6)')form21,intch(irpr + 1),form22
      if (ifail .gt. 0) goto 1032
      write (iwrite, 6) (icolon, i = 1, nxvals)
      write (iwrite, iform2) (vals(i), i = 1, nxvals)
      if (ifact .ne. 0) write (iwrite, 4) ifact
      if (offset .ne. 0.0) write (iwrite, 5) offset
      ifault = iplted - n * ncy
      return
c
c        set error indicator
c
 1064 ifault = ifault + 32
 1032 ifault = ifault + 16
 1016 ifault = ifault + 16
      return
      end
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      FUNCTION CHISQN(X, DF, FL, IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C
C        ALGORITHM AS 170  APPL. STATIST. (1981) VOL.30, NO.3
C
C        The non-central chi-squared distribution.
C
C     Auxiliary routines required: GAMMDS = AS147, ALOGAM = CACM 291.
C     See AS245 for an alternative to ALOGAM.
C
      CHISQN = 0.0
      IFAULT = 0
C
C        TEST FOR ADMISSIBILITY OF ARGUMENTS
C
      IF (DF.LE.0.0) IFAULT = 1
      IF (X.LT.0.0) IFAULT = 2
      IF (FL.LT.0.0) IFAULT = 3
      IF (IFAULT.GT.0.OR.X.EQ.0.0) RETURN
C
      DF2 = 0.5*DF
      X2 = 0.5*X
      FXP = GAMMDS(X2,DF2,IFAULT)
      CHISQN = CHI(X2,DF2,FL,FXP)
      RETURN
      END
C
      SUBROUTINE CHISQL(X, DF, FX, FL, IFAULT)
C
C        ALGORITHM AS 170.1  APPL.STATIST. (1981) VOL.30, NO.3
C
C        DEFINE ACCURACY AND INITIALIZE
C
C        N SHOULD BE SPECIFIED SUCH THAT ACC IS GREATER THAN
C        OR EQUAL TO (AU-AL)/2**N
C
      PARAMETER (ACC = 1.0E-6, N = 30)
C
      AL = 0.0
      AINC = 80.0
      AU = 80.0
C
      IFAULT = 0
C
C        TEST FOR ADMISSIBILITY OF ARGUMENTS
C
      IF (DF.LE.0.0) IFAULT = 1
      IF (X.LT.0.0) IFAULT = 2
      IF (FX.LE.0.0) IFAULT = 3
      IF (IFAULT.GT.0) GO TO 4
C
      DF2 = 0.5*DF
      X2 = 0.5*X
      FX1 = GAMMDS(X2,DF2,IFAULT)
    1 APROX = CHI(X2,DF2,AU,FX1)
      IF (FX.GT.APROX) GOTO 2
```

```
        IF (FX.LT.APROX) AL = AU
        AU = AU+AINC
        GO TO 1
      2 DO 3 J = 1,N
          FL = 0.5*(AL+AU)
          APROX = CHI(X2, DF2, FL, FX1)
          IF (ABS(FX-APROX).LT.ACC) GO TO 4
          IF (FX.LT.APROX) AL = FL
          IF (FX.GE.APROX) AU = FL
      3 CONTINUE
      4 RETURN
        END
C
        FUNCTION CHI(X, DF, FL, FXC)
C
C          ALGORITHM AS 170.2  APPL. STATIST. (1981) VOL.30, NO.3
C
        PARAMETER (ACC2 = 1.0E-8)
C
        CHI = FXC
        DF1 = DF
        FL2 = 0.5*FL
        C = 1.0
        T = 0.0
      1 T = T+1.0
        C = C*FL2/T
        DF1 = DF1+1.0
        TERM = C*GAMMDS(X, DF1, IFAULT)
        CHI = CHI+TERM
        IF (TERM.GE.ACC2) GO TO 1
        CHI = CHI*EXP(-FL2)
        RETURN
        END
```

```
      SUBROUTINE TPSHV(N, T, NTMAX, SY2, PROB, KGP, KPR, A, KF, DLF,
     *     IFAULT)
C
C     ALGORITHM AS 171  APPL. STATIST. (1982) VOL.31, NO.3
C
C     Fisher's exact variance test for the Poisson distribution
C
      INTEGER T, SY2, TS, A(T), KF(T)
      REAL DD, DC, DM, DMIN, DP, D1, DZ, DPRB, DLF(NTMAX)
      DATA MAXP /1000000/, MAXT /80/
C
      IFAULT = 0
      DPRB = DZ
      DM = D1 - DMIN
      IF (N .LT. 2 .OR. N .GT. NTMAX) IFAULT = 2
      IF (T .LT. 2 .OR. T .GT. NTMAX .OR. T .GT. MAXT) IFAULT = 3
      IF (IFAULT .NE. 0) GO TO 200
      KGP = 0
      KPR = 0
C
C     Generate log factorials and constant
C
      DLF(1) = DZ
      DO 10 I = 2, NTMAX
   10 DLF(I) = DLF(I-1) + LOG(FLOAT(I))
      DC = - T * LOG(FLOAT(N)) + DLF(N) + DLF(T)
C
C     Generate partitions of sample total (T) starting with M the
C     number of non-zero values in the sample.
C
      M = MIN(T, N)
      DO 15 J = 1, M
   15 KF(J) = 0
   20 DO 25 J = 1, M
   25 A(J) = 1
      TS = M - 1
   30 A(M) = T
      DO 35 J = 2, M
   35 A(M) = A(M) - A(J-1)
      TS = TS + A(M)**2
      KGP = KGP + 1
C
C     The current M-part partition of T:
C         A(1) + A(2) + ... + A(M) = T
C     with sums of squares  TS = SUM A(J)**2
C     is admissible if TS < SY2
C
      IF (TS .GE. SY2) GO TO 80
C
C     Convert sample into frequency distribution.
C     KZ is the number of zero values in the sample.
C     KF(J) is the number of A()'s = J.
C
      KZ = N - M
      MX = A(M)
      DO 60 J = 1, M
        IAK = A(J)
        KF(IAK) = KF(IAK) + 1
   60 CONTINUE
C
C     Compute DP = probability for this sample
```

```
C
      KPR = KPR + 1
      DD = DZ
      IF (KZ .GT. 0) DD = DLF(KZ)
      IAK = KF(1)
      IF (KF(1) .GT. 0) DD = DD + DLF(IAK)
      KF(1) = 0
      IF (MX .LT. 2) GO TO 70
      DO 65 J = 2, MX
        IF (KF(J) .EQ. 0) GO TO 65
        IAK = KF(J)
        DD = DD + DLF(IAK) + KF(J) * DLF(J)
        KF(J) = 0
   65 CONTINUE
   70 DP = EXP(DC - DD)
      DPRB = DPRB + DP
      IF (DPRB .LT. DM) GO TO 80
      IFAULT = 1
      DPRB = DM
      GO TO 200
   80 CONTINUE
      IT = M - 1
   90 IF (IT .EQ. 0) GO TO 120
C
C     Determine next M-partition & update sum of squares.
C
      IF ((A(M) - A(IT)) .GT. 1) GO TO 100
      IT = IT - 1
      GO TO 90
  100 IAT = A(IT) + 1
      M1 = M - 1
      DO 110 J = IT, M1
        TS = TS - A(J)**2 + IAT**2
        A(J) = IAT
  110 CONTINUE
      TS = TS - A(M)**2
C
C     New values for A(1), ..., A(M-1) are determined.
C
      IF (KGP .LT. MAXP) GO TO 30
      IFAULT = 1
      GO TO 200
  120 M = M - 1
C
C     End evaluation of M-part partitions.
C     Decrease M by 1 and continue if SY2 > sum of squares for
C     the last partition.
C
      IF (TS .GE. SY2) GO TO 200
      IF (M .GT. 1) GO TO 20
C
  200 PROB = D1 - DPRB
      RETURN
      END
```

```
      SUBROUTINE SIMDO(QIND, QFOR, IPROD, KDIM, JSUB, IVEC, IFAULT)
C
C     ALGORITHM AS 172  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Generates the subscript vector (index subscript) given the index
C     subscript (subscript vector) for a sequence of Fortran DO-loops
C     nested to depth KDIM
C
      LOGICAL QIND, QFOR
      INTEGER IPROD(KDIM), IVEC(KDIM)
C
      IFAULT = 0
      IF (.NOT. QIND) GO TO 12
C
C     Index subscript to subscript vector conversion
C
      IF (JSUB .LE. IPROD(KDIM)) GO TO 5
      IFAULT = 1
      RETURN
    5 ITEMPV = JSUB - 1
      IJ = KDIM - 1
      DO 10 I = 1, IJ
        IK = KDIM - I
        IVEC(I) = ITEMPV / IPROD(IK)
        ITEMPV = ITEMPV - IPROD(IK) * IVEC(I)
        IVEC(I) = IVEC(I) + 1
   10 CONTINUE
      IVEC(KDIM) = ITEMPV + 1
      IF (QFOR) CALL REVERS(IVEC, KDIM)
      RETURN
C
C     Subscript vector to index subscript conversion
C
   12 IF (IVEC(1) .LE. IPROD(1)) GO TO 14
      IFAULT = 2
      RETURN
   14 DO 15 I = 2, KDIM
        IF (IVEC(I) .LE. IPROD(I)/IPROD(I-1)) GO TO 15
        IFAULT = 2
        RETURN
   15 CONTINUE
      IF (.NOT. QFOR) CALL REVERS(IVEC, KDIM)
      JSUB = IVEC(1)
      DO 20 I = 2, KDIM
        JSUB = JSUB + (IVEC(I) - 1) * IPROD(I-1)
   20 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE REVERS(ITAB, IDIM)
      INTEGER ITAB(IDIM)
C
C     ALGORITHM AS 172.1  APPL. STATIST. (1982) VOL.31, NO. 1
C
C     Reorders subscript vector, if required
C
      ITER = IDIM / 2
      K = IDIM + 1
      DO 10 I = 1, ITER
        ITEMP = ITAB(I)
```

```
            IK = K - I
            ITAB(I) = ITAB(IK)
            ITAB(IK) = ITEMP
         10 CONTINUE
      C
            RETURN
            END
```

```
      SUBROUTINE DESMAT(MAXROW,MAXCOL,NDIM,NSUB,MXDINT,MSUB,
     1LSUB,ISUB,IBEG,MATDES,IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C       ALGORITHM AS 173  APPL. STATIST. (1982) VOL.31, NO.1
C
C       GENERATES A DESIGN MATRIX FOR BALANCED FACTORIAL
C       EXPERIMENTS
C
      DIMENSION MATDES(MAXROW,MAXCOL),NSUB(NDIM),MSUB(NDIM),
     1          LSUB(NDIM),ISUB(NDIM),IBEG(NDIM)
C
C     PRIMARY INPUT PARAMETER CHECK
C
      IFAULT = 0
      DO 5 I = 1,NDIM
      IF(NSUB(I) .GE.2) GO TO 5
      IFAULT = 1
      RETURN
    5 CONTINUE
      IF(MXDINT .LE.NDIM) GO TO 8
      IFAULT = 2
      RETURN
C
C     CALCULATE PRODUCT VECTOR MSUB FOR M.E. GENERATION
C     CALCULATE PRODUCT VECTOR LSUB FOR INTRT. GENERATION
C
    8 IBEG(1) = 1
      MSUB(1) = NSUB(1)
      LSUB(1) = NSUB(1)-1
      DO 40 I = 2,NDIM
      IK  = I-1
      IBEG(I) = IBEG(IK)+NSUB(IK)-1
      MSUB(I) = MSUB(IK) * NSUB(I)
      LSUB(I) = LSUB(IK) *(NSUB(I)-1)
   40 CONTINUE
C
C     RESERVE THE TOTAL NUMBER OF OBSERVATIONS
C
      NOBS = MSUB(NDIM)
C
C     SECONDARY INPUT PARAMETER CHECK
C
      IF(NOBS .LE. MAXROW) GO TO 50
      IFAULT = 3
      RETURN
C
C     GENERATE THE MAIN EFFECTS
C
   50 CALL MANEFF(MAXROW,MAXCOL,NDIM,NSUB,MSUB,ISUB,MATDES,JCOL)
C
C     GENERATE ALL POSSIBLE INTERACTIONS
C
      IF(MXDINT.GE.2)CALL INTER(MAXROW,MAXCOL,NDIM,NOBS,NSUB,MXDINT,IBEG
     1,MSUB,LSUB,ISUB,MATDES,JCOL,IFAULT)
C
C       PASS BACK NOBS AND JCOL IRRESPECTIVE OF VALUE OF IFAULT
C
      IBEG(1) = NOBS
```

```
      IBEG(2) = JCOL
      RETURN
      END
      SUBROUTINE MANEFF(MAXROW,MAXCOL,NDIM,NSUB,MSUB,ISUB,MATDES,
     1JCOL)
C
C        ALGORITHM AS 173.1  APPL. STATIST. (1982) VOL.31, NO.1
C
C       COMPUTES THOSE COLUMNS OF THE DESIGN MATRIX (MATDES) ASSOCIATED
C       WITH THE MEAN AND THE MAIN EFFECTS
C
      DIMENSION MATDES(MAXROW,MAXCOL),NSUB(NDIM),MSUB(NDIM),ISUB(NDIM)
C
C        SET THE TOTAL NUMBER OF OBSERVATIONS
C
      NOBS = MSUB(NDIM)
C
C     SET UP MUU
C
      DO 10 I = 1,NOBS
      MATDES(I,1) = 1
   10 CONTINUE
C
C     SET UP OTHER MAIN EFFECTS
C
      DO 40 I = 1,NOBS
      CALL SIMDO(.TRUE.,.TRUE.,MSUB,NDIM,I,ISUB,JFALT)
C
C        ROUTINE PROTECTED FROM FAILURE SO FAULT PARAMETER UNTESTED
C
      JCOL = 1
      DO 30 J = 1,NDIM
      IK = NSUB(J)-1
      DO 20 K = 1,IK
      JCOL = JCOL+1
      IF (ISUB(J) .EQ. K) GO TO 15
      IF (ISUB(J) .EQ. NSUB(J))GO TO 18
      MATDES(I,JCOL) = 0
      GO TO 20
   15 MATDES(I,JCOL) = 1
      GO TO 20
   18 MATDES(I,JCOL) = -1
   20 CONTINUE
   30 CONTINUE
   40 CONTINUE
      RETURN
      END
      SUBROUTINE INTER(MAXROW,MAXCOL,NDIM,NOBS,NSUB,MXDINT,IBEG,
     1ICOMB,LSUB,IREF,MATDES,JCOL,IFAULT)
C
C        ALGORITHM AS 173.2  APPL. STATIST. (1982) VOL.31, NO.1
C
C        GENERATES THE SPECIFIED INTERACTION TERMS
C
      DIMENSION MATDES(MAXROW,MAXCOL),NSUB(NDIM),IBEG(NDIM),ICOMB(NDIM)
     1,LSUB(NDIM),IREF(NDIM)
      LOGICAL QNXT
C
C     GENERATE ALL POSSIBLE COMBINATIONS OF DIMENSION .LE. MXDINT
C
      QNXT = .FALSE.
```

```
      DO 50 ICMB = 2,MXDINT
    5 CALL COMB(NDIM,ICMB,ICOMB,QNXT)
C
C     HERE TO PROCESS CURRENT COMBINATION OF DIMENSION ICMB
C
C     CALCULATE THE ADDITIONAL COLUMNS REQUIRED TO STORE THIS
C     INTERACTION
C
      NPROD = 1
      DO 10 I = 1,ICMB
      IK = ICOMB(I)
      NPROD = NPROD*(NSUB(IK)-1)
   10 CONTINUE
C
C     CHECK IF SUFFICIENT COLUMNS ARE AVAILABLE IN MATDES
C
      IF (JCOL+NPROD .LE. MAXCOL) GO TO 15
      IFAULT = 4
      RETURN
C
C     PROCESS THE CURRENT INTERACTION FOR WHICH THERE IS SPACE
C
   15 DO 40 J = 1,NPROD
      CALL SIMDO(.TRUE.,.TRUE.,LSUB,NDIM,J,IREF,JFALT)
C
C        ROUTINE PROTECTED FROM FAILURE SO FAULT PARAMETER UNTESTED
C
      JCOL = JCOL+1
      DO 30 I = 1,NOBS
      MATDES(I,JCOL) = 1
      DO 20 L = 1,ICMB
      IK = ICOMB(L)
      JREF = IBEG(IK) + IREF(L)
      MATDES(I,JCOL) = MATDES(I,JCOL)*MATDES(I,JREF)
   20 CONTINUE
   30 CONTINUE
   40 CONTINUE
C
C     FINISHED ADDING IN THIS INTERACTION - PROCESS NEXT ONE
C
      IF(QNXT)GO TO 5
   50 CONTINUE
      RETURN
      END
      SUBROUTINE COMB(N,K,A,MTC)
C
C        ALGORITHM AS 173.3  APPL. STATIST. (1982) VOL.31, NO.1
C
C        GENERATES ALL POSSIBLE COMBINATIONS OF DIMENSION K
C        FROM N INTEGERS
C
      INTEGER A(K),H
      LOGICAL MTC
C
      IF(.NOT.MTC) GO TO 20
      DO 10 H = 1,K
      M = K + 1 - H
      M2 = A(M)
      IF(M2.NE.N+1-H)GO TO 30
   10 CONTINUE
   20 M2 = 0
```

```
      H = K
   30 DO 40 J = 1,H
      M = K + J - H
      A(M) = M2 + J
   40 CONTINUE
      MTC = A(1).NE.N-K+1
      RETURN
      END
```

```
      SUBROUTINE NONPAR(ISTAT, IT, NPOP, IN, IP, Y, NAR, ST, IDF,
     *   VM, AR, W, VP, VR, VY, VV, IFAULT)
C
C     ALGORITHM AS 174  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Computes the distribution-free test statistics for either the
C     multivariate multi-sample rank sum test (MMRST) or the multi-
C     variate multi-sample median test (MMMT)
C
      INTEGER IN(NPOP)
      REAL Y(IT,IP), VM(IP,IP), AR(NAR), W(IP), VP(NPOP,IP), VR(IP),
     *     VY(IT), VV(NAR)
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/
C
      IFAULT = 3
      IF (ISTAT .NE. 0 .AND. ISTAT .NE. 1) GO TO 22
      IFAULT = 4
      IF (NPOP .LT. 2) GO TO 22
      IFAULT = 5
      IF (IP .LT. 1) GO TO 22
      IFAULT = 6
      ISUM = 0
      DO 1 I = 1, NPOP
    1 ISUM = ISUM + IN(I)
      IF (ISUM .NE. IT) GO TO 22
      IFAULT = 7
      JP = IP * (IP+1) / 2
      IF (NAR .NE. JP) GO TO 22
      IFAULT = 0
C
      DO 6 J = 1, IP
        DO 5 I = 1, IT
          R = HALF
          DO 4 II = 1, IT
            IF (Y(I,J) - Y(II,J)) 4, 2, 3
    2       R = R + HALF
            GO TO 4
    3       R = R + ONE
    4     CONTINUE
          VY(I) = R
    5   CONTINUE
        DO 6 I = 1, IT
          Y(I,J) = VY(I)
    6 CONTINUE
C
C     Matrix Y now has the ranks replacing the observed values
C
      B = IT
      IF (ISTAT .EQ. 1) GO TO 9
      A = HALF * B
      DO 8 J = 1, IP
        DO 8 IJ = 1, IT
          IF (Y(IJ,J) .LE. A) GO TO 7
          Y(IJ,J) = ZERO
          GO TO 8
    7     Y(IJ,J) = ONE
    8 CONTINUE
C
C     If MMMT test is selected (ISTAT = 0), the Y-matrix will now
C     contain 1's and 0's.   1 indicates an observation <= the median;
C     0 if greater than the median.
```

```
C
    9 DO 13 I = 1, IP
        IA = 0
        DO 11 IB = 1, NPOP
          IST = IN(IB)
          CST = ZERO
          DO 10 IC = 1, IST
            IA = IA + 1
            CST = CST + Y(IA,I)
   10     CONTINUE
          VP(IB,I) = CST
   11   CONTINUE
        DO 13 IR = 1, IP
          V = ZERO
          DO 12 JR = 1, IT
   12     V = V + Y(JR,I) * Y(JR,IR)
          VM(I,IR) = V
   13 CONTINUE
C
C     Matrix VP contains the sum of the Y-matrix elements for each
C     population and each multivariate response.
C
C     Matrix VM contains the sum of squares and cross-products of
C     the columns of Y, that is VM = Y'Y.
C
      DO 14 I = 1, IP
   14 VR(I) = ZERO
      DO 15 I = 1, IP
        DO 15 IPR = 1, NPOP
          VR(I) = VR(I) + VP(IPR,I)
   15 CONTINUE
C
C     Vector VR contains the sum of elements in each column of Y.
C
      KPR = 0
      DO 16 J = 1, IP
        DO 16 I = 1, J
          KPR = KPR + 1
          AR(KPR) = VM(I,J)/B - VR(I)*VR(J)/B**2
   16 CONTINUE
C
C     Vector AR contains the upper triangular portion of VM adjusted for
C     the overall means read in columnwise.   This is the upper triangle
C     of the dispersion matrix required for subroutine SYMINV.
C
      DO 17 I = 1, NPOP
        AA = IN(I)
        DO 17 J = 1, IP
          VP(I,J) = VP(I,J)/AA - VR(J)/B
   17 CONTINUE
C
C     Matrix VP now contains the sum of the Y-matrix elements for each
C     population and multivariate response adjusted for the overall
C     population response mean.
C
      CALL SYMINV(AR, IP, VV, W, NULLTY, IFAULT)
      IF (IFAULT .NE. 0) GO TO 22
      K = 0
      DO 18 J = 1, IP
        DO 18 I = 1, J
          K = K + 1
```

```
            VM(I,J) = VV(K)
            VM(J,I) = VV(K)
   18 CONTINUE
C
C     Matrix VM now is the inverse of the dispersion matrix.
C
      ST = ZERO
      DO 21 IPR = 1, NPOP
        TSTAT = ZERO
        DO 20 JPR = 1, IP
          ACC = ZERO
          DO 19 KPR = 1, IP
   19     ACC = ACC + VM(KPR,JPR) * VP(IPR,KPR)
          TSTAT = TSTAT + ACC * VP(IPR,JPR) * IN(IPR)
   20   CONTINUE
        ST = ST + TSTAT
   21 CONTINUE
      IDF = IP * (NPOP - 1)
C
   22 RETURN
      END
```

```
      SUBROUTINE FACSYM(A, B, W, N1, ITER0, ITER1, ITER2, EPS, IFAULT)
C
C     ALGORITHM AS 175  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Cramer-Wold factorization; 3-stage algorithm.
C     Factorizes: A(x) = a0 + a1(x + 1/x) + a2(x^2 + 1/x^2) + ...
C                      = B(x).B(1/x)
C     where       B(x) = b0 + b1.x + b2.x^2 + ... + bn.x^n
C
C     Auxiliary routine required: SDSDOT from IMSL; an equivalent function
C     has been added to this file.
C
      REAL A(N1), B(N1), W(N1,2), HALF
      DATA HALF /0.5E0/
C
C     Generate initial values for B using Bauer algorithm
C
      IFAULT = 0
      DO 10 I = 1, N1
   10 W(I,1) = A(I)
      CALL BAUER(W, B, N1, ITER0, IER)
      IF (IER .NE. 0) GO TO 130
C
C     Iterate to convergence using Wilson algorithm
C
      IF (ITER1 .LE. 0) GO TO 50
      DO 40 ITER = 1, ITER1
        DO 20 I = 1, N1
          W(I,1) = A(I)
          W(I,2) = B(I)
   20   CONTINUE
        CALL RECURS(B, W, N1, IER)
        IF (IER .NE. 0) GO TO 120
        DO 30 I = 1, N1
   30   B(I) = B(I) + HALF * W(I,2)
        IF (B(1) .LE. ABS(B(N1))) GO TO 120
        IF (W(1,2) - B(1) .LE. EOS * W(1,2) .AND. ITER .GT. 1) GO TO 50
   40 CONTINUE
      GO TO 110
C
C     Iterative improvement using incremental form
C
   50 IF (ITER2 .EQ. 0) GO TO 140
      DO 80 ITER = 1, ITER2
        DO 60 I = 1, N1
          W(I,1) = - A(I)
          W(I,2) = B(I)
   60   CONTINUE
        CALL RESIDU(W, B, N1)
        CALL RECURS(B, W, N1, IER)
        IF (IER .NE. 0) GO TO 120
        DO 70 I = 1, N1
   70   B(I) = W(I,2) - B(I)
        IF (B(1) .LE. ABS(B(N1))) GO TO 120
        IF (B(1) .GE. W(1,2) .AND. ITER .GT. 1) GO TO 140
   80 CONTINUE
      GO TO 140
C
C     Exits from subroutine
C
  110 IFAULT = 1
```

```
      GO TO 140
  120 IFAULT = 2
      GO TO 140
  130 IFAULT = 3
  140 RETURN
      END
C
      SUBROUTINE BAUER(A, B, N1, MAXIT, IFAULT)
C
C     ALGORITHM AS 175.1  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Bauer-Rissanen algorithm.
C
      REAL A(N1), B(N1), C, ONE, ZERO
      DATA ONE /1.0E0/, ZERO /0.0E0/
C
      N = N1 - 1
      IFAULT = 0
      DO 10 I = 1, N
   10 B(I) = A(I+1)
      B(N1) = ZERO
      IF (MAXIT .EQ. 0) GO TO 30
      DO 20 ITER = 1, MAXIT
        IF (A(1) .LE. ABS(A(N1))) GO TO 100
        C = - B(1) / A(1)
        IF (ABS(C) .GT. ONE) GO TO 100
        DO 20 I = 1, N
          A(I) = A(I) + C * B(I)
          B(I) = B(I+1) + C * A(I+1)
   20 CONTINUE
   30 IF (A(1) .LE. ZERO) GO TO 100
      C = SQRT(A(1))
      DO 40 I = 1, N1
   40 B(I) = A(I) / C
      RETURN
C
  100 IFAULT = 1
      RETURN
      END
C
      SUBROUTINE RECURS(B, W, N1, IFAULT)
C
C     ALGORITHM AS 175.2  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Recursive solution for polynomial coefficients
C
      REAL B(N1), W(N1), S, HALF, ZERO
      DATA HALF /0.5E0/, ZERO /0.0E0/
C
      N = N1 - 1
      M = N1
      W(1) = HALF * W(1)
      DO 20 I = 1, N
        IF (B(1) .LE. ZERO) GO TO 40
        S = W(M) / B(1)
        K = M
        DO 10 J = 1, M
          W(J) = W(J) - S * B(K)
          K = K - 1
   10   CONTINUE
        W(M) = S
```

```
            S = B(M) / B(1)
            CALL LINCOM(B, S, M)
            B(M) = S
            M = M - 1
      20 CONTINUE
C
         IF (B(1) .LE. ZERO) GO TO 40
         B(1) = W(1) / B(1)
         DO 30 I = 2, N1
            S = B(I)
            B(I) = ZERO
            CALL LINCOM(B, S, I)
            B(I) = B(I) + W(I)
      30 CONTINUE
         IFAULT = 0
         RETURN
C
      40 IFAULT = 1
         RETURN
         END
C
         SUBROUTINE LINCOM(X, P, M)
C
C     ALGORITHM AS 175.3  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Replaces X with X - P * XSTAR
C
         REAL X(M), P, S
C
         I = 1
         J = M
      10 S = X(I)
         X(I) = X(I) - P * X(J)
         IF (I .NE. J) X(J) = X(J) - P * S
         I = I + 1
         J = J - 1
         IF (I .LE. J) GO TO 10
C
         RETURN
         END
C
         SUBROUTINE RESIDU(A, B, N1)
C
C     ALGORITHM AS 175.4  APPL. STATIST. (1982) VOL.31, NO.1
C
C     Replaces A with A + B * BSTAR
C
         REAL A(N1), B(N1), SDSDOT
C
         DO 10 I = 1, N1
      10 A(I) = SDSDOT(N1-I+1, A(I), B(1), 1, B(I), 1)
C
         RETURN
         END
C
C
         REAL FUNCTION SDSDOT(N, SB, SX, INCX, SY, INCY)
C
C     Evaluate the single-precision dot-product B + X'Y using double
C     precision internally.
C     Modified from the BLAS function SDOT by Alan Miller, CSIRO
```

```
C     Division of Mathematics & Statistics, Melbourne, Australia
C
      INTEGER N, INCX, INCY
      REAL SB, SX(*), SY(*)
C
C     Local variables
C
      DOUBLE PRECISION DDOT
      INTEGER I, IX, IY, M, MP1, NS
C***FIRST EXECUTABLE STATEMENT  SDSDOT
      DDOT = SB
      IF(N.LE.0) RETURN
      IF(INCX.EQ.INCY) IF(INCX-1)5, 20, 60
    5 CONTINUE
C
C        CODE FOR UNEQUAL INCREMENTS OR NONPOSITIVE INCREMENTS.
C
      IX = 1
      IY = 1
      IF(INCX.LT.0) IX = (-N+1)*INCX + 1
      IF(INCY.LT.0) IY = (-N+1)*INCY + 1
      DO 10 I = 1, N
        DDOT = DDOT + DBLE(SX(IX)) * DBLE(SY(IY))
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      SDSDOT = DDOT
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 5.
C
   20 M = MOD(N, 5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1, M
        DDOT = DDOT + DBLE(SX(I)) * DBLE(SY(I))
   30 CONTINUE
      IF( N .LT. 5 ) THEN
        SDSDOT = DDOT
        RETURN
      END IF
   40 MP1 = M + 1
      DO 50 I = MP1, N, 5
        DDOT = DDOT + DBLE(SX(I)) * SY(I) + DBLE(SX(I+1)) * SY(I+1) +
     1   DBLE(SX(I+2)) * SY(I+2) + DBLE(SX(I+3)) * SY(I+3) +
     2   DBLE(SX(I+4)) * SY(I+4)
   50 CONTINUE
      SDSDOT = DDOT
      RETURN
C
C        CODE FOR POSITIVE EQUAL INCREMENTS .NE.1.
C
   60 CONTINUE
      NS = N*INCX
      DO 70 I = 1, NS, INCX
        DDOT = DDOT + DBLE(SX(I)) * DBLE(SY(I))
   70   CONTINUE
      SDSDOT = DDOT
      RETURN
```

```
END
```

```
      SUBROUTINE DENEST(DT, NDT, DLO, DHI, WINDOW, FT, SMOOTH,
     *  NFT, ICAL, IFAULT)
      REAL DT(NDT), FT(NFT), SMOOTH(NFT)
C
C     ALGORITHM AS 176  APPL. STATIST. (1982) VOL.31, NO.1
C     Modified using AS R50 (Appl. Statist. (1984))
C
C     Find density estimate by kernel method using Gaussian kernel.
C     The interval on which the estimate is evaluated has end points
C     DLO and DHI.   If ICAL is not zero then it is assumed that the
C     routine has been called before with the same data and end points
C     and that the array FT has not been altered.
C
C     Auxiliary routines called: FORRT & REVRT from AS 97
C
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/, SIX/6.0/, THIR2/32.0/
      DATA BIG/30.0/, KFTLO/5/, KFTHI/11/
C
C     The constant BIG is set so that exp(-BIG) can be calculated
C     without causing underflow problems and can be considered = 0.
C
C     Initialize and check for valid parameter values.
C
      IF (WINDOW .LE. ZERO) GO TO 92
      IF (DLO .GE. DHI) GO TO 93
      II = 2**KFTLO
      DO 1 K = KFTLO, KFTHI
        IF (II .EQ. NFT) GO TO 2
        II = II + II
    1 CONTINUE
      IFAULT = 1
      RETURN
    2 STEP = (DHI - DLO) / FLOAT(NFT)
      AINC = ONE / (NDT * STEP)
      NFT2 = NFT / 2
      HW = WINDOW / STEP
      FAC1 = THIR2 * (ATAN(ONE) * HW / NFT) ** 2
      IF (ICAL .NE. 0) GO TO 10
C
C     Discretize the data
C
      DLO1 = DLO - STEP * HALF
      DO 3 J = 1, NFT
    3 FT(J) = ZERO
      DO 4 I = 1, NDT
        WT = (DT(I) - DLO1) / STEP
        JJ = INT(WT)
        IF (JJ .LT. 1 .OR. JJ .GT. NFT) GO TO 4
        WT = WT - JJ
        WINC = WT * AINC
        KK = JJ + 1
        IF (JJ .EQ. NFT) KK = 1
        FT(JJ) = FT(JJ) + AINC - WINC
        FT(KK) = FT(KK) + WINC
    4 CONTINUE
C
C     Transform to find FT.
C
      CALL FORRT(FT, NFT)
C
C     Find transform of density estimate.
```

```
C
   10 JHI = SQRT(BIG / FAC1)
      JMAX = MIN(NFT2 - 1, JHI)
      SMOOTH(1) = FT(1)
      RJ = ZERO
      DO 11 J = 1, JMAX
        RJ = RJ + ONE
        RJFAC = RJ * RJ * FAC1
        BC = ONE - RJFAC / (HW * HW * SIX)
        FAC = EXP(-RJFAC) / BC
        J1 = J + 1
        J2 = J1 + NFT2
        SMOOTH(J1) = FAC * FT(J1)
        SMOOTH(J2) = FAC * FT(J2)
   11 CONTINUE
C
C     Cope with underflow by setting tail of transform to zero.
C
      IF (JHI + 1 - NFT2) 21, 23, 20
   20 SMOOTH(NFT2 + 1) = EXP(-FAC1 * FLOAT(NFT2)**2) * FT(NFT2 + 1)
      GO TO 24
   21 J2LO = JHI + 2
      DO 22 J1 = J2LO, NFT2
        J2 = J1 + NFT2
        SMOOTH(J1) = ZERO
        SMOOTH(J2) = ZERO
   22 CONTINUE
   23 SMOOTH(NFT2 + 1) = ZERO
C
C     Invert Fourier transform of SMOOTH to get estimate and eliminate
C     negative density values.
C
   24 CALL REVRT(SMOOTH, NFT)
      DO 25 J = 1, NFT
   25 IF (SMOOTH(J) .LT. ZERO) SMOOTH(J) = ZERO
      IFAULT = 0
      RETURN
C
   92 IFAULT = 2
      RETURN
   93 IFAULT = 3
      RETURN
      END
```

```
      subroutine nscor1 (s, n, n2, work, ifault)
c
c        Algorithm AS 177   Appl. Statist. (1982) Vol. 31, No. 2
c
c        Exact calculation of Normal Scores
c
      double precision s(n2), work(4,721)
      double precision zero, one, c, scor, ai1, ani, an, h
      data one/1.0d0/, zero/0.0d0/, h/0.025d0/, nstep/721/
c
      ifault=3
      if (n2 .ne. n/2) return
      ifault=1
      if (n .le. 1) return
      ifault=0
      if (n .gt. 2000) ifault=2
c
      an=n
      c=log(an)
c
c        Accumulate ordinates for calculation of integral for rankits
c
      do 20 i=1, n2
        i1=i-1
        ni=n-i
        ai1=i1
        ani=ni
        scor=zero
        do 10 j=1,nstep
   10   scor=scor+exp(work(2,j) + ai1 * work(3,j) + ani * work(4,j)
     *        + c) * work(1,j)
        s(i)=scor * h
        c=c+log(ani/dble(i))
   20 continue
      return
      end
c
c
      subroutine init(work)
c
c        Algorithm AS 177.1   Appl. Statist. (1982) Vol. 31, No. 2
c
      double precision work(4,721)
      double precision xstart, h, pi2, half, xx, alnorm
      data xstart/-9.0d0/, h/0.025d0/, pi2/-0.918938533d0/,
     *     half/0.5d0/,  nstep/721/
      xx=xstart
c
c        Set up arrays for calculation of integral
c
      do 10 i=1,nstep
        work(1,i)=xx
        work(2,i)=pi2 - xx * xx * half
        work(3,i)=log(alnorm(xx, .true.))
        work(4,i)=log(alnorm(xx, .false.))
        xx=xstart + dble(i) * h
   10 continue
      return
      end
c
c
```

```
      double precision function alnorm(x,upper)
c
c         Algorithm AS66 Applied Statistics (1973) vol22 no.3
c
c       Evaluates the tail area of the standardised normal curve
c       from x to infinity if upper is .true. or
c       from minus infinity to x if upper is .false.
c
      double precision zero,one,half
      double precision con,z,y,x
      double precision p,q,r,a1,a2,a3,b1,b2,c1,c2,c3,c4,c5,c6
      double precision d1,d2,d3,d4,d5
      logical upper,up
c*** machine dependent constants
      double precision ltone,utzero
      data zero/0.0d0/, one/1.0d0/, half/0.5d0/
      data ltone/7.0d0/,utzero/18.66d0/
      data con/1.28d0/
      data p/0.398942280444d0/,q/0.39990348504d0/,r/0.398942280385d0/
      data a1/5.75885480458d0/,a2/2.62433121679d0/,a3/5.92885724438d0/
      data b1/-29.8213557807d0/,b2/48.6959930692d0/
      data c1/-3.8052d-8/,c2/3.98064794d-4/,c3/-0.151679116635d0/
      data c4/4.8385912808d0/,c5/0.742380924027d0/,c6/3.99019417011d0/
      data d1/1.00000615302d0/,d2/1.98615381364d0/,d3/5.29330324926d0/
      data d4/-15.1508972451d0/,d5/30.789933034d0/
c
      up=upper
      z=x
      if(z.ge.zero)goto 10
      up=.not.up
      z=-z
   10 if(z.le.ltone.or.up.and.z.le.utzero)goto 20
      alnorm=zero
      goto 40
   20 y=half*z*z
      if(z.gt.con) goto 30
c
      alnorm=half-z*(p-q*y/(y+a1+b1/(y+a2+b2/(y+a3))))
      goto 40
   30 alnorm=r*exp(-y)/(z+c1+d1/(z+c2+d2/(z+c3+d3/(z+c4+d4/(z+c5+d5/
     2   (z+c6))))))
   40 if(.not.up)alnorm=one-alnorm
      return
      end
c
c
      subroutine nscor2(s,n,n2,ier)
c
c     algorithm as 177.3, applied statistics, v.31, 161-165, 1982.
c
c     calculates approximate expected values of normal order statistics.
c     claimed accuracy is 0.0001, though usually accurate to 5-6 dec.
c
c***  N.B. This routine is NOT in double precision ***
c
c     arguments:
c     s(n2)   = output, the first n2 expected values.
c     n       = input, the sample size.
c     n2      = input, the number of order statistics required; must
c                      be <= n/2.
c     ier     = output, error indicator
```

```
c                   = 0 if no error detected
c                   = 1 if n <= 1.
c                   = 2 if n > 2000, in which case the order statistics
c                          are still calculated, but may be inaccurate.
c                   = 3 if n2 > n/2 (n.b. this differs from the
c                          published algorithm which returns an error
c                          if n2 is not equal to n/2.)
c
c     calls ppnd = applied statistics algorithm 111.
c     An alternative is ppnd7 in algorithm AS 241.
c
      real s(n2), eps(4), dl1(4), dl2(4), gam(4), lam(4),
     *   bb, d, b1, an, ai, e1, e2, l1, correc, ppnd
      data eps/0.419885e0, 0.450536e0, 0.456936e0, 0.468488e0/,
     1 dl1/0.112063e0, 0.121770e0, 0.239299e0, 0.215159e0/,
     2 dl2/0.080122e0, 0.111348e0, -0.211867e0, -0.115049e0/,
     3 gam/0.474798e0, 0.469051e0, 0.208597e0, 0.259784e0/,
     4 lam/0.282765e0, 0.304856e0, 0.407708e0, 0.414093e0/,
     5 bb/-0.283833/, d/-0.106136/, b1/0.5641896/
c
c     input parameter checks.
c
      ier = 3
      if(n2.gt.n/2) return
      ier = 1
      if(n.le.1) return
      ier = 0
      if(n.gt.2000) ier = 2
      s(1) = b1
      if(n.eq.2) return
c
c     calculate normal tail areas for first 3 order statistics.
c
      an = n
      k = 3
      if(n2.lt.k) k = n2
      do 5 i = 1,k
        ai = i
        e1 = (ai - eps(i))/(an + gam(i))
        e2 = e1**lam(i)
        s(i) = e1 + e2*(dl1(i) + e2*dl2(i))/an - correc(i,n)
    5 continue
      if(n2.eq.k) go to 20
c
c     calculate normal areas for other cases.
c
      do 10 i = 4,n2
        ai = i
        l1 = lam(4) + bb/(ai + d)
        e1 = (ai - eps(4))/(an + gam(4))
        e2 = e1**l1
        s(i) = e1 + e2*(dl1(4) + e2*dl2(4))/an - correc(i,n)
   10 continue
c
c     convert tail areas to normal deviates.
c
   20 do 30 i = 1,n2
   30 s(i) = -ppnd(s(i),ier)
      return
      end
c
```

```
c
      real function correc(i,n)
c
c     calculates correction for tail area of the i-th largest of n
c     order statistics.
c
      dimension c1(7),c2(7),c3(7)
      real mic
      data c1/9.5, 28.7, 1.9, 0., -7.0, -6.2, -1.6/,
     1 c2/-6195., -9569., -6728., -17614., -8278., -3570., 1075./,
     2 c3/9.338e4, 1.7516e5, 4.1040e5, 2.1576e6, 2.376e6, 2.065e6,
     3 2.065e6/, mic/1.e-6/, c14/1.9e-5/
c
      correc = c14
      if(i*n.eq.4) return
      correc = 0.0
      if(i.lt.1.or.i.gt.7) return
      if(i.ne.4.and.n.gt.20) return
      if(i.eq.4.and.n.gt.40) return
      an = n
      an = 1.0/(an*an)
      correc = (c1(i) + an*(c2(i) + an*c3(i)))*mic
      return
      end
c
c
      function ppnd(p,ier)
c
c     algorithm as 111, appl.statist., vol.26, 118-121, 1977.
c
c     produces normal deviate corresponding to lower tail area = p.
c
      data split/0.42/
      data a0,a1,a2,a3/2.506628, -18.61500, 41.39120, -25.44106/,
     1 b1,b2,b3,b4/-8.473511, 23.08337, -21.06224, 3.130829/,
     2 c0,c1,c2,c3/-2.787189, -2.297965, 4.850141, 2.321213/,
     3 d1,d2/3.543889, 1.637068/
      ier=0
      q=p-0.5
      if(abs(q).gt.split) go to 10
c
c     0.08 < p < 0.92
c
      r=q*q
      ppnd=q*(((a3*r+a2)*r+a1)*r+a0)/((((b4*r+b3)*r+b2)*r+b1)*r+1.)
      return
c
c     p < 0.08 or p > 0.92, set r = min(p,1-p)
c
   10 r=p
      if(q.gt.0.) r=1.-p
      if(r.le.0.) go to 20
      r=sqrt(-log(r))
      ppnd=(((c3*r+c2)*r+c1)*r+c0)/((d2*r+d1)*r+1.)
      if(q.lt.0.) ppnd=-ppnd
      return
   20 ier=1
      ppnd=0.
      return
      end
```

```
      double precision function alnfac(j)
c
c        algorithm as 177.2  appl. statist. (1982) vol.31, no.2
c
c        natural logarithm of factorial for non-negative agrument
c
      implicit logical (a-z)
      integer j
      double precision r(7), one, half, a0, three, four, fourtn, fortty,
     *  fivfty, w, z
      data r(1), r(2), r(3), r(4), r(5), r(6), r(7) /0.0d0, 0.0d0,
     *  0.69314718056d0, 1.79175946923d0, 3.17805383035d0,
     *  4.78749174278d0, 6.57925121101d0/
      data one, half, a0, three, four, fourtn, fortty, fivfty /
     *  1.0d0, 0.5d0, 0.918938533205d0, 3.0d0, 4.0d0, 14.0d0, 420.0d0,
     *  5040.0d0/
      if (j .ge. 0) goto 10
      alnfac = one
      return
   10 if (j .ge. 7) goto 20
      alnfac = r(j + 1)
      return
   20 w = j + 1
      z = one / (w * w)
      alnfac = (w - half) * log(w) - w + a0 + (((four - three * z)
     *  * z - fourtn) * z + fortty) / (fivfty * w)
      return
      end
```

```
      subroutine gsweep (s, t, k, l, m, n, e, ifault)
c
c        Algorithm AS 178   Appl. Statist.  (1982)  Vol. 31, No. 2
c
c        Performs Gauss-Jordan pivot for row/col K in N by N working
c        array stored lower triangle only row by row, shortest row
c        first, in locations 1 to (N*N+N)/2 of T
c
      double precision a, b, e, s(n), t(m), zero, one
      data zero/0.0d0/, one/1.0d0/
c
      ifault=1
      if(n.lt.1 .or. m.lt.(n*n+n)/2) return
      if(k.lt.1 .or. k.gt.n) return
      if(e.lt.zero) return
      ifault=0
c
c        Parameters in range so test for collinearity
c
      l=k
      kk=(k*k+k)/2
      if(t(kk).lt.zero) goto 20
      if(t(kk).lt.e*s(k)) return
      ii=0
      ik=kk-k
      do 10 l=1,n
        ii=ii+l
        ik=ik+1
        if (l.gt.k) ik=ik+l-2
        if (t(ii).ge.zero) goto 10
        if(one/(t(ik)*t(ik)/t(kk)-t(ii)).lt.e*s(l)) return
   10 continue
c
c        No collinearity so update triangle
c
   20 l=0
      t(kK)=-one/t(kk)
      a=abs(t(kk))
      ik=kk-k
      ij=0
      do 90 i=1,n
        ik=ik+1
        if(i-k) 50,30,40
   30   ij=ij+k
        goto 90
   40   ik=ik+i-2
   50   b=t(ik)
        if(t(kk).lt.zero) b=-b
        t(ik)=a*t(ik)
        jk=kk-k
        do 80 j=1,i
          ij=ij+1
          jk=jk+1
          if(j-k) 70,80,60
   60     jk=jk+j-2
   70     t(ij)=t(ij)+b*t(jk)
   80   continue
   90 continue
      return
      end
```

```
##### User Supplied Comment and Possible Correction #####
  I think I've found a bug in routine 193 permut. There is an
array overflow caused statement label "7", which currently reads

    7 do 8 j = 1,t

t is equal to n at the end of the algorithm, therefore j
equals n.  However, the a(k,n) array is referenced in the following lines
as a(i,j+1) which is outside the array bounds.

The correct line should read:

    7 do 8 j = 1,t-1

I've tested the change and the program appears to behave the same.  The
program appeared to work previously but this relied on IFAULT being zero
and immediately after A in memory.

cheers,
Mark
Mark Lakata <lakata@sseos.lbl.gov>
#####

      subroutine permut(n,k,r,a,ifault)
c
c     Algorithm AS 179   Appl. Statist. (1982) Vol.31, No.2
c
c     A single call generates all possible permutations of N
c     objects, partitioned into K non-empty subsets, and passed
c     as array R.
c
      integer a(k,n),r(k),t,t2,tj1
c
c     check for fault conditions
c
      ifault = 1
      do 1 i = 1,k
        if (r(i).eq.0) return
    1 continue
      ifault = 2
      isum = 0
      do 2 i = 1,k
    2 isum = isum + r(i)
      if (isum.ne.n) return
      ifault = 0
c
c     step 1.  To initialise array A, fill rows, 1 to k-1, with
c              paired marks.
c
      kount = 1
      l = n
      k1 = k-1
      do 5 i = 1,k1
        ir = r(i)
        do 3 j = 1,ir
    3   a(i,j) = i
        ir1 = r(i) + 1
        do 4 j = ir1,l
    4   a(i,j) = i+1
        l = l - r(i)
    5 continue
```

```
c
c     step 2.  Begin generation of permutations by setting looping
c              indices i and t.  i indicates a row in array a, j
c              indicates a column in array a, and t indicates the
c              number of marks in row i.
c              If this is the first permutation, go to step 5.
c
    6 i = k - 1
      t = r(k) + r(i)
      if (kount.eq.1) goto 14
c
c     step 3.  Search for an i mark followed by an i+1 mark in
c              row i and interchange the two marks.  If an
c              interchange is made, left-shift any marks out of
c              sequence, otherwise go to step 4.
c
    7 do 8 j = 1,t
        j1 = j + 1
        if (a(i,j).ne.i.or.a(i,j1).ne.i+1) goto 8
        limit = j - 2
        a(i,j) = a(i,j1)
        a(i,j1)= a(i,j) - 1
        if (limit) 14, 14, 12
    8 continue
c
c     step 4.  Interchange not made, so return row i to its initial
c              configuration in step 1 and go to step 1.
c
      if (t.eq.1) goto 10
      t2 = t/2
      do 9 j = 1,t2
        tj1 = t - j + 1
        itemp = a(i,j)
        a(i,j) = a(i,tj1)
        a(i,tj1) = itemp
    9 continue
c
c     Reset looping indices i and t.  Return if i = 0, otherwise
c     go to step 3.
c
   10 i = i -1
      if (i.gt.0) goto 11
      return
   11 t = t + r(i)
      goto 7
c
c     Interchange made, so left-shift any marks out of sequence
c     and go to step 5.
c
   12 iflag = 0
      do 13 j = 1,limit
        j1 = j + 1
        if (a(i,j).ne.i + 1.or.a(i,j1).ne.i) goto 13
        a(i,j) = a(i,j1)
        a(i,j1)= a(i,j) + 1
        iflag = 1
   13 continue
      if (iflag.eq.1) goto 12
c
c     step 5.  Fill last row (k) of array a with marks from row 1
c              and generate current permutation in last row (k)
```

```
c              of array a.
c
   14 do 15 j = 1,n
   15 a(k,j) = a(1,j)
c
      if (k.eq.2) goto 18
      do 17 l = 2,k1
        m = 1
        do 16 j = 1,n
          if (a(k,j).ne.l) goto 16
          a(k,j) = a(l,m)
          m = m + 1
   16   continue
   17 continue
c
c     step 6.  At this point, call subroutine job to process the
c              current permutation, or execute equivalent
c              statements, and go to step 2.
c
   18 call job(n,k,a,kount)
c
      kount = kount + 1
      goto 6
      end
```

```
      REAL FUNCTION TRIMSD(X, N, K, IW, IFAULT)
C
C     ALGORITHM AS 180  APPL. STATIST. (1982) VOL.31, NO.2
C
C     The value returned is a linear estimate of the standard deviation
C     from a sample of N values of X symmetrically trimmed to retain K
C     values.
C
      INTEGER IW(N)
      REAL X(N), QTR, HALF, P74, P98, AMID, B, P, SUM, C(6)
      DATA C/1.06683, 1.72534, 0.403049, 1.32458, -3.91154, 17.5265/,
     *  QTR/0.25/, HALF/0.5/, P74/0.74/, P98/0.98/
C
      IFAULT = 1
      TRIMSD = -1.0
      IF (N .LT. 4) RETURN
      IFAULT = 2
      P = K / FLOAT(N)
      IF (P .LT. HALF .OR. P .GT. P98) RETURN
      IFAULT = 3
      IF (MOD(N-K, 2) .NE. 0) RETURN
      IFAULT = 0
C
      ILO = (N-K)/2 + 1
      IHI = N - ILO + 1
      AMID = N + 1
C
C     Get ranks of X in IW
C
      IW(1) = 1
      DO 10 I = 2, N
        IW(I) = 1
        DO 10 J = 1, I-1
          IF (X(I) .GE. X(J)) GO TO 9
          IW(J) = IW(J) + 1
          GO TO 10
    9     IW(I) = IW(I) + 1
   10 CONTINUE
C
      SUM = 0.0
      DO 12 I = 1, N
   12 IF (IW(I) .GE. ILO .AND. IW(I) .LE. IHI) SUM =
     *            SUM + (2.0 * IW(I) - AMID) * X(I)
C
C     Calculate the unbiassing factor
C
      P = P74 - P
      B = ((((C(6)*P + C(5))*P + C(4))*P + C(3))*P + C(2))*P + C(1)
C
      TRIMSD = EXP(B) * SUM / (QTR * (2*K*(2*K - 1)))
      RETURN
      END
```

```
      SUBROUTINE FORKAL(IP, IQ, IR, NP, IRD, IRZ, ID, IL, N, NRBAR,
     *   PHI, THETA, DELTA, W, Y, AMSE, A, P, V, RESID, E, XNEXT, XROW,
     *   RBAR, THETAB, STORE, IFAULT)
C
C     ALGORITHM AS 182  APPL. STATIST. (1982) VOL.31, NO.2
C
C     Finite sample prediction from ARIMA processes.
C
C     Auxiliary routines required: KARMA & STARMA from AS 154 and
C     routines called by them: INCLU2 from ASR 17 (a slight variant on
C     AS 75, and REGRES from AS 75.
C
      REAL PHI(IR), THETA(IR), DELTA(ID), W(N), Y(IL), AMSE(IL),
     *   A(IRD), P(IRZ), V(NP), RESID(N), E(IR), XNEXT(NP), XROW(NP),
     *   RBAR(NRBAR), THETAB(NP), STORE(IRD)
      REAL ZERO, ONE, TWO
      DATA ZERO/0.0/, ONE/1.0/, TWO/2.0/
C
C     Invoking this routine will calculate the finite sample predictions
C     and their conditional mean square errors for any ARIMA process.
C
C     Check for input faults.
C
      IFAULT = 0
      IF (IP .LT. 0) IFAULT = 1
      IF (IQ .LT. 0) IFAULT = IFAULT + 2
      IF (IP * IP + IQ * IQ .EQ. 0) IFAULT = 4
      K = IQ + 1
      IF (K .LT. IP) K = IP
      IF (IR .NE. K) IFAULT = 5
      IF (NP .NE. IR * (IR + 1) / 2) IFAULT = 6
      IF (NRBAR .NE. NP * (NP - 1) / 2) IFAULT = 7
      IF (ID .LT. 0) IFAULT = 8
      IF (IRD .NE. IR + ID) IFAULT = 9
      IF (IRZ .NE. IRD * (IRD + 1) / 2) IFAULT = 10
      IF (IL .LT. 1) IFAULT = 11
      IF (IFAULT .NE. 0) RETURN
C
C     Calculate initial conditions for Kalman filter
C
      A(1) = ZERO
      V(1) = ONE
      IF (NP .EQ. 1) GO TO 130
      DO 100 I = 2, NP
  100 V(I) = ZERO
      IF (IQ .EQ. 0) GO TO 130
      IQ1 = IQ + 1
      DO 110 I = 2, IQ1
  110 V(I) = THETA(I-1)
      DO 120 J = 1, IQ
        LL = J * (2*IR + 1 - J) / 2
        DO 120 I = J, IQ
          LLI = LL + I
          V(LLI) = THETA(I) * THETA(J)
  120 CONTINUE
C
C     Find initial likelihood conditions.
C     IFAULT not tested on exit from STARMA as all possible errors
C     have been checked above.
C
  130 IF (IR .EQ. 1) P(1) = ONE / (ONE - PHI(1) * PHI(1))
```

```
      IF (IR .NE. 1) CALL STARMA(IP, IQ, IR, NP, PHI, THETA, A, P, V,
     *  THETAB, XNEXT, XROW, RBAR, NRBAR, IFAULT)
C
C     Calculate data transformations
C
      NT = N - ID
      IF (ID .EQ. 0) GO TO 170
      DO 140 J = 1, ID
        NJ = N - J
        STORE(J) = W(NJ)
  140 CONTINUE
      DO 160 I = 1, NT
        AA = ZERO
        DO 150 K = 1, ID
          IDK = ID + I - K
          AA = AA - DELTA(K) * W(IDK)
  150   CONTINUE
        IID = I + ID
        W(I) = W(IID) + AA
  160 CONTINUE
C
C     Evaluate likelihood to obtain final KF conditions
C
  170 SUMLOG = ZERO
      SSQ = ZERO
      IUPD = 1
      DEL = - ONE
      NIT = 0
      CALL KARMA(IP, IQ, IR, NP, PHI, THETA, A, P, V, NT, W, RESID,
     *    SUMLOG, SSQ, IUPD, DEL, E, NIT)
C
C     Calculate M.L.E. of sigma squared
C
      SIGMA = ZERO
      DO 200 J = 1, NT
  200 SIGMA = SIGMA + RESID(J)**2
      SIGMA = SIGMA / NT
C
C     Reset the initial A and P when differencing occurs
C
      IF (ID .EQ. 0) GO TO 250
      DO 210 I = 1, NP
  210 XROW(I) = P(I)
      DO 220 I = 1, IRZ
  220 P(I) = ZERO
      IND = 0
      DO 230 J = 1, IR
        K = (J-1) * (ID + IR + 1) - (J-1) * J / 2
        DO 230 I = J, IR
          IND = IND + 1
          K = K + 1
          P(K) = XROW(IND)
  230 CONTINUE
      DO 240 J = 1, ID
        IRJ = IR + J
        A(IRJ) = STORE(J)
  240 CONTINUE
C
C     Set up constants
C
  250 IR2 = IR + 1
```

```
        IR1 = IR - 1
        ID1 = ID - 1
        ID2R = 2 * IRD
        ID2R1 = ID2R - 1
        IDD1 = 2 * ID + 1
        IDD2 = IDD1 + 1
        I45 = ID2R + 1
        IDRR1 = IRD + 1
        IDDR = 2 * ID + IR
        JKL = IR * (IDDR + 1) / 2
        JKL1 = JKL + 1
        ID2R2 = ID2R + 2
        IBC = IR * (I45 - IR) / 2
        DO 560 L = 1, IL
C
C     Predict A
C
          A1 = A(1)
          IF (IR .EQ. 1) GO TO 310
          DO 300 I = 1, IR1
  300     A(I) = A(I+1)
  310     A(IR) = ZERO
          IF (IP .EQ. 0) GO TO 330
          DO 320 J = 1, IP
  320     A(J) = A(J) + PHI(J) * A1
  330     IF (ID .EQ. 0) GO TO 360
          DO 340 J = 1, ID
            IRJ = IR + J
            A1 = A1 + DELTA(J) * A(IRJ)
  340     CONTINUE
          IF (ID .LT. 2) GO TO 360
          DO 350 I = 1, ID1
            IRI1 = IRD - I
            A(IRI1 + 1) = A(IRI1)
  350     CONTINUE
  360     A(IR2) = A1
C
C     Predict P
C
          IF (ID .EQ. 0) GO TO 480
          DO 370 I = 1, ID
            STORE(I) = ZERO
            DO 370 J = 1, ID
              LL = MAX(I,J)
              K = MIN(I,J)
              JJ = JKL + (LL - K) + 1 + (K-1) * (IDD2 - K) / 2
              STORE(I) = STORE(I) + DELTA(J) * P(JJ)
  370     CONTINUE
          IF (ID .EQ. 1) GO TO 400
          DO 380 J = 1, ID1
            JJ = ID - J
            LK = (JJ-1) * (IDD2 - JJ) / 2 + JKL
            LK1 = JJ * (IDD1 - JJ) / 2 + JKL
            DO 380 I = 1, J
              LK = LK + 1
              LK1 = LK1 + 1
              P(LK1) = P(LK)
  380     CONTINUE
          DO 390 J = 1, ID1
            JKLJ = JKL1 + J
            IRJ = IR + J
```

```
            P(JKLJ) = STORE(J) + P(IRJ)
      390   CONTINUE
      400   P(JKL1) = P(1)
            DO 410 I = 1, ID
              IRI = IR + I
              P(JKL1) = P(JKL1) + DELTA(I) * (STORE(I) + TWO * P(IRI))
      410   CONTINUE
            DO 420 I = 1, ID
              IRI = IR + I
              STORE(I) = P(IRI)
      420   CONTINUE
            DO 430 J = 1, IR
              KK1 = J * (ID2R1 - J) / 2 + IR
              K1 = (J-1) * (ID2R - J) / 2 + IR
              DO 430 I = 1, ID
                KK = KK1 + I
                K = K1 + I
                P(K) = PHI(J) * STORE(I)
                IF (J .NE. IR) P(K) = P(K) + P(KK)
      430   CONTINUE
      C
            DO 440 J = 1, IR
              STORE(J) = ZERO
              KKK = J * (I45 - J) / 2 - ID
              DO 440 I = 1, ID
                KKK = KKK + 1
                STORE(J) = STORE(J) + DELTA(I) * P(KKK)
      440   CONTINUE
            IF (ID .EQ. 1) GO TO 460
            DO 450 J = 1, IR
              K = J * IDRR1 - J * (J+1) / 2 + 1
              DO 450 I = 1, ID1
                K = K - 1
                P(K) = P(K-1)
      450   CONTINUE
      460   DO 470 J = 1, IR
              K = (J-1) * (ID2R - J) / 2 + IR + 1
              P(K) = STORE(J) + PHI(J) * P(1)
              IF (J .LT. IR) P(K) = P(K) + P(J+1)
      470   CONTINUE
      480   DO 490 I = 1, IR
      490   STORE(I) = P(I)
      C
            IND = 0
            DT = P(1)
            DO 500 J = 1, IR
              PHIJ = PHI(J)
              PHIJDT = PHIJ * DT
              IND2 = (J-1) * (ID2R2 - J) / 2
              IND1 = J * (I45 - J) / 2
              DO 500 I = J, IR
                IND = IND + 1
                IND2 = IND2 + 1
                PHII = PHI(I)
                P(IND2) = V(IND) + PHII * PHIJDT
                IF (J .LT. IR) P(IND2) = P(IND2) + STORE(J+1) * PHII
                IF (I .EQ. IR) GO TO 500
                IND1 = IND1 + 1
                P(IND2) = P(IND2) + STORE(I+1) * PHIJ + P(IND1)
      500   CONTINUE
      C
```

```
C     Predict Y
C
        Y(L) = A(1)
        IF (ID .EQ. 0) GO TO 520
        DO 510 J = 1, ID
          IRJ = IR + J
          Y(L) = Y(L) + A(IRJ) * DELTA(J)
  510   CONTINUE
C
C     Calculate M.S.E. of Y
C
  520   AMS = P(1)
        IF (ID .EQ. 0) GO TO 550
        DO 530 J = 1, ID
          JRJ = IBC + (J-1) * (IDD2 - J) / 2
          IRJ = IR + J
          AMS = AMS + TWO * DELTA(J) * P(IRJ) + P(JRJ+1) * DELTA(J)**2
  530   CONTINUE
        IF (ID .EQ. 1) GO TO 550
        DO 540 J = 1, ID1
          J1 = J + 1
          JRK = IBC + 1 + (J-1) * (IDD2 - J) / 2
          DO 540 I = J1, ID
            JRK = JRK + 1
            AMS = AMS + TWO * DELTA(I) * DELTA(J) * P(JRK)
  540   CONTINUE
  550   AMSE(L) = AMS * SIGMA
  560 CONTINUE
C
        RETURN
        END
```

```
      real function random()
c
c     Algorithm AS 183 Appl. Statist. (1982) vol.31, no.2
c
c     Returns a pseudo-random number rectangularly distributed
c     between 0 and 1.   The cycle length is 6.95E+12 (See page 123
c     of Applied Statistics (1984) vol.33), not as claimed in the
c     original article.
c
c     IX, IY and IZ should be set to integer values between 1 and
c     30000 before the first entry.
c
c     Integer arithmetic up to 30323 is required.
c
      integer ix, iy, iz
      common /randc/ ix, iy, iz
c
      ix = 171 * mod(ix, 177) - 2 * (ix / 177)
      iy = 172 * mod(iy, 176) - 35 * (iy / 176)
      iz = 170 * mod(iz, 178) - 63 * (iz / 178)
c
      if (ix .lt. 0) ix = ix + 30269
      if (iy .lt. 0) iy = iy + 30307
      if (iz .lt. 0) iz = iz + 30323
c
c     If integer arithmetic up to 5212632 is available, the preceding
c     6 statements may be replaced by:
c
c     ix = mod(171 * ix, 30269)
c     iy = mod(172 * iy, 30307)
c     iz = mod(170 * iz, 30323)
c
      random = mod(float(ix) / 30269. + float(iy) / 30307. +
     +                       float(iz) / 30323., 1.0)
      return
      end
c
c
c
c
        real function uniform()
c
c       Generate uniformly distributed random numbers using the 32-bit
c       generator from figure 3 of:
c       L'Ecuyer, P. Efficient and portable combined random number
c       generators, C.A.C.M., vol. 31, 742-749 & 774-?, June 1988.
c
c       The cycle length is claimed to be 2.30584E+18
c
c       Seeds can be set by calling the routine set_uniform
c
c       It is assumed that the Fortran compiler supports long variable
c       names, and integer*4.
c
        integer*4 z, k, s1, s2
        common /unif_seeds/ s1, s2
        save /unif_seeds/
c
        k = s1 / 53668
        s1 = 40014 * (s1 - k * 53668) - k * 12211
        if (s1 .lt. 0) s1 = s1 + 2147483563
```

```
c
      k = s2 / 52774
      s2 = 40692 * (s2 - k * 52774) - k * 3791
      if (s2 .lt. 0) s2 = s2 + 2147483399
c
      z = s1 - s2
      if (z .lt. 1) z = z + 2147483562
c
      uniform = z / 2147483563.
      return
      end


      subroutine set_uniform(seed1, seed2)
c
c     Set seeds for the uniform random number generator.
c
      integer*4 s1, s2, seed1, seed2
      common /unif_seeds/ s1, s2
      save /unif_seeds/

      s1 = seed1
      s2 = seed2
      return
      end
```

```
      SUBROUTINE PROB(EPSI, N, NU, C, M, VM, ANS, BOUND, RND, IFAULT)
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C        ALGORITHM AS 184  APPL. STATIST. (1982) VOL.31, NO.3
C
C        CALCULATES EXPRESSION (1) OF THE PURPOSE SECTION.
C        USEFUL FOR NON-CENTRAL STUDENTIZED MAXIMUM, MAXIMUM MODULUS,
C        AND MULTIPLE T-TEST PROBABILITIES.
C
      DIMENSION PROD(2), EK(3), PHI(2, 10), C(2, N), AC(2), AM(2), M(N),
     *  PH(200, 10), S(5), G2(3), G1(3), GMX(4), G2S(200, 3), AD(2, 10),
     *  ER(4), PHM(10), RS(10), VM(N), R(200), PR(200)
C
C        ALR2PI=ALOG(2.0*PI), AL2=ALOG(2.0),
C        ER CONTAINS ROUNDING ERROR BOUNDS
C
      DATA ALR2PI /0.91893853320467274178/,AL2 /0.69314718055994530942/,
     *  ER(1), ER(2), ER(3), ER(4) /5.0E-07, 5.0E-07, 5.0E-07, 5.0E-07/,
     *  ZERO, HALF, ONE, TWO, SEVEN, EIGHT /0.0, 0.5, 1.0, 2.0, 7.0,
     *  8.0/, EITH, SEVEIG, FOUR, TWEL, XNIN6 /0.125, 0.875, 4.0, 12.0,
     *  96.0/, MAXN /10/
C
C        FUNCTION STATEMENTS
C
      BDD1(X, Y, Z) = ABS((X - Z) / Y - X * Y)
      BDD2(W, X, Y, Z) = ABS((W - Y) * (W - Z) + (X * X) * ((-Z * W + Y)
     *  * W + W * W * X * X))
      GABS(X) = ABS(X)
      GEXP(X) = EXP(X)
      GMAX1(X, Y) = AMAX1(X, Y)
      TMAX1(X, Y, Z) = AMAX1(X, Y, Z)
      GMIN1(X, Y) = AMIN1(X, Y)
      GSQRT(X) = SQRT(X)
      GFLOAT(IJ) = FLOAT(IJ)
      GLOG(X) = ALOG(X)
C
C        CHECKING FOR FAULTY INPUT DATA
C
      IFAULT = 0
      IF (NU .LE. 0) IFAULT = 1
      IF (EPSI .LE. ER(3)) IFAULT = 2
      IF (N .LE. 0 .OR. N .GT. MAXN) IFAULT = 3
      IF (IFAULT .NE. 0) RETURN
      IZERO = 0
      DO 2 J = 1, N
      IF (C(1, J) .EQ. C(2, J) .AND. M(J) .EQ. 0) IZERO = 1
      IF (C(1, J) .GT. C(2, J) .AND. M(J) .EQ. 0) IFAULT = 100 + J
      IF (M(J) .EQ. 1 .AND. C(1, J) .NE. ZERO) IFAULT = 200 + J
      IF (M(J) .EQ. 2 .AND. C(2, J) .NE. ZERO) IFAULT = 300 + J
      IF (M(J) .EQ. 3 .AND. (C(1, J) .NE. ZERO .OR. C(2, J) .NE. ZERO))
     *  IFAULT = 400 + J
      IF (M(J) .LT. 0 .OR. M(J) .GT. 3) IFAULT = 500 + J
    2 CONTINUE
      IF (IFAULT .NE. 0) RETURN
C
C        IZERO=1 IF ONE OF THE RANGES (C1,C2) IS DEGENERATE
C
      ANS = ZERO
      BOUND = ZERO
```

```
      RND = ZERO
      IF (IZERO .EQ. 1) RETURN
C
C        INITIALIZING VALUES TO BE USED IN PROGRAM
C
      NK = N - 1
      ITCH = 2
      INDXL = 3
      DF = GFLOAT(NU)
      HD = HALF * DF
      COEFF = HD * GLOG(HD) - ALGAMA(HD) + AL2
      COFF = GEXP(COEFF)
      EG = ONE
      FE = ZERO
      K = 1
      TD = HALF / DF
      SDFT = GSQRT(EIGHT * DF - SEVEN) * TD
      S(1) = ZERO
      S(2) = GSQRT(ONE - TD - SDFT)
      S(3) = GSQRT((DF - ONE) / DF)
      S(4) = GSQRT(ONE - TD + SDFT)
      S(5) = GMAX1(TWO, GSQRT(ONE + GSQRT(GSQRT(XNIN6 * (FOUR + DF)
     *  / (EPSI * DF ** 3)))))
      EPSL = EPSI * SEVEIG / S(5)
      ES = S(5) - (S(4) - S(2)) * GMIN1(ONE, GFLOAT(NU / 3))
      EK(2) = ZERO
      EK(3) = ZERO
      PROD(1) = ONE
C
C        THE FIRST 4 LINES OF THE DO 6 LOOP DETERMINE WHERE THE FACTORS,
C        PHI( ,JK), OF PROD ARE MONOTONE.
C        THE LOOP COMPUTES PHI AND PROD FOR S=0.0
C
      DO 6 JK = 1, N
      RS(JK) = -ONE
      SUMC = C(1, JK) + C(2, JK)
      IF (C(1, JK) * C(2, JK) .GT. ZERO) RS(JK) = VM(JK) / SUMC +
     *  GSQRT(VM(JK) * VM(JK) + TWO * SUMC * GLOG(C(2, JK) / C(1, JK))
     *  / (C(2, JK) - C(1, JK))) / GABS(SUMC)
      IF (M(JK) .EQ. 0) PHI(1, JK) = ZERO
      IF (M(JK) .EQ. 1) PHI(1, JK) = ALNORM(-VM(JK), .FALSE.)
      IF (M(JK) .EQ. 2) PHI(1, JK) = ONE - ALNORM(-VM(JK), .FALSE.)
      IF (M(JK) .EQ. 3) PHI(1, JK) = ONE
      PROD(1) = PROD(1) * PHI(1, JK)
    6 CONTINUE
C
C        THESE 5 LINES AND THE FIRST 4 LINES OF THE DO 60 LOOP ARE
C        SPECIAL HANDLING FOR NU.LE.2
C
      IXG = MIN0(2, NU - 1)
      XH = GFLOAT((2 - IXG) / 2)
      G1(1) = XH * COFF
      G1(2) = GFLOAT(IXG - 2 * (IXG / 2)) * COFF
      G1(3) = BDD2(DF, ZERO, ONE, TWO) + GFLOAT(1 - (1 + IXG) / 2)
C
C        INTEGRATION BEGINS.  (FE,R(K)) IS CURRENT SUBINTERVAL
C
      DO 60 L = 2, 5
      IF ((NU .LE. 2 .AND. L .EQ. 2) .OR. (NU .EQ. 1 .AND. L .LE. 3))
     *  GOTO 60
      R(1) = S(L)
```

```
      ITCH = 2
      IF (NU .LE. 2 .OR. L .EQ. 2 .OR. L .EQ. 5) GOTO 13
      INDXL = 2
      ITCH = 1
      GOTO 13
C
C        SETTING UP THE MESH.   STATEMENTS 9-12 MAKE THE MESH FINER
C
    9 IF (K .GE. 200) GOTO 62
      R(K + 1) = (R(K) + FE) * HALF
      PR(K) = PROD(2)
      DO 11 J = 1, N
   11 PH(K, J) = PHI(2, J)
      DO 12 J = 1, INDXL
   12 G2S(K, J) = G2(J)
      K = K + 1
      ITCH = INDXL - 1
C
C        COMPUTING PHI, PROD, DENSITY OF S, AND ITS DERIVATIVE AT S=R(K)
C
   13 G2(1) = GEXP(COEFF + (DF - ONE) * GLOG(R(K)) - DF * R(K) * R(K)
     *  * HALF)
      G2(2) = G2(1) * BDD1(DF, R(K), ONE)
      PROD(2) = ONE
      DO 14 J = 1, N
      IF (M(J) .EQ. 0) PHI(2, J) = ALNORM(C(2, J) * R(K) - VM(J),
     *  .FALSE.) - ALNORM(C(1, J) * R(K) - VM(J), .FALSE.)
      IF (M(J) .EQ. 1) PHI(2, J) = ALNORM(C(2, J) * R(K) - VM(J),
     *  .FALSE.)
      IF (M(J) .EQ. 2) PHI(2, J) = ONE - ALNORM(C(1, J) * R(K) - VM(J),
     *  .FALSE.)
      IF (M(J) .EQ. 3) PHI(2, J) = ONE
      PROD(2) = PROD(2) * PHI(2, J)
   14 CONTINUE
C
C        STATEMENTS 18-22 DECIDE WHETHER TO USE FORM A (INDXL=3)
C        OR FORM B (INDXL=2).
C        ITCH DETERMINES WHETHER OR NOT G2(3) NEEDS TO BE COMPUTED
C
   18 EPS = R(K) * EPSL
      GMIN = GMIN1(G1(1), G2(1))
      IF (NU .LE. 2) GOTO 22
      IF (GMIN .LT. ONE) GOTO 20
      INDXL = 2
      GOTO 26
   20 IF (((INDXL .EQ. 3) .OR. (L .EQ. 3)) .OR. (L .EQ. 4)) GOTO 22
      G1(3) = BDD2(DF, FE, ONE, TWO) / (FE * FE)
      INDXL = 3
      ITCH = 2
   22 IF (ITCH .EQ. 2) G2(3) = BDD2(DF, R(K), ONE, TWO) / (R(K) * R(K))
C
C        THE DO 32 LOOP COMPUTES BOUNDS FOR FIRST (AD(1,J)) AND
C        SECOND (AD(2,J)) DERIVATIVES OF PHI( ,J)
C
   26 PRM = ONE
      DO 32 J = 1, N
      PHM(J) = GMAX1(PHI(1, J), PHI(2, J))
      IF (FE .LT. RS(J) .AND. R(K) .GT. RS(J)) PHM(J) = ALNORM(C(2, J)
     *  * RS(J) - VM(J), .FALSE.) - ALNORM(C(1, J) * RS(J) - VM(J),
     *  .FALSE.)
      PRM = PRM * PHM(J)
```

```
      DO 30 I = 1, 2
      AC1 = GABS(GABS(C(I, J) * FE) - GABS(VM(J)))
      AC2 = GABS(GABS(C(I, J) * R(K)) - GABS(VM(J)))
      AC(I) = GMIN1(AC1, AC2)
      IF (C(I, J) .EQ. ZERO) GOTO 28
      RATC = VM(J) / C(I, J)
      IF (FE .LT. RATC .AND. R(K) .GT. RATC) AC(I) = ZERO
   28 AM(I) = GMAX1(AC1, AC2)
      AC(I) = GABS(C(I, J)) * GEXP(-HALF * AC(I) * AC(I) - ALR2PI)
   30 CONTINUE
      AD(1, J) = AC(1) + AC(2)
      AD(2, J) = GABS(C(2, J)) * AM(2) * AC(2) + GABS(C(1, J)) * AM(1)
     *  * AC(1)
   32 CONTINUE
C
C        THE DO 33 AND DO 34 LOOPS BOUND FIRST (BOU1) AND
C        SECOND (BOU2) DERIVATIVES OF PROD
C
      BOU1 = ZERO
      BOU2 = ZERO
      DO 33 J = 1, N
      BOU2 = BOU2 + PRM / PHM(J) * AD(2, J)
   33 BOU1 = BOU1 + PRM / PHM(J) * AD(1, J)
      IF (N .EQ. 1) GOTO 36
      DO 34 J = 1, NK
      JJ = J + 1
      DO 34 KM = JJ, N
      BOU2 = BOU2 + TWO * PRM / (PHM(J) * PHM(KM)) * AD(1, J) *
     *  AD(1, KM)
   34 CONTINUE
   36 IF (INDXL .EQ. 2) GOTO 40
C
C        CHECKING THE BOUND FOR THE FORM A CASES
C
      DEL = R(K) - FE
      DO 38 J = 1, 3
   38 GMX(J) = GMAX1(G1(J), G2(J))
      GMX(4) = GMX(1) * GMX(3)
      IF (NU .EQ. 3 .AND. FE .EQ. ZERO) GMX(4) = GMAX1(GMX(4), TWO *
     *  COFF)
      CHECK = (DEL ** 3) * (PRM * GMX(4) + BOU2 * GMX(1) + BOU1 * TWO *
     *  GMX(2)) / TWEL
      GA = G1(1)
      GB = G2(1)
      GOTO 43
C
C        CHECKING THE BOUND FOR THE FORM B CASES
C
   40 DEL = GAMAIN(DF, R(K) * R(K) * DF) - GAMAIN(DF, FE * FE * DF)
      BOU3 = GMAX1(BDD1(DF, FE, ONE), BDD1(DF, R(K), ONE))
      CHECK = (DEL ** 3) * (BOU2 + BOU1 * BOU3) / (GMIN * GMIN * TWEL)
      GA = ONE
      GB = ONE
   43 BOU4 = BOUND + CHECK
      IF (BOU4 .GT. EPS) GOTO 9
      ANS = ANS + HALF * (PROD(1) * GA + PROD(2) * GB) * DEL
      EK(INDXL) = EK(INDXL) + ONE
C
C        ADVANCING THE INTEGRATION IF THE BOUND IS SMALL ENOUGH
C
      BOUND = BOU4
```

```
      IF (INDXL .EQ. 3) EG = TMAX1(EG, G1(1), G2(1))
      FE = R(K)
      PROD(1) = PROD(2)
      DO 46 J = 1, INDXL
   46 G1(J) = G2(J)
      DO 50 J = 1, N
   50 PHI(1, J) = PHI(2, J)
      IF (K .LE. 1) GOTO 60
      K = K - 1
      ITCH = 1
      DO 54 J = 1, N
   54 PHI(2, J) = PH(K, J)
      PROD(2) = PR(K)
      DO 56 J = 1, INDXL
   56 G2(J) = G2S(K, J)
      GOTO 18
   60 CONTINUE
      BOUND = BOUND + EPSI * EITH
C
C        COMPUTING THE BOUND ON ROUNDING ERROR
C
      AN = GFLOAT(N)
      EA = GABS(GEXP(ER(4)) - ONE)
      RND = (EG * TWO * AN * ER(1) + EG * EA / (ONE + EA) + ER(3))
     *  * (ES + EK(3) * ER(3)) + ER(3) * EK(3) * EG + TWO * AN * ER(1)
     *  + (ONE + TWO * AN * ER(1)) * EK(2) * ER(2)
      RETURN
   62 IFAULT = 4
      RETURN
      END
```

```
      SUBROUTINE ASRCH(NVAR, DIM, TABLE, NTAB, FIT, TFIT, INIT, CONFIG,
     *   NCON, MCON, CONFG1, CONFG2, CONFG3, MARG, NMAR, U, NU, MAXDEV,
     *   MAXIT, DEV, L1, L2, IA, MA, IB, MB, NI, NA, IFAULT)
C
C     ALGORITHM AS 185  APPL. STATIST. (1982) VOL.31, NO.3
C
C     This algorithm searches for a best log-linear model in contingency
C     tables using backward elimination procedure.
C
C     This version permits up to 7 variables.   If this limit is too
C     small, change the value of MAXVAR to its new limit, and the value
C     of the dimension of LOCMAR, MASK, & DELEF to the values of MCON,
C     MAXVAR + 1, and MAXVAR respectively.   See also changes needed in
C     subroutines LOGLIN (AS 51, Haberman 1972), LOWEFF, and DEGREF.
C
C     AS R87 has been incorporated from AS vol.40(2)
C
C     Auxiliary routines required: AS 51 and AS 66
C
      INTEGER DIM(NVAR), LOCMAR(35), CONFIG(NVAR, MCON), CONFG1(NVAR,
     *  MCON), CONFG2(NVAR, MCON), CONFG3(NVAR, MCON), DELEF(7),
     *  MASK(8), IB(NVAR, MB), SIZE
      REAL TABLE(NTAB), FIT(NTAB), TFIT(NTAB), MARG(NMAR), U(NU),
     *  MAXDEV, DEV(MAXIT), IA(4, MA), L1, L2, LR, LR1, LR3, ZERO
      DATA MAXVAR /7/, ZERO /0.0/
C
      IFAULT = 0
      NA = 0
      NB = 0
      IF (NVAR .GT. 0 .AND. NVAR .LE. MAXVAR) GO TO 10
    5 IFAULT = 4
      RETURN
C
   10 SIZE = 1
      DO 12 I = 1, NVAR
        IF (DIM(I) .LE. 0) GO TO 5
        SIZE = SIZE * DIM(I)
   12 CONTINUE
      IF (SIZE .NE. NTAB) GO TO 5
      DO 15 I = 1, NTAB
        IF (TABLE(I) .LT. ZERO .OR. FIT(I) .LT. ZERO) GO TO 5
        TFIT(I) = FIT(I)
   15 CONTINUE
C
C     Select a best initial fit.   Skip step if given initial fit by user.
C
      DO 70 NWAY = 1, NVAR
        IF (INIT .EQ. 1) GO TO 50
C
C     Construct CONFIG for model with full I-order effects
C
        NCON = 1
        DO 20 I = 1, NWAY
   20   NCON = NCON * (NVAR - I + 1) / I
        IF (MCON .LT. NCON) GO TO 400
        DO 25 J = 1, NVAR
          MASK(J) = J - 1
          DO 25 I = 1, NCON
            CONFIG(J,I) = 0
   25   CONTINUE
        MASK(NWAY+1) = NVAR
```

```
      DO 45 L = 1, NCON
        DO 35 J = 1, NWAY
35      CONFIG(J,L) = MASK(J) + 1
        DO 40 J = 1, NWAY
          MASK(J) = MASK(J) + 1
          IF (MASK(J) .LT. MASK(J+1)) GO TO 45
          MASK(J) = J - 1
40      CONTINUE
45    CONTINUE
C
C     LOGLIN performs an iterative proportional fit of the marginal
C     totals (described in CONFIG) for a contingency table.
C
50    CALL LOGLIN(NVAR, DIM, NCON, CONFIG, NTAB, TABLE, FIT, LOCMAR,
     *     NMAR, MARG, NU, U, MAXDEV, MAXIT, DEV, NLAST, IFAULT)
      IF (IFAULT .GT. 0) RETURN
      NA = NA + 1
      NI = NA
      CALL LRCHSQ(TABLE, FIT, NTAB, LR)
      CALL DEGREF(NVAR, DIM, NCON, NTAB, CONFIG, DF)
      P = XTAIL(LR, DF)
      CALL CPYLR(LR, DF, P, IA(1,NA), IA(2,NA), IA(3,NA))
      IF (NA .GT. MA .OR. (NB + NCON) .GT. MB) GO TO 400
      IA(4,NA) = NCON
      DO 55 K = 1, NCON
        NB = NB + 1
        DO 55 I = 1, NVAR
          IB(I,NB) = CONFIG(I,K)
55    CONTINUE
      IF (INIT .EQ. 1) GO TO 90
      IF (NWAY .EQ. NVAR) RETURN
      IF (P .LE. L1) GO TO 70
      IF (NA .EQ. 1) GO TO 65
      LA = NA - 1
      IF (IA(3,LA) .LE. L1) GO TO 60
C
C     If the previous model is a good fit and the current fit does not
C     improve significantly over it, use the previous fit as the best
C     initial fit.
C
        DLR = IA(1,LA) - IA(1,NA)
        DDF = IA(2,LA) - IA(2,NA)
        DP = XTAIL(DLR,DDF)
        IF (DP .LE. L2) GO TO 60
        CALL CPYCON(CONFG1, CONFIG, NVAR, NCON1, NCON)
        CALL CPYLR(IA(1,LA), IA(2,LA), IA(3,LA), LR, DF, P)
        GO TO 90
60    IF (NWAY .EQ. NVAR - 1) GO TO 90
65    CALL CPYCON(CONFIG, CONFG1, NVAR, NCON, NCON1)
70  CONTINUE
C
C     Backward elimination of effects from the best initial fit.
C
90  PP = ZERO
    N = 0
    NA = NA + 1
    IF (NA .GT. MA .OR. NB + NCON .GT. MB) GO TO 400
    CALL CPYLR(LR, DF, P, IA(1,NA), IA(2,NA), IA(3,NA))
    IA(4,NA) = NCON
    DO 100 K = 1, NCON
      NB = NB + 1
```

```
            DO 100 I = 1, NVAR
              IB(I,NB) = CONFIG(I,K)
      100 CONTINUE
          LCON = NCON - 1
          DO 300 J = 1, NCON
    C
    C     Re-initialize FIT when fitting a simpler model
    C
            DO 150 I = 1, NTAB
      150   FIT(I) = TFIT(I)
            CALL CPYCON(CONFIG, CONFG1, NVAR, NCON, NCON1)
    C
    C     Store effect to be deleted in DELEF and compress CONFG1.
    C     Obtain lower order effects of DELEF not implied in CONFG1 and
    C     store them in CONFG2.
    C
            NCON2 = NCON1 - J + 1
            DO 200 I = 1, NVAR
              DELEF(I) = CONFG1(I,NCON2)
              CONFG1(I,NCON2) = CONFG1(I,NCON1)
      200   CONTINUE
            CALL LOWEFF(NVAR, LCON, MCON, CONFG1, CONFG2, DELEF, JJ)
            NCON1 = LCON + JJ
            IF (JJ .EQ. 0) GO TO 225
            DO 220 L = 1, JJ
              LL = LCON + L
              DO 220 I = 1, NVAR
                CONFG1(I,LL) = CONFG2(I,L)
      220   CONTINUE
      225   NA = NA + 1
            IF (NA .GT. MA .OR. NB + NCON1 .GT. MB) GO TO 400
            CALL LOGLIN(NVAR, DIM, NCON1, CONFG1, NTAB, TABLE, FIT, LOCMAR,
         *      NMAR, MARG, NU, U, MAXDEV, MAXIT, DEV, NLAST, IFAULT)
            IF (IFAULT .GT. 0) RETURN
            CALL LRCHSQ(TABLE, FIT, NTAB, LR1)
            CALL DEGREF(NVAR, DIM, NCON1, NTAB, CONFG1, DF1)
            P1 = XTAIL(LR1, DF1)
            CALL CPYLR(LR1, DF1, P1, IA(1,NA), IA(2,NA), IA(3,NA))
            IA(4,NA) = NCON1
            DO 250 K = 1, NCON1
              NB = NB + 1
              DO 250 I = 1, NVAR
                IB(I,NB) = CONFG1(I,K)
      250   CONTINUE
            DLR = LR1 - LR
            DDF = DF1 - DF
            DP = XTAIL(DLR, DDF)
    C
    C     Select the best fitted model that does not result in a significant
    C     increase in LR chi-squared value for further simplification.
    C
            IF (P1 .LT. L1 .OR. DP .LT. L2) GO TO 300
            IF (P1 .LT. PP) GO TO 300
            PP = P1
            N = N + 1
            CALL CPYCON(CONFG1, CONFG3, NVAR, NCON1, NCON3)
            CALL CPYLR(LR1, DF1, P1, LR3, DF3, P3)
      300 CONTINUE
          IF (N .EQ. 0) GO TO 500
    C
    C     Copy the best simplified fit of each iteration to CONFIG.
```

```
C
      CALL CPYCON(CONFG3, CONFIG, NVAR, NCON3, NCON)
      CALL CPYLR(LR3, DF3, P3, LR, DF, P)
      IF (NCON .EQ. 1 .AND. CONFIG(2,1) .EQ. 0) GO TO 500
      GO TO 90
C
  400 IFAULT = 5
      RETURN
C
C     Obtain fitted table of the best final / intermediate fit.
C
  500 CALL LOGLIN(NVAR, DIM, NCON, CONFIG, NTAB, TABLE, FIT, LOCMAR,
     *   NMAR, MARG, NU, U, MAXDEV, MAXIT, DEV, NLAST, IFAULT)
      RETURN
      END
C
      SUBROUTINE LOWEFF(NVAR, NCON, MCON, CONFG1, CONFG2, DELEF, JJ)
C
C     ALGORITHM AS 185.1  APPL. STATIST. (1982) VOL.31, NO.3
C
C     Find lower order effects of DELEF not implied in CONFG1 and sore
C     them in CONFG2.  If NVAR is to be greater than 7, the dimensions
C     of MASK, MASK1 and MASK2 must all be increased to the value NVAR.
C
      INTEGER CONFG1(NVAR,NCON), CONFG2(NVAR,MCON), DELEF(NVAR),
     *   MASK(7), MASK1(7), MASK2(7)
C
C     Determine the order of the deleted effect.
C
      NWAY = 0
      DO 100 I = 1, NVAR
  100 IF (DELEF(I) .NE. 0) NWAY = NWAY + 1
      JJ = 0
      IF (NWAY .EQ. 1) RETURN
C
C     Find all possible lower order effects of the deleted effect.
C
      DO 300 J = 1, NWAY
        MASK(J) = J - 1
        DO 300 I = 1, NVAR
          CONFG2(I,J) = 0
  300 CONTINUE
      N = NWAY - 1
      MASK(N+1) = NWAY
  350 CONTINUE
      DO 360 I = 1, NVAR
  360 MASK1(I) = 0
      JJ = JJ + 1
      DO 400 J = 1, N
        L = MASK(J) + 1
        K = DELEF(L)
        MASK1(K) = 1
        CONFG2(J,JJ) = K
  400 CONTINUE
      IF (NCON .EQ. 0) GO TO 810
      NN = 0
C
C     Check if each lower order effect is implied in CONFG1.
C
      DO 700 J = 1, NCON
        DO 450 I = 1, NVAR
```

```
  450   MASK2(I) = 0
        DO 500 I = 1, NVAR
          K = CONFG1(I,J)
          IF (K .NE. 0) MASK2(K) = 1
  500   CONTINUE
        DO 550 K = 1, NVAR
          IF (MASK1(K) .EQ. 1 .AND. MASK2(K) .NE. 1) GO TO 650
  550   CONTINUE
        GO TO 800
  650   NN = NN + 1
  700 CONTINUE
      IF (NN .EQ. NCON) GO TO 810
  800 JJ = JJ - 1
  810 DO 850 J = 1, N
        MASK(J) = MASK(J) + 1
        IF (MASK(J) .LT. MASK(J+1)) GO TO 350
        MASK(J) = J - 1
  850 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE LRCHSQ(TABLE, FIT, NTAB, RATIO)
C
C     ALGORITHM AS 185.2  APPL. STATIST. (1982) VOL.31, NO.3
C
C     LRCHSQ calculates twice the likelihood ratio.
C
      REAL TABLE(NTAB), FIT(NTAB), ZERO
      DATA ZERO /0.0/
C
      SUM = ZERO
      DO 10 I = 1, NTAB
C
C     Note that FIT(I) = 0 implies TABLE(I) = 0.
C
   10 IF (TABLE(I) .NE. ZERO) SUM = SUM + TABLE(I) * LOG(TABLE(I) /
     *    FIT(I))
      RATIO = SUM + SUM
      RETURN
      END
C
      SUBROUTINE DEGREF(NVAR, DIM, NCON, NTAB, CONFIG, DF)
C
C     ALGORITHM AS 185.3  APPL. STATIST. (1982) VOL.31, NO.3
C
C     DEGREF computes the degrees of freedom of a model described in
C     CONFIG.   If NVAR is to be greater than 7, the dimensions of MASK
C     and HASH must be increased to the values (NVAR + 1) and
C     (2.NVAR - 1) respectively).
C
      INTEGER DIM(NVAR), CONFIG(NVAR,NCON), MASK(8), HASH(127)
C
      NP = 0
      N = 2 ** NVAR - 1
      DO 5 I = 1, N
    5 HASH(I) = 0
      DO 100 I = 1, NCON
C
C     Find the order of each effect in CONFIG.
C
```

```
            DO 10 J = 1, NVAR
            IF (CONFIG(J,I) .EQ. 0) GO TO 20
      10    CONTINUE
            J = NVAR + 1
      20    N = J - 1
C
C     Set up MASK.
C
            DO 30 J = 1, N
      30    MASK(J) = 0
C
C     Use MASK to find all possible lower order effects.
C
      40    DO 50 J = 1, N
              MASK(J) = MASK(J) + 1
              IF (MASK(J) .EQ. 1) GO TO 60
              MASK(J) = 0
      50    CONTINUE
            GO TO 100
      60    K = 0
            L = 1
            DO 80 J = 1, N
              IF (MASK(J) .NE. 1) GO TO 80
C
C     Get the lower order effect degrees of freedom.
C
              IJ = CONFIG(J,I)
              L = L * (DIM(IJ) - 1)
              K = K + 2 ** (CONFIG(J,I) - 1)
      80    CONTINUE
C
C     Check if the lower order effect already existed.
C
            IF (HASH(K) .EQ. 1) GO TO 40
            HASH(K) = 1
            NP = NP + L
            GO TO 40
     100  CONTINUE
          DF = NTAB - NP - 1
C
          RETURN
          END
C
          REAL FUNCTION XTAIL(RATIO, DF)
C
C     ALGORITHM AS 185.4  APPL. STATIST. (1982) VOL.31, NO.3
C
C     Computes probability of a chi-squared variable > RATIO using an
C     approximation from JASA (1968) vol.68, p.1420.
C
          REAL RATIO, DF, ZERO, ONE, TWO, SIX, PT04
          DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, SIX /6.0/, PT04 /0.04/
C
          IF (RATIO .EQ. ZERO) GO TO 50
          IF (DF .EQ. ONE) GO TO 100
          S = (DF - ONE) / TWO
          T = RATIO / TWO
          D = T - S - ONE / SIX - PT04 / DF
          Z = (D * SQRT(TWO * S * LOG(S / T) + TWO * (T - S))) / ABS(S - T)
C
C     ALNORM (AS 66, Hill 1973) evaluates the tail prob. of Z = N(0,1)
```

```
C
      XTAIL = ALNORM(Z, .TRUE.)
      RETURN
C
   50 XTAIL = ONE
      RETURN
  100 Z = SQRT(RATIO)
      XTAIL = TWO * ALNORM(Z, .TRUE.)
      RETURN
      END
C
      SUBROUTINE CPYCON(CONFG1, CONFG2, NVAR, NCON1, NCON2)
C
C     ALGORITHM AS 185.5  APPL. STATIST. (1982) VOL.31, NO.3
C
      INTEGER CONFG1(NVAR,NCON1), CONFG2(NVAR,NCON1)
C
      DO 10 J = 1, NCON1
        DO 10 I = 1, NVAR
          CONFG2(I,J) = CONFG1(I,J)
   10 CONTINUE
      NCON2 = NCON1
      RETURN
      END
C
      SUBROUTINE CPYLR(LR1, DF1, P1, LR2, DF2, P2)
C
C     ALGORITHM AS 185.6  APPL. STATIST. (1982) VOL.31, NO.3
C
      REAL LR1, DF1, P1, LR2, DF2, P2
C
      LR2 = LR1
      DF2 = DF1
      P2 = P1
      RETURN
      END
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE RBO(N, A, M, IFAULT)
      COMPLEX A(N), T
C
C<<<<<  Acquired in machine-readable form from 'Applied Statistics'
C<<<<<  algorithms editor, January 1983.
C
C
C       ALGORITHM AS 186  APPL. STATIST. (1982) VOL.31, NO.3
C
C       SUBROUTINE RBO (WITH AN AUXILIARY SUBROUTINE FFT)
C       PERFORMS THE N-POINT DISCRETE FAST FOURIER TRANSFORM OF
C       THE SAMPLED FUNCTION STORED IN A(N), WHERE N=2**M,
C       M AN INTEGER
C
C       CHECK IF VALUES OF N AND M ARE PROPERLY CHOSEN
C
      IFAULT = 1
      IF (N .LE. 0 .OR. 2 ** M .NE. N) RETURN
      IFAULT = 0
C
C       PART 1: PROCEDURE OF REVERSE BINARY ORDER PERMUTATION.
C       CONTENTS OF CELLS ARE INTERCHANGED WHEN J.GT.I
C
      NW = N / 2
      NV = N + 1
      IMAX = NW - 1
      J = 1
      DO 6 I = 1, IMAX, 2
C
C       IN THIS LOOP THE MINIMAL SET Z10 OF PAIRS OF ADDRESSES (I,J)
C       IS GENERATED. THE Z10 GENERATION INSTEAD OF ALL POSSIBLE
C       PAIRS (I,J) GENERATION REDUCES TOTAL TIME OF RBO PERMUTATION.
C
        IF (J - I) 1, 3, 2
C
C       THE PAIR OF ADDRESSES (I,J) AVAILABLE AT THIS POINT IS USED
C       (DUE TO SYMMETRY PRINCIPLE (1)) FOR (IP,JP) ADDRESS
C       GENERATION (IP.GT.N/2)
C
    1   IP = NV - I
        JP = NV - J
C
C       CONTENTS OF IP AND JP CELLS ARE INTERCHANGED
C
        T = A(JP)
        A(JP) = A(IP)
        A(IP) = T
C
C       THE NEXT PAIR OF ADDRESSES IS GENERATED USING EVEN ELEMENT
C       PROPERTY AND APPROPRIATE PERMUTATION IS PERFORMED.
C
        GOTO 3
    2   T = A(J)
        A(J) = A(I)
        A(I) = T
    3   IP = I + 1
        JP = J + NW
        T = A(JP)
        A(JP) = A(IP)
```

```
        A(IP) = T
C
C        THE NEXT PAIR OF ADDRESSES (I,J) IS GENERATED. THIS PART OF
C        THE PROCEDURE DOES NOT DIFFER FROM THE APPROPRIATE PART OF
C        THE RADER ALGORITHM IN RABINER AND GOLD (1975).
C
        K = NV
    4   IF (K .GE. J) GOTO 5
        J = J - K
        K = K / 2
        GOTO 4
    5   J = J + K
    6 CONTINUE
C
C        PART 2: NOW THE FFT SUBROUTINE IS CALLED. ONE CAN USE ANY OF
C        THE KNOWN PROCEDURES TO PERFORM THE COOLEY-TUKEY ALGORITHM.
C
      CALL FFT(M, N, A)
      RETURN
      END
C
      SUBROUTINE FFT(M, N, A)
C
C        ALGORITHM AS 186.1  APPL. STATIST. (1982) VOL.31, NO.3
C
C        SUBROUTINE FFT USES (WITH A SLIGHT MODIFICATION) PART OF FFT
C        ALGORITHM PUBLISHED IN RABINER AND GOLD (1975).
C
      COMPLEX A(N), U, W, T
      PI = 3.141592653589793
      LE = 1
      DO 20 L = 1, M
        LE1 = LE
        LE = LE + LE
        U = (1.0, 1.0)
        W = CMPLX(COS(PI / FLOAT(LE1)), -SIN(PI / FLOAT(LE1)))
        DO 20 J = 1, LE1
          DO 10 I = J, N, LE
            IP = I + LE1
            T = A(IP) * U
            A(IP) = A(I) - T
            A(I) = A(I) + T
   10     CONTINUE
          U = U * W
   20 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DIGAMI(D, X, P, GPLOG, GP1LOG, PSIP, PSIP1, PSIDP,
     *     PSIDP1, IFAULT)
C
C     ALGORITHM AS 187  APPL. STATIST. (1982) VOL.31, NO.3
C
C     Computes derivatives of the incomplete gamma integral for positive
C     parameters, X, P, using a series expansion if P > X or X <= 1, and
C     a continued fraction expansion otherwise.
C
C     Calculation of D(4) in line 60 corrected 5 October 1993.
C
C     N.B. The user must input values of the incomplete gamma, digamma
C          and trigamma functions.  These can be obtained using AS 239
C          (or 32), AS 103 and AS 121 respectively.
C
      REAL PN(6), D(6), DP(6), DPP(6), ZERO, ONE, TWO
      DATA E, OFLO, TMAX, VSMALL/1.E-6, 1.E30, 100.0, 1.E-30/
      DATA ZERO/0.0/, ONE/1.0/, TWO/2.0/
C
      IFAULT = 0
C
C     Derivatives with respect to X
C
      PM1 = P - ONE
      XLOG = LOG(X)
      D(1) = EXP(-GPLOG + PM1*XLOG - X)
      D(2) = D(1) * (PM1/X - ONE)
      D(5) = D(1) * (XLOG - PSIP)
C
C     Derivatives with respect to P
C
      IF (X .GT. ONE .AND. X .GE. P) GO TO 30
C
C     Series expansion
C
      F = EXP(P*XLOG - GP1LOG - X)
      DFP = F * (XLOG - PSIP1)
      DFPP = DFP*DFP/F - F*PSIDP1
C
      TMAXP = TMAX + P
      C = ONE
      S = ONE
      CP = ZERO
      CPP = ZERO
      DSP = ZERO
      DSPP = ZERO
      A = P
    1 A = A + ONE
      CPC = CP / C
      CP = CPC - ONE/A
      CPP = CPP/C - CPC*CPC + ONE/A**2
      C = C*X/A
      CP = CP*C
      CPP = CPP*C + CP*CP/C
      S = S + C
      DSP = DSP + CP
      DSPP = DSPP + CPP
      IF (A .GT. TMAXP) GO TO 1001
      IF (C .GT. E*S) GO TO 1
      D(6) = S*F
      D(3) = S*DFP + F*DSP
```

```
      D(4) = S*DFPP + TWO*DFP*DSP + F*DSPP
      RETURN
C
C     Continued fraction expansion
C
   30 F = EXP(P*XLOG - GPLOG - X)
      DFP = F * (XLOG - PSIP)
      DFPP = DFP*DFP/F - F*PSIDP
C
      A = PM1
      B = X + ONE - A
      TERM = ZERO
      PN(1) = ONE
      PN(2) = X
      PN(3) = X + ONE
      PN(4) = X * B
      S0 = PN(3) / PN(4)
      DO 31 I = 1, 4
        DP(I) = ZERO
        DPP(I) = ZERO
   31 CONTINUE
      DP(4) = -X
C
   32 A = A - ONE
      B = B + TWO
      TERM = TERM + ONE
      AN = A*TERM
      PN(5) = B*PN(3) + AN*PN(1)
      PN(6) = B*PN(4) + AN*PN(2)
      DP(5) = B*DP(3) - PN(3) + AN*DP(1) + PN(1)*TERM
      DP(6) = B*DP(4) - PN(4) + AN*DP(2) + PN(2)*TERM
      DPP(5) = B*DPP(3) + AN*DPP(1) + TWO*(TERM*DP(1) - DP(3))
      DPP(6) = B*DPP(4) + AN*DPP(2) + TWO*(TERM*DP(2) - DP(4))
C
      IF (ABS(PN(6)) .LT. VSMALL) GO TO 35
      S = PN(5) / PN(6)
      C = ABS(S - S0)
      IF (C*P .GT. E) GO TO 34
      IF (C .LE. E*S) GO TO 42
C
   34 S0 = S
   35 DO 36 I = 1, 4
        I2 = I + 2
        DP(I) = DP(I2)
        DPP(I) = DPP(I2)
        PN(I) = PN(I2)
   36 CONTINUE
C
      IF (TERM .GT. TMAX) GO TO 1001
      IF (ABS(PN(5)) .LT. OFLO) GO TO 32
      DO 41 I = 1, 4
        DP(I) = DP(I) / OFLO
        DPP(I) = DPP(I) / OFLO
        PN(I) = PN(I) / OFLO
   41 CONTINUE
      GO TO 32
C
   42 D(6) = ONE - F*S
      DSP = (DP(5) - S*DP(6)) / PN(6)
      DSPP = (DPP(5) - S*DPP(6) - TWO*DSP*DP(6)) / PN(6)
      D(3) = -F*DSP - S*DFP
```

```
      D(4) = -F*DSPP - TWO*DSP*DFP - S*DFPP
      RETURN
C
C     Set fault indicator
C
 1001 IFAULT = 1
      RETURN
      END
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE EODSEQ(KA, IKA, LENGTH, NSTATE, MAXDEP, MAXINT, REP,
     * PERC, AU, CU, CU0, NPOINT, IFAULT)
C
C     ALGORITHM AS 188  APPL. STATIST. (1983) VOL.32, NO.2 (pp185-196)
C
C     Estimation of the order of dependence in sequences
C
      INTEGER MAXKA, MAXJF
      PARAMETER (MAXKA=10, MAXJF=20)
      INTEGER IKA, LENGTH, NSTATE, MAXDEP, MAXINT, NPOINT, IFAULT
      INTEGER KA(IKA,LENGTH)
      REAL AU(MAXDEP), CU(MAXDEP), CU0, PERC(MAXDEP)
      LOGICAL REP
C
C     Local variables
C
      INTEGER NWORD, JFIX, NGP, KTEMP, KK
      INTEGER JRANGE, MURANG, IGF, NEOS, LAST, I, NGPSIZ
      INTEGER K, LIMIT, IPASSI, NCOUNT, JFIX2, JCTEMP, IW
      REAL SUM, ALN2, ACCX
C
C..NC AND JFROG ARE ARRAYS USED FOR TEMPORARY STORAGE
C..NC(MAXKA) NEEDS TO BE CHANGED TO NC(IKA) IF IKA .GT. MAXKA
C..JFROG(MAXJF) NEEDS TO BE CHANGED TO JFROG(MAXDEP)
C..               IF MAXDEP .GT. MAXJF
C
      INTEGER NC(MAXKA), JFROG(MAXJF)
C
C..NEOS (END OF SECTION) IS A NEGATIVE INTEGER USED TO
C..SIGNIFY A BREAK IN THE INPUT SEQUENCE
C
      NEOS = -1
C
C..INITIALIZATION PROCEDURE
C..CHECK INPUT PARAMETERS AND INITIALISE LOCAL VARIABLES
C..USED IN PACKING THE CODED OBSERVATIONS
C..INTWRD = NO. OF OBSERVATIONS STORABLE IN AN INTEGER
C..NWORD =  NO. OF INTEGERS REQUIRED TO STORE LARGEST GROUP
C
      IF (IKA.LE.0 .OR. LENGTH.LE.0) GO TO 300
      IF (IKA.GT.MAXKA .OR. MAXDEP.GT.MAXJF) GO TO 300
      IF (NSTATE.LE.0 .OR. MAXDEP.LE.0 .OR. MAXINT.LE.0) GO TO 300
      ALN2 = ALOG(2.0)
      JRANGE = 0
      MURANG = 1
   10 JRANGE = JRANGE + 1
      IF (JRANGE.GT.MAXINT) GO TO 300
      MURANG = MURANG*2
      IF (NSTATE.GT.MURANG) GO TO 10
      INTWRD = MAXINT/JRANGE
      NWORD = MAXDEP/INTWRD
      JFIX = MURANG**(INTWRD-1)
      JFIX2 = JFIX
      IW3 = MAXDEP - NWORD*INTWRD
      IF (IW3.EQ.0) GO TO 20
      NWORD = NWORD + 1
      JFIX2 = MURANG**(IW3-1)
      GO TO 30
   20 IW3 = INTWRD
```

```
C
C..SECTION FOR CODING DATA
C
C..CODE UP FIRST GROUP
C
   30 IF (NWORD.GT.IKA) GO TO 300
      NGPS = 0
      NPOINT = 0
   40 JCTEMP = 0
      IW = 0
      NW = 1
      LAST = -1
C
C..CONSTRUCT FIRST MAXDEP-TUPLE IN ARRAY NC
C
      DO 60 I=1,MAXDEP
   50   NPOINT = NPOINT + 1
        IF (NPOINT.GT.LENGTH) GO TO 320
        INNON = KA(1,NPOINT)
        IF (INNON.EQ.NEOS) GO TO 320
        IF (INNON.LT.0 .OR. INNON.GT.NSTATE-1) GO TO 310
        IF (.NOT.REP .AND. (INNON.EQ.LAST)) GO TO 50
C
C..CODE MAXDEP-TUPLES INTO SUCCESSIVE COLUMNS OF THE ARRAY KA
C
        JCTEMP = JCTEMP/MURANG + INNON*JFIX
        LAST = INNON
        IW = IW + 1
        IF (IW.LT.INTWRD) GO TO 60
C
C..NEED ANOTHER INTEGER TO STORE PACKED CODES
C
        IW = 0
        NC(NW) = JCTEMP
        JCTEMP = 0
        NW = NW + 1
   60 CONTINUE
      IF (IW.NE.0) NC(NW) = JCTEMP/(MURANG**(INTWRD - IW))
C
C..END OF CONSTRUCTION OF FIRST MAXDEP-TUPLE
C..STORE IT IN THE FIRST COLUMN OF KA
C
   70 NGPS = NGPS + 1
      DO 80 I=1,NWORD
        KA(I,NGPS) = NC(I)
   80 CONTINUE
C
C..CONSTRUCT NEW MAXDEP-TUPLE IN C
C
   90 NPOINT = NPOINT + 1
      IF (NPOINT.GT.LENGTH) GO TO 120
      INNON = KA(1,NPOINT)
C
C..NEW SEQUENCE TEST
C
      IF (INNON.EQ.NEOS) GO TO 40
      IF (INNON.LT.0 .OR. INNON.GT.NSTATE-1) GO TO 310
C
C..TEST REPEAT FLAG
C
      IF (.NOT.REP .AND. (INNON.EQ.LAST)) GO TO 90
```

```
C
C..SHIFT OUT MOST ANCIENT ELEMENT AND SHIFT OTHER ELEMENTS ALONG
C
      NC(1) = NC(1)/MURANG
      IF (NWORD.LT.2) GO TO 110
      DO 100 I=2,NWORD
        JCTEMP = NC(I)/MURANG
        NC(I-1) = NC(I-1) + (NC(I)-JCTEMP*MURANG)*JFIX
        NC(I) = JCTEMP
  100 CONTINUE
  110 NC(NWORD) = NC(NWORD) + JFIX2*INNON
      LAST = INNON
      GO TO 70
C
C..ALL MAXDEP-TUPLES CONSTRUCTED NOW SORT ARRAY KA BY COLUMNS
C
  120 CALL EODSRT(KA, IKA, NGPS, NWORD)
C
C..COUNT GROUPS - DECREASING GROUP SIZE AT THE SAME TIME
C..IF A GROUP IS FOUND TO BE REPEATED M TIMES THIS MEANS
C..THAT EACH SUBGROUP WILL ALSO BE REPEATED M TIMES. TO
C..SAVE HAVING TO RECOUNT THESE SUBGROUPS THE FIRST ELEMENT
C..OF THE COLUMN FOLLOWING THE REPEATED GROUP IS USED TO
C..STORE THE REPEAT COUNT AS -(M-1).  THIS COUNT NEEDS TO
C..BE NEGATIVE TO DISTINGUISH IT FROM THE POSITIVE CODED GROUPS
C
      NW = 1
      IW = INTWRD + 1
      LAST = MAXDEP + 1
      DO 280 K=1,MAXDEP
        IW = IW - 1
        IF (IW.NE.0) GO TO 130
        IW = INTWRD
        NW = NW + 1
        IF (NW.EQ.NWORD) IW = IW3
  130   NGPSIZ = LAST - K
C
C..SET USER DEFINED TRIPWIRES
C
        LIMIT = INT(FLOAT(NGPS)*PERC(NGPSIZ)/100.0)
        SUM = 0.0
        IPASSI = 1
C
C..START TO PROCESS NEW GROUP
C..FIRST - SAVE IT IN ARRAY NC
C
  140   DO 150 I=NW,NWORD
          NC(I) = KA(I,IPASSI)
  150   CONTINUE
C
C..REMOVE LEAST SIGNIFICANT GROUP ELEMENT
C
        KA(NW,IPASSI) = KA(NW,IPASSI)/MURANG
C
C..UNSET IGF (IDENTICAL GROUP FOUND) FLAG
C
        IGF = 0
        NCOUNT = 1
C
C..LOOK AT INFORMATION ON NEXT GROUP (IF ANY)
C
```

```
  160   IPASSI = IPASSI + 1
        IF (IPASSI.GT.NGPS) GO TO 190
        KTEMP = KA(1,IPASSI)
        IF (KTEMP.GE.0) GO TO 170
C
C..A NEGATIVE ELEMENT MEANS THE CONTENTS ARE A REPEAT COUNT
C..SUBTRACT IT FROM BOTH THE POINTER AND THE COUNT
C
        NCOUNT = NCOUNT - KTEMP
        IPASSI = IPASSI - KTEMP
        IF (IPASSI.GT.NGPS) GO TO 190
C
C..REAL GROUP ENCOUNTERED - TEST IF SAME AS LAST ONE
C
  170   DO 180 I=NW,NWORD
          IF (KA(I,IPASSI).NE.NC(I)) GO TO 190
  180   CONTINUE
C
C..IDENTICAL GROUP FOUND  SET IGF FLAG
C
        IGF = 1
        NCOUNT = NCOUNT + 1
        GO TO 160
C
C..NEW GROUP FOUND OR NO MORE GROUPS
C..UPDATE AVERAGE UNCERTAINTY SUM
C
  190   ACCX = FLOAT(NCOUNT)
        SUM = SUM + ACCX*ALOG(ACCX)
        IF (NCOUNT.LE.LIMIT) GO TO 260
C
C..LAST GROUP OCCURS SUFFICIENTLY FREQUENTLY - UNPACK GROUP AND
C..PASS GROUP, SIZE AND FREQUENCY TO USER SUPPLIED ROUTINE
C..OUTFRG
C..FIRST WORD - MAY NOT BE FULL
C
        NGP = NC(NW)
        J = 0
        DO 200 I=1,IW
          J = J + 1
          KTEMP = NGP/MURANG
          JFROG(J) = NGP - KTEMP*MURANG
          NGP = KTEMP
  200   CONTINUE
C
C..MIDDLE WORDS - ALL FULL
C
        IF (NGPSIZ.EQ.IW) GO TO 250
        JCTEMP = NWORD - 1
        IF (NW.EQ.JCTEMP) GO TO 230
        NW2 = NW + 1
        KK = NW
        DO 220 I=NW2,JCTEMP
          KK = KK + 1
          NGP = NC(KK)
          DO 210 L=1,INTWRD
            J = J + 1
            KTEMP = NGP/MURANG
            JFROG(J) = NGP - KTEMP*MURANG
            NGP = KTEMP
  210     CONTINUE
```

```
  220   CONTINUE
C
C..LAST WORD - MAY NOT BE FULL
C
  230   NGP = NC(NWORD)
        DO 240 I=1,IW3
          J = J + 1
          KTEMP = NGP/MURANG
          JFROG(J) = NGP - KTEMP*MURANG
          NGP = KTEMP
  240   CONTINUE
  250   CONTINUE
        CALL OUTFRG(JFROG, NGPSIZ, NCOUNT)
  260   IF (IGF.EQ.0) GO TO 270
C
C..IGF FLAG SET - STORE REPEAT COUNT
C
        KTEMP = IPASSI + 1 - NCOUNT
        KA(1,KTEMP) = 1 - NCOUNT
C
C..ESTIMATE AVERAGE UNCERTAINTY AND CONDITIONAL UNCERTAINTY
C
  270   IF (IPASSI.LE.NGPS) GO TO 140
        ACCX = FLOAT(NGPS)
        AU(NGPSIZ) = -(SUM/ACCX - ALOG(ACCX))/ALN2
  280 CONTINUE
      DO 290 K=2,MAXDEP
        CU(K) = AU(K) - AU(K-1)
  290 CONTINUE
      CU(1) = AU(1)
      CU0 = ALOG(FLOAT(NSTATE))/ALN2
      IFAULT = 0
      RETURN
C
C..ERROR EXITS
C
C..IFAULT=1 - NSTATE-1 NOT STORABLE IN AN INTEGER
C..
C..  OR       NUMBER OF INTEGERS REQUIRED TO STORE TO
C..           GIVEN MAXDEP .GT. IKA
C..
C..  OR       ONE OF IKA, LENGTH, NSTATE, MAXDEP OR
C.            MAXINT .LE. 0
C
  300 IFAULT = 1
      RETURN
C
C..IFAULT=2 - ILLEGAL ACTIVITY NUMBER AT POSITION NPOINT IN
C..           THE PREPARED DATA STRING.  NUMBER EITHER GREATER
C..            THAN NSTATE-1 OR LESS THAN ZERO AND NOT EQUAL TO EOS
C
  310 IFAULT = 2
      RETURN
C
C..IFAULT=3 - DATA STRING IS TO SHORT ENDING AT POSITION NPOINT
C..           IN THE PREPARED DATA STRING
C
  320 IFAULT = 3
      RETURN
      END
      SUBROUTINE EODSRT(KA, IKA, NGPS, NWORD)
```

```
C
C..ROUTINE TO SORT ARRAY A BY COLUMNS FROM IKA TO 1
C..I.E. FIRST SORT KA(IKA,1).LT.KA(IKA,2).LT.KA(IKA,3).......
C..THEN            KA(IKA-1,1).LT.KA(IKA-1,2).LT.KA(IKA-1,3)......
C..E.T.C
C..THIS ROUTINE IS BASED ON M01AQF FROM THE NAG MK8
C..FORTRAN LIBRARY AND SINGLETON (1969)
C
      PARAMETER (MAXKA=10)
      INTEGER IKA, NWORD
      INTEGER KA(IKA,NGPS), IL(22), IU(22)
C
C..NOTE  IF MORE THAN 10 WORDS ARE REQUIRED FOR STORING
C..THE MAXIMUM SIZE GROUP THE FOLLOWING ARRAY
C..DECLARATIONS MUST BE INCREASED.
C
      INTEGER IT(MAXKA), ITT(MAXKA)
      NWORD1 = NWORD + 1
      M = 1
      I = 1
      J = NGPS
      II = I
   10 IF (I-J.GE.0) GO TO 240
   20 K = I
      IJ = (J+I)/2
      DO 30 MM=1,NWORD
        IT(MM) = KA(MM,IJ)
   30 CONTINUE
      MM = NWORD1
      DO 40 NN=1,NWORD
        MM = MM - 1
        IF (KA(MM,I)-IT(MM)) 70, 40, 50
   40 CONTINUE
      GO TO 70
   50 DO 60 MM=1,NWORD
        KA(MM,IJ) = KA(MM,I)
        KA(MM,I) = IT(MM)
        IT(MM) = KA(MM,IJ)
   60 CONTINUE
   70 L = J
      MM = NWORD1
      DO 80 NN=1,NWORD
        MM = MM - 1
        IF (KA(MM,J)-IT(MM)) 90, 80, 160
   80 CONTINUE
      GO TO 160
   90 DO 100 MM=1,NWORD
        KA(MM,IJ) = KA(MM,J)
        KA(MM,J) = IT(MM)
        IT(MM) = KA(MM,IJ)
  100 CONTINUE
      MM = NWORD1
      DO 110 NN=1,NWORD
        MM = MM - 1
        IF (KA(MM,I)-IT(MM)) 160, 110, 120
  110 CONTINUE
      GO TO 160
  120 DO 130 MM=1,NWORD
        KA(MM,IJ) = KA(MM,I)
        KA(MM,I) = IT(MM)
        IT(MM) = KA(MM,IJ)
```

```
130 CONTINUE
    GO TO 160
140 DO 150 MM=1,NWORD
      KA(MM,L) = KA(MM,K)
      KA(MM,K) = ITT(MM)
150 CONTINUE
160 L = L - 1
    MM = NWORD1
    DO 170 NN=1,NWORD
      MM = MM - 1
      IF (KA(MM,L)-IT(MM)) 180, 170, 160
170 CONTINUE
180 DO 190 MM=1,NWORD
      ITT(MM) = KA(MM,L)
190 CONTINUE
200 K = K + 1
    MM = NWORD1
    DO 210 NN=1,NWORD
      MM = MM - 1
      IF (KA(MM,K)-IT(MM)) 200, 210, 220
210 CONTINUE
220 IF (K-L.LE.0) GO TO 140
    IF (L-I-J+K.LE.0) GO TO 230
    IL(M) = I
    IU(M) = L
    I = K
    M = M + 1
    GO TO 250
230 IL(M) = K
    IU(M) = J
    J = L
    M = M + 1
    GO TO 250
240 M = M - 1
    IF (M.EQ.0) GO TO 360
    I = IL(M)
    J = IU(M)
250 IF (J-I-11.GE.0) GO TO 20
    IF (I-II) 270, 10, 270
260 I = I + 1
270 IF (I-J.EQ.0) GO TO 240
    I1 = I + 1
    DO 280 MM=1,NWORD
      IT(MM) = KA(MM,I1)
280 CONTINUE
    MM = NWORD1
    DO 290 NN=1,NWORD
      MM = MM - 1
      IF (KA(MM,I)-IT(MM))260,290,300
290 CONTINUE
    GO TO 260
300 K = I
310 K1 = K + 1
    DO 320 MM=1,NWORD
      KA(MM,K1) = KA(MM,K)
320 CONTINUE
    MM = NWORD1
    K = K - 1
    DO 330 NN=1,NWORD
      MM = MM - 1
      IF (IT(MM)-KA(MM,K)) 310, 330, 340
```

```
330 CONTINUE
340 K1 = K + 1
    DO 350 MM=1,NWORD
      KA(MM,K1) = IT(MM)
350 CONTINUE
    GO TO 260
360 RETURN
    END
```

```
C UKC NETLIB DISTRIBUTION COPYRIGHT 1990 RSS
C
      SUBROUTINE BBML(N, IX, IN, W, P, RL, MRL, ITER, CCRIT, MEW, THETA,
     *  SEM, SETH, RNL, IFAULT)
C
C        ALGORITHM AS189 APPL. STATIST. (1983) VOL.32, NO.2
C
C        SUBROUTINE FOR CALCULATING THE MAXIMUM LIKELIHOOD ESTIMATES
C        OF THE PARAMETERS OF THE BETA BINOMIAL DISTRIBUTION
C
      REAL W(N), P(N), CCRIT, MEW, THETA, SEM, SETH, RNL, INF, DUM,
     *  FD(2), SD(3), TD(4), UB(2), DEL, EPS, A, B, C, D, E, F
      INTEGER IX(N), IN(N), RL(MRL,3), LM(3), RD1(2,2), RD2(2,3),
     *  RD3(2,4)
      LOGICAL MC
      PARAMETER (INF = 1.0E6)
      DATA
     *  RD1(1,1), RD1(2,1), RD1(1,2), RD1(2,2)/1,-1,1,1/,
     *  RD2(1,1), RD2(2,1), RD2(1,2), RD2(2,2),
     *  RD2(1,3), RD2(2,3)/-1,-1,-1,1,-1,-1/,
     *  RD3(1,1), RD3(2,1), RD3(1,2), RD3(2,2), RD3(1,3),
     *  RD3(2,3), RD3(1,4), RD3(2,4)/2,-2,2,2,2,-2,2,2/
C
      I = ITER
      ITER = 0
      MC = .TRUE.
      UB(1) = 0.01
      UB(2) = 0.01
C
C        SET THE ARRAYS RL AND LM
C
      CALL SET(N, IX, IN, RL, MRL, LM, IFAULT)
      IF(IFAULT.NE.0) RETURN
      SEM = -1.0
      SETH = -1.0
      NND = 0
C
C        CALCULATION OF INITIAL ESTIMATES (BY MOMENTS)
C
      CALL BBME(N, IX, IN, W, P, INF, MEW, THETA)
      IF(THETA.EQ.INF) GOTO 50
C
C        NEWTON-RAPHSON ITERATION ON FIRST DERIVATIVES
C
    5 IF(ITER.LE.I) GOTO 10
      IFAULT = 7
      GOTO 60
C
C        CALCULATE FIRST DERIVATIVES OF LOG LIKELIHOOD
C
   10 CALL GDER(MEW, THETA, RL, MRL, LM, 2, RD1, FD)
C
C        CALCULATE SECOND DERIVATIVES OF LOG_LIKELIHOOD
C
      CALL GDER(MEW, THETA, RL, MRL, LM, 3, RD2, SD)
C
C        CALCULATE THIRD DERIVATIVES OF LOG LIKELIHOOD
C
      CALL GDER(MEW, THETA, RL, MRL, LM, 4, RD3, TD)
C
C        CALCULATE INCREMENTS
```

```
C
      DUM = SD(1)*SD(3) - SD(2)*SD(2)
      IF(SD(1).LT.0.0.AND.DUM.GT.0.0) GOTO 15
C
C        NON NEGATIVE DEFINITE MATRIX
C
      NND = NND+1
C
C        SD(1) IS ALWAYS NEGATIVE SO A GRADIENT STEP IS MADE ON MEW
C
      A = MEW - FD(1)/SD(1)
      B = THETA
      IF(FD(2).NE.0.0) B = B + SIGN(UB(2),FD(2))
      IF(A.LE.0.0) A = 0.0001
      IF(A.GE.1.0) A = 0.9999
      IF(B.LT.0.0) B = 0.0
      IF(B.GT.INF) B = INF
      CALL BBL(MEW, THETA, RL, MRL, LM, C)
      CALL BBL(A, B, RL, MRL, LM, D)
      IF(NND.GT.10.OR.C.GE.D) GOTO 40
      ITER = ITER+1
      MEW = A
      THETA = B
      GOTO 5
   15 DEL = (FD(2)*SD(2) - FD(1)*SD(3))/DUM
      EPS = (FD(1)*SD(2) - FD(2)*SD(1))/DUM
C
C        CHECK LIPSCHITZ CONDITION SATISFIED
C
      A = SD(2)*TD(2) - TD(1)*SD(3)
      B = SD(2)*TD(3) - TD(2)*SD(3)
      C = TD(1)*SD(2) - TD(2)*SD(1)
      D = SD(2)*TD(2) - SD(1)*TD(3)
      E = SD(2)*TD(4) - TD(3)*SD(3)
      F = TD(3)*SD(2) - TD(4)*SD(1)
      A = DEL*A + EPS*B
      C = DEL*C + EPS*D
      E = DEL*B + EPS*E
      F = DEL*D + EPS*F
      DUM = (A*A + C*C + E*E + F*F)/(DUM*DUM)
      IF(DUM.GE.1.0) GOTO 20
      IF(ABS(DEL).LE.CCRIT.AND.ABS(EPS).LE.CCRIT) MC = .FALSE.
      GOTO 45
C
C        FAILURE OF LIPSCHITZ CONDITION. A STEP IN THE DIRECTION OF THE
C        GRADIENT IS MADE.
C
   20 A = FD(1)*FD(1)
      B = FD(2)*FD(2)
      C = A*SD(1) + 2.0*SD(2)*FD(1)*FD(2) + B*SD(3)
      IF(C.NE.0.0) GOTO 25
      DEL = 0.0
      IF(FD(1).NE.0.0) DEL = SIGN(UB(1),FD(1))
      EPS = 0.0
      IF(FD(2).NE.0.0) EPS = SIGN(UB(2),FD(2))
      GOTO 30
   25 C = -(A+B)/C
      DEL = C*FD(1)
      EPS = C*FD(2)
      IF(ABS(DEL).GT.UB(1)) DEL = SIGN(UB(1),DEL)
      UB(1) = 2.0*ABS(DEL)
```

```
      IF(ABS(EPS).GT.UB(2)) EPS = SIGN(UB(2),EPS)
      UB(2) = 2.0*ABS(EPS)
   30 CALL BBL(MEW, THETA, RL, MRL, LM, C)
   35 A = MEW + DEL
      B = THETA + EPS
      IF(A.LE.0.0) A = 0.0001
      IF(A.GE.1.0) A = 0.9999
      DEL = A - MEW
      IF(B.LT.0.0) B = 0.0
      IF(B.GT.INF) B = INF
      EPS = B - THETA
      CALL BBL(A, B, RL, MRL, LM, D)
C
C        CHECK TO SEE IF GRADIENT STEP HAS INCREASED LOG LIKELIHOOD
C
      IF(D.GT.C) GOTO 45
      DEL = DEL/2.0
      EPS = EPS/2.0
      IF(ABS(DEL).GT.CCRIT.OR.ABS(EPS).GT.CCRIT) GOTO 35
   40 IFAULT = 8
      GOTO 60
   45 ITER = ITER + 1
      A = MEW + DEL
      B = THETA + EPS
      IF(A.GT.0.0.AND.A.LT.1.0.AND.B.GE.0.0.AND.B.LE.INF) GOTO 55
      IF(A.LE.0.0) MEW = 0.0
      IF(A.GE.1.0) MEW = 1.0
      IF(B.LT.0.0) THETA = 0.0
      IF(B.GT.INF) THETA = INF
   50 IFAULT = 6
      GOTO 60
   55 MEW = A
      THETA = B
      IF(MC) GOTO 5
C
C        CALCULATE LOG LIKELIHOOD AND S.E.S
C
      IF(SD(1).LT.0.0) SEM = SQRT(-1.0/SD(1))
      IF(SD(3).LT.0.0) SETH = SQRT(-1.0/SD(3))
   60 CALL BBL(MEW, THETA, RL, MRL, LM, RNL)
      RETURN
      END
C
      SUBROUTINE BBME(N, IX, IN, W, P, INF, MEW, THETA)
C
C        ALGORITHM AS 189.1 APPL. STATIST. (1983) VOL.32, NO.2
C
C        SUBROUTINE TO ESTIMATE MEW AND THETA OF THE BETA BINOMIAL
C        DISTRIBUTION BY THE METHOD OF MOMENTS
C
      REAL W(N), P(N), INF, MEW, THETA, D1, D2, R, S, TP, WT
      INTEGER IX(N), IN(N)
      LOGICAL J
C
      J = .FALSE.
      DO 5 I = 1,N
        W(I) = FLOAT(IN(I))
        P(I) = FLOAT(IX(I))/W(I)
    5 CONTINUE
   10 WT = 0.0
      TP = 0.0
```

```
      DO 15 I = 1,N
        WT = WT+W(I)
        TP = TP+W(I)*P(I)
   15 CONTINUE
      TP = TP/WT
      S = 0.0
      D1 = 0.0
      D2 = 0.0
      DO 20 I = 1,N
        R = P(I)-TP
        S = S+W(I)*R*R
        R = W(I)*(1.0-W(I)/WT)
        D1 = D1+R/FLOAT(IN(I))
        D2 = D2+R
   20 CONTINUE
      S = FLOAT(N-1)*S/FLOAT(N)
      R = TP*(1.0-TP)
      IF(R.EQ.0.0) GOTO 30
      R = (S-R*D1)/(R*(D2-D1))
      IF(R.LT.0.0) R = 0.0
      IF(J) GOTO 30
      DO 25 I = 1,N
   25 W(I) = W(I)/(1.0+R*(W(I)-1.0))
      J = .TRUE.
      GOTO 10
   30 MEW = TP
      IF(R.GE.1.0) GOTO 35
      THETA = R/(1.0-R)
      IF(THETA.LE.INF) RETURN
   35 THETA = INF
      RETURN
      END
C
      SUBROUTINE SET(N, IX, IN, RL, MRL, LM, IFAULT)
C
C        ALGORITHM AS 189.2 APPL. STATIST. (1983) VOL.32, NO.2
C
C        SUBROUTINE FOR SETTING UP ARRAY FOR CALCULATION OF
C        THE BETA BINOMIAL LOG LIKELIHOOD AND ITS DERIVATIVES
C
      INTEGER IX(N), IN(N), RL(MRL,3), LM(3)
C
C     TEST ADMISSIBILITY OF DATA
C
      IF(N.GT.1) GOTO 5
      IFAULT = 1
      RETURN
    5 DO 10 I = 1,N
        IF(IX(I).GT.0) GOTO 15
   10 CONTINUE
      IFAULT = 2
      RETURN
   15 DO 20 I = 1,N
        IF(IX(I).LT.IN(I)) GOTO 25
   20 CONTINUE
      IFAULT = 3
      RETURN
C
C        FORM MATRIX OF COUNTS
C
   25 IFAULT = 4
```

```
      DO 30 I = 1,3
        LM(I) = 0
        DO 30 J = 1,MRL
          RL(J,I) = 0
   30 CONTINUE
      DO 65 I = 1,N
        JJ = IX(I)
        MAR = 1
        GOTO 45
   35   JJ = IN(I)-IX(I)
        MAR = 2
        GOTO 45
   40   JJ = IN(I)
        MAR = 3
   45   IF(JJ) 50,60,55
   50   IFAULT = 5
        RETURN
   55   IF(JJ.GT.MRL) RETURN
        IF(JJ.GT.LM(MAR)) LM(MAR) = JJ
        RL(JJ,MAR) = RL(JJ,MAR)+1
   60   GOTO(35,40,65) MAR
   65 CONTINUE
      IFAULT = 0
C
C       EVALUATE NUMBER OF CALLS TO DIFFERENT TERMS OF LIKELIHOOD
C       FUNCTION
C
      DO 75 I = 1,3
        JJ = LM(I)-1
        IF(JJ.LE.0) GOTO 75
        K = JJ
        DO 70 J = 1,JJ
          RL(K,I) = RL(K,I)+RL(K+1,I)
          K = K-1
   70   CONTINUE
   75 CONTINUE
      RETURN
      END
C
      SUBROUTINE BBL(MEW, THETA, RL, MRL, LM, RNL)
C
C       ALGORITHM AS 189.3 APPL. STATIST. (1983) VOL.32, NO.2
C
C       SUBROUTINE FOR CALCULATION OF THE BETA BINOMIAL LOG
C       LIKELIHOOD
C
      REAL MEW, THETA, RNL, A
      INTEGER RL(MRL,3), LM(3)
C
      RNL = 0.0
      MLM = LM(3)
      DO 5 I = 1,MLM
        A = FLOAT(I-1)*THETA
        IF(I.LE.LM(1))RNL = RNL + FLOAT(RL(I,1))*ALOG(MEW+A)
        IF(I.LE.LM(2))RNL = RNL + FLOAT(RL(I,2))*ALOG(1.0-MEW+A)
        RNL = RNL - FLOAT(RL(I,3))*ALOG(1.0+A)
    5 CONTINUE
      RETURN
      END
C
      SUBROUTINE GDER(MEW, THETA, RL, MRL, LM, IDER, RD, PD)
```

```
C
C        ALGORITHM AS 189.4 APPL. STATIST. (1983) VOL.32, NO.2
C
C        GENERAL DERIVATIVE SUBROUTINE
C
      REAL MEW, THETA, PD(IDER), A, B, C, D
      INTEGER RL(MRL,3), LM(3), RD(2,IDER)
C
      MLM = LM(3)
      KK = IDER-1
      DO 5 I = 1,IDER
    5 PD(I) = 0.0
      DO 45 I = 1,MLM
        C = FLOAT(I-1)
        A = C*THETA
        DO 40 J = 1,3
          IF(I.GT.LM(J)) GOTO 40
          GOTO (10,15,20) J
   10     D = MEW+A
          GOTO 25
   15     D = 1.0-MEW+A
          GOTO 25
   20     D = 1.0+A
   25     B = FLOAT(RL(I,J))/D**KK
          IF(J.EQ.3) GOTO 35
          DO 30 K = 1,IDER
            PD(K) = PD(K)+FLOAT(RD(J,K))*B
            B = B*C
   30     CONTINUE
          GOTO 40
   35     D = -FLOAT(RD(1,1))*B*C**KK
          PD(IDER) = PD(IDER)+D
   40   CONTINUE
   45 CONTINUE
      RETURN
      END
```

```
      double precision function prtrng(q, v, r, ifault)
      implicit double precision (a-h, o-z)
c
c        Algorithm AS 190  Appl. Statist. (1983) Vol.32, No. 2
c        Incorporates corrections from Appl. Statist. (1985) Vol.34 (1)
c
c        Evaluates the probability from 0 to q for a studentized
c        range having v degrees of freedom and r samples.
c
c        Uses subroutine ALNORM = algorithm AS66.
c
c        Arrays vw and qw store transient values used in the
c        quadrature summation.  Node spacing is controlled by
c        step.  pcutj and pcutk control truncation.
c        Minimum and maximum number of steps are controlled by
c        jmin, jmax, kmin and kmax.  Accuracy can be increased
c        by use of a finer grid - Increase sizes of arrays vw
c        and qw, and jmin, jmax, kmin, kmax and 1/step proportionally.
c
      double precision q,v,r,vw(30), qw(30), pcutj, pcutk, step,
     #    vmax,zero,fifth,half,one,two,cv1,cv2,cvmax,cv(4)
      double precision g, gmid, r1, c, h, v2, gstep, pk1, pk2, gk, pk
      double precision w0, pz, x, hj, ehj, pj
      data pcutj, pcutk, step, vmax /0.00003d0, 0.0001d0, 0.45d0,
     #   120.0d0/, zero, fifth, half, one, two /0.0d0, 0.2d0, 0.5d0,
     #   1.0d0, 2.0d0/, cv1, cv2, cvmax /0.193064705d0, 0.293525326d0,
     #   0.39894228d0/, cv(1), cv(2), cv(3), cv(4) /0.318309886d0,
     #   -0.268132716d-2, 0.347222222d-2, 0.833333333d-1/
      data jmin, jmax, kmin, kmax/3, 15, 7, 15/
c
c        Check initial values
c
      prtrng = zero
      ifault = 0
      if (v .lt. one .or. r .lt. two) ifault = 1
      if (q .le. zero .or. ifault .eq. 1) goto 99
c
c        Computing constants, local midpoint, adjusting steps.
c
      g = step * r ** (-fifth)
      gmid = half * log(r)
      r1 = r - one
      c = log(r * g * cvmax)
      if(v.gt.vmax) goto 20
c
      h = step * v ** (-half)
      v2 = v * half
      if (v .eq. one) c = cv1
      if (v .eq. two) c = cv2
      if (.not. (v .eq. one .or. v .eq. two)) c = sqrt(v2)
     #    * cv(1) / (one + ((cv(2) / v2 + cv(3)) / v2 + cv(4)) / v2)
      c = log(c * r * g * h)
c
c        Computing integral
c        Given a row k, the procedure starts at the midpoint and works
c        outward (index j) in calculating the probability at nodes
c        symmetric about the midpoint.  The rows (index k) are also
c        processed outwards symmetrically about the midpoint.  The
c        centre row is unpaired.
c
   20 gstep = g
```

```
      qw(1) = -one
      qw(jmax + 1) = -one
      pk1 = one
      pk2 = one
      do 28 k = 1, kmax
        gstep = gstep - g
   21   gstep = -gstep
        gk = gmid + gstep
        pk = zero
        if (pk2 .le. pcutk .and. k .gt. kmin) goto 26
        w0 = c - gk * gk * half
        pz = alnorm(gk, .true.)
        x = alnorm(gk - q, .true.) - pz
        if (x .gt. zero) pk = exp(w0 + r1 * log(x))
        if (v .gt. vmax) goto 26
c
        jump = -jmax
   22   jump = jump + jmax
        do 24 j = 1, jmax
          jj = j + jump
          if (qw(jj) .gt. zero) goto 23
          hj = h * j
          if (j .lt. jmax) qw(jj + 1) = -one
          ehj = exp(hj)
          qw(jj) = q * ehj
          vw(jj) = v * (hj + half - ehj * ehj * half)
c
   23     pj = zero
          x = alnorm(gk - qw(jj), .true.) - pz
          if (x .gt. zero) pj = exp(w0 + vw(jj) + r1 * log(x))
          pk = pk + pj
          if (pj .gt. pcutj) goto 24
          if (jj .gt. jmin .or. k .gt. kmin) goto 25
   24   continue
   25   h = -h
        if (h .lt. zero) goto 22
c
   26   prtrng = prtrng + pk
        if (k .gt. kmin .and. pk .le. pcutk .and. pk1 .le. pcutk)goto 99
        pk2 = pk1
        pk1 = pk
        if (gstep .gt. zero) goto 21
   28 continue
c
   99 return
      end
c
c
      double precision function qtrng(p, v, r, ifault)
      implicit double precision (a-h, o-z)
c
c       Algorithm AS 190.1  Appl. Statist. (1983) Vol.32, No. 2
c
c       Approximates the quantile p for a studentized range
c       distribution having v degrees of freedom and r samples
c       for probability p, p.ge.0.90 .and. p.le.0.99.
c
c       Uses functions  alnorm, ppnd, prtrng and qtrng0 -
c       Algorithms AS 66, AS 111, AS 190 and AS 190.2
c
      double precision p, v, r, pcut, p75, p80, p90, p99, p995
```

```
      double precision p175, one, two, five
      double precision q1, p1, q2, p2, e1, e2
      double precision eps
      data jmax, pcut, p75, p80, p90, p99, p995, p175, one, two, five
     #  /8, 0.001d0, 0.75d0, 0.80d0, 0.90d0, 0.99d0, 0.995d0, 1.75d0,
     #  1.0d0, 2.0d0, 5.0d0/
      data eps/1.0d-4/
c
c        Check input parameters
c
      ifault = 0
      nfault = 0
      if (v .lt. one .or. r.lt. two) ifault = 1
      if (p .lt. p90 .or. p .gt. p99) ifault = 2
      if (ifault .ne. 0) goto 99
c
c        Obtain initial values
c
      q1 = qtrng0(p, v, r, nfault)
      if (nfault .ne. 0) goto 99
      p1 = prtrng(q1, v, r, nfault)
      if (nfault .ne. 0) goto 99
      qtrng = q1
      if (abs(p1-p) .lt. pcut) goto 99
      if (p1 .gt. p) p1 = p175 * p - p75 * p1
      if (p1 .lt. p) p2 = p + (p - p1) * (one - p) / (one - p1) * p75
      if (p2 .lt. p80) p2 = p80
      if (p2 .gt. p995) p2 = p995
      q2 = qtrng0(p2, v, r, nfault)
      if (nfault .ne. 0) goto 99
c
c        Refine approximation
c
      do 14 j = 2, jmax
        p2 = prtrng(q2, v, r, nfault)
        if (nfault .ne. 0) goto 99
        e1 = p1 - p
        e2 = p2 - p
        qtrng = (q1 + q2) / two
        d = e2 - e1
        if (abs(d) .gt. eps) qtrng = (e2 * q1 - e1 * q2) / d
        if(abs(e1) .lt. abs(e2)) goto 12
        q1 = q2
        p1 = p2
   12   if (abs(p1 - p) .lt. pcut * five) goto 99
        q2 = qtrng
   14 continue
c
   99 if (nfault .ne. 0) ifault = 9
      return
      end
c
c
      double precision function qtrng0(p, v, r, ifault)
      implicit double precision (a-h, o-z)
c
c        Algorithm AS 190.2  Appl. Statist. (1983) Vol.32, No.2
c
c        Calculates an initial quantile p for a studentized range
c        distribution having v degrees of freedom and r samples
c        for probability p, p.gt.0.80 .and. p.lt.0.995.
```

```
c
c       Uses function ppnd - Algorithm AS 111
c
      double precision p, v, r, q, t, vmax, half, one, four, c1, c2, c3
      double precision c4, c5
      data vmax, half, one, four, c1, c2, c3, c4, c5 / 120.0d0, 0.5d0,
     #  1.0d0, 4.0d0, 0.8843d0, 0.2368d0, 1.214d0, 1.208d0, 1.4142d0/
c
      t=ppnd(half + half * p,ifault)
      if (v .lt. vmax) t = t + (t * t* t + t) / v / four
      q = c1 - c2 * t
      if (v .lt. vmax) q = q - c3 / v + c4 * t / v
      qtrng0 = t * (q * log(r - one) + c5)
      return
      end
```

```
      subroutine sarmas(z, nz, n, beta, nbeta, ip, iq, ips, iqs, isea,
     +          iqap, maxit, a, s, sm, w, nw, ifault)
c
c     ALGORITHM AS 191  APPL. STATIST. (1983) VOL. 32, NO. 2
c
      double precision z(nz), a(nz), w(nw), beta(nbeta), s, sm
c
c     Local variables
c
      logical switch
      double precision zero, one, oneneg, etol, sprev, detm, detms,
     +          relerr, temp
c
c     Initialize numerical constants
c
      data zero/0.d0/, one/1.d0/, oneneg/-1.d0/
c
c     ETOL = error tolerance in convergence criterion
c
      data etol/1.d-10/
c
      iter = 0
      switch = .false.
      sprev = zero
      ipq = ip + iq
      ipqps = ipq + ips
      ipsts = ips + isea
      ipsts1 = ipsts + 1
      iqsts = iqs + isea
      iqsts1 = iqsts + 1
      ipsqs = ips + iqs
      iqap2 = iqap
      if (ip .eq. 0 .and. ips .eq. 0) iqap2 = min(iqap, iq + iqsts)
      maxit2 = maxit
      if (iq .eq. 0 .and. iqs .eq. 0) maxit2 = 0
      if (maxit2 .eq. 0) switch = .true.
      nby2 = n / 2
c
c     Input validation
c
      ifault = 0
      ir = max(iq + iqsts, ip + ipsts)
      if (ir .ge. n) ifault = 5
      if (maxit .gt. 0 .and. ir .gt. iqap) ifault = 7
      if (ipq + ipsqs .gt. nbeta) ifault = 4
      if (n + iqap2 .gt. nz) ifault = 4
      if (nw .lt. max(nz, 1 + ipq * (ipq + 3)/2,
     +          1 + ipsqs * (ipsqs + 3)/2)) ifault = 4
      if (min(ip, iq, ips, iqs, isea, iqap, maxit) .lt. 0) ifault = 6
      if (ifault .ge. 1) return
c
c     Obtain necessary determinants.
c     Check for stationarity/invertibility
c
      detm = one
      detms = one
      ier = 0
      if (ipq .ne. 0) call dtarma(beta, ipq, ip, iq, w, nw, detm, ier)
      if (ier .gt. 0) go to 340
      if (ipsqs .eq. 0) go to 20
      ii = ipq
```

```
      do 10 i = 1, ipsqs
        ii = ii + 1
        a(i) = beta(ii)
   10 continue
      call dtarma(a, ipsqs, ips, iqs, w, nw, detms, ier)
      if (ier .gt. 0) go to 340
c
c     If IQAP2 = 0, use conditional sum of squares method
c
   20 if (iqap2 .eq. 0) go to 200
c
c     If no seasonal component and no moving-average component, proceed
c     directly to back-forecasting step (Y and E-series not needed)
c
      if (ipsqs .eq. 0 .and. iq .eq. 0) go to 110
c
c     Calculate Y-series, use W-vector
c
      do 60 i = 1, n
        w(i) = zero
        if (i .le. ip) go to 60
        w(i) = z(i)
        if (ip .eq. 0) go to 40
        do 30 j = 1, ip
          iii = i - j
          w(i) = w(i) - beta(j) * z(iii)
   30   continue
   40   l = min(iq, i - 1)
        if (l .eq. 0) go to 60
        do 50 j = 1, l
          jj = ip + j
          iii = i - j
          w(i) = w(i) + beta(jj) * w(iii)
   50   continue
   60 continue
c
c     Calculate E-series, use A-vector
c
      lqs = iqs
      do 100 i = 1, n
        a(i) = zero
        if (i .le. ipsts) go to 100
        a(i) = w(i)
        if (ips .eq. 0) go to 80
        iii = i
        jj = ipq
        do 70 j = 1, ips
          iii = iii - isea
          jj = jj + 1
          a(i) = a(i) - beta(jj) * w(iii)
   70   continue
   80   if (iqs .eq. 0) go to 100
        if (i .le. iqsts1) lqs = (i - 1) / isea
        if (lqs .eq. 0) go to 100
        iii = i
        do 90 j = 1, lqs
          iii = iii - isea
          jj = ipqps + j
          a(i) = a(i) + beta(jj) * a(iii)
   90   continue
  100 continue
```

```
c
c     Back-forecast Y-series, use w(n+1), w(n+2), ...
c
  110 do 150 i = 1, iqap2
        npi = n + i
        w(npi) = zero
        a(npi) = zero
        if (i .gt. iqsts) go to 130
        iii = npi
        do 120 j = 1, iqs
          iii = iii - isea
          jj = ipqps + j
          w(npi) = w(npi) - beta(jj) * a(iii)
  120   continue
  130   if (ips .eq. 0) go to 150
        iii = npi
        do 140 j = 1, ips
          iii = iii - isea
          jj = ipq + j
          w(npi) = w(npi) + beta(jj) * w(iii)
  140   continue
  150 continue
c
c     Back-forecast Z-series, use z(n+1), z(n+2), ...
c
      do 190 i = 1, iqap2
        npi = n + i
        z(npi) = w(npi)
        if (iq .eq. 0) go to 170
        do 160 j = 1, iq
          npimj = npi - j
          jj = ip + j
          z(npi) = z(npi) - beta(jj) * w(npimj)
  160   continue
  170   if (ip .eq. 0) go to 190
        do 180 j = 1, ip
          npij = npi - j
          z(npi) = z(npi) + beta(j) * z(npij)
  180   continue
  190 continue
c
c     Calculate X-series, use W-vector
c
  200 npqap = n + iqap2
      ii = npqap + 1
      do 240 i = 1, npqap
        ii = ii - 1
        w(ii) = z(ii)
        im1 = i - 1
        l = min(im1, ip)
        if (l .eq. 0) go to 220
        iii = ii
        do 210 j = 1, l
          iii = iii + 1
          w(ii) = w(ii) - beta(j) * z(iii)
  210   continue
  220   l = min(im1, iq)
        if (l .eq. 0) go to 240
        iii = ii
        do 230 j = 1, l
          iii = iii + 1
```

```
            jj = ip + j
            w(ii) = w(ii) + beta(jj) * w(iii)
  230     continue
  240   continue
c
c       Calculate A-series, use A-vector
c
        ii = npqap + 1
        do 280 i = 1, npqap
          ii = ii - 1
          a(ii) = w(ii)
          if (isea .eq. 0) go to 280
          if (i .le. ipsts1) lps = (i - 1) / isea
          if (lps .eq. 0) go to 260
          iii = ii
          do 250 j = 1, lps
            iii = iii + isea
            jj = ipq + j
            a(ii) = a(ii) - beta(jj) * w(iii)
  250     continue
  260     if (i .le. iqsts1) lqs = (i - 1) / isea
          if (lqs .eq. 0) go to 280
          iii = ii
          do 270 j = 1, lqs
            iii = iii + isea
            jj = ipqps + j
            a(ii) = a(ii) + beta(jj) * a(iii)
  270     continue
  280   continue
c
c       Calculate the sum of squares
c
        s = zero
        do 300 i = 1, npqap
  300   s = s + a(i) * a(i)
c
c       Test for convergence
c
        if (iqap2 .eq. 0) go to 330
        if (switch) go to 310
        ifault = 0
        relerr = (s - sprev) / s
        if (abs(relerr) .le. etol) go to 330
c
c       Convergence not obtained
c
  310   ifault = 1
        if (iter .ge. maxit2) go to 330
c
c       Reverse the series and proceed to the forecasting step.
c
        sprev = s
        ii = n
        do 320 i = 1, nby2
          temp = w(ii)
          w(ii) = w(i)
          w(i) = temp
          temp = a(ii)
          a(ii) = a(i)
          a(i) = temp
          temp = z(ii)
```

```
            z(ii) = z(i)
            z(i) = temp
            ii = ii - 1
      320 continue
          if (switch) iter = iter + 1
          switch = .not. switch
          go to 110
c
c     Modified sum of squares
c
      330 temp = oneneg / float(n)
          sm = s * detm**temp * detms**(float(isea) * temp)
          if (maxit2 .eq. 0) ifault = 0
          return
c
c     Model is nonstationary or noninvertible
c
      340 ifault = ier + 1
          return
          end
c
c
c
          subroutine dtarma(beta, nbeta, ip, iq, ws, nws, detm, ifault)
c
c     ALGORITHM AS 191.1  APPL. STATIST. (1983) VOL. 32, NO. 2
c
          double precision beta(nbeta), ws(nws), detm
c
c     Local variables
c
          double precision zero, one, oneneg, det, det1
          data zero/0.d0/, one/1.d0/, oneneg/-1.d0/
c
          ifault = 0
          if (nbeta .lt. ip + iq) go to 140
          nwchek = 1 + nbeta * (nbeta + 3)/2
          if (nwchek .gt. nws) go to 140
          det = one
          det1 = one
          ir = ip + iq
          irs = nws - ir - 1
          irsp1 = irs + 1
          ws(irsp1) = oneneg
          isw = 0
          if (ip .eq. 0) isw = 1
          iloop = ip
       10 if (isw .eq. 1) iloop = iq
          if (iloop .eq. 0) go to 120
          if (isw .eq. 2) go to 30
          do 20 i = 1, iloop
            irspi = irs + i + 1
            ippi = isw * ip + i
            ws(irspi) = beta(ippi)
       20 continue
          go to 60
       30 if (ip .eq. 0) go to 120
          iloop = ir
c
c     Multiply the autoregressive and moving-average operators to obtain
c     coefficients in the left-adjoint AR(ip+iq) model
```

```
c
      do 50 i = 1, ir
        ii = irs + 1 + i
        ws(ii) = zero
        imiq = i - iq
        j1 = max(0, imiq) + 1
        j2 = min(i, ip) + 1
        do 40 j = j1, j2
          jm1 = j - 1
          if (j .eq. 1) then
            bj = oneneg
          else
            bj = beta(jm1)
          end if
          imj = i - j + 1
          ippimj = ip + imj
          if (imj .eq. 0) then
            bi = oneneg
          else
            bi = beta(ippimj)
          end if
          ws(ii) = ws(ii) - bi * bj
   40   continue
   50 continue
c
c     Form the Schur matrix
c
   60 m = 0
      iend = iloop + 1
      do 90 i = 1, iloop
        do 80 j = 1, i
          m = m + 1
          ws(m) = zero
          l = min(i, j)
          do 70 k = 1, l
            irsi = irs + i - k + 1
            irsj = irs + j - k + 1
            irspi = irs + iend - i + k
            irspj = irs + iend - j + k
            ws(m) = ws(m) + ws(irsi) * ws(irsj)
            ws(m) = ws(m) - ws(irspi) * ws(irspj)
   70     continue
   80   continue
   90 continue
c
c     Calculate the determinant using the modified Cholesky
c     decomposition
c
      call mchol(ws, nws, iloop, det, ifault)
      if (ifault .gt. 0) go to 130
c
      if (isw .ge. 1) go to 110
      isw = 1
      det1 = det * det
      go to 10
c
  110 if (isw .eq. 2) go to 120
      isw = 2
      det1 = det1 * det * det
      go to 10
c
```

```
  120 detm = det1 / det
      return
c
  130 ifault = isw + 1
      return
  140 ifault = 3
      return
      end
c
c
c
      subroutine mchol(a, na, n, det, ifault)
c
c        Algorithm AS 191.2  Appl. Statist. (1983) Vol. 32, No. 2
c
c     This algorithm performs the modified Cholesky decomposition
c     A=LDL' where L is lower triangular and D is diagonal.  This
c     form of decomposition avoids the square-root computation of
c     the standard decomposition.
c
c     Parameters:-
c
c     a     Double precision array (na)
c                               input:  Positive definite matrix stored in
c                                       symmetric mode as a11, a21, a22,...
c                                       amm.
c                               output: Modified Cholesky decomposition
c                                       stored as a one-dimensional array
c                                       as d11,l21,d22,l31,l32,d33,....,
c                                       lm1,lm2,...,lmm-1,dmm.
c     na     Integer            input:  m*(m+1)/2
c     n      Integer            input:  m, order of input matrix
c     det    Double precision   output: Determinant of a
c     ifault Integer            output: Fault Indicator
c                                       =1 if na or n is invalid
c                                       =2 if matrix a is not positive-definite
c                                       =0 otherwise
c
      double precision a(na), eta, one, det, w, t, tt
      data one /1.0d0/
c
c        eta  - largest number such that 1.0 + eta = 1.0
c        (depends upon machine precision)
c
       data eta /1.0d-14/
c
      ifault = 1
      det = one
      if (n .le. 0) goto 70
      if (na .lt. n*(n+1)/2) goto 70
      ifault = 2
      j = 1
      k = 0
      do 60 irow = 1,n
        l = 0
        do 20 icol = 1,irow
          k = k+1
          w = a(k)
          m = j
          if (irow .eq. icol) goto 30
          do 10 i = 1,icol
```

```
               l = l+1
               if (i .eq. icol) goto 20
               w = w-a(l)*a(m)
               m = m+1
      10     continue
      20     a(k) = w
c
      30   ii = 0
           do 40 i = 1,icol
             if(i .eq. icol) goto 50
             ii = ii+i
             t = a(m)
             tt = a(m) / a(ii)
             w = w - t * tt
             a(m) = tt
             m  =  m+1
      40   continue
      50   if (w .lt. eta*abs(a(k))) goto 70
           a(k) = w
           det = det*w
           j = j+irow
      60 continue
         ifault = 0
      70 return
         end
```

```
SUBROUTINE pearsn(xbar, sd, itype, rb1, b2, bndry, sigpt, ifault)

! Code converted using TO_F90 by Alan Miller
! Date: 1999-09-25  Time: 16:22:51
! N.B. Array a has been removed from the list of arguments,
!      and replaced with a PARAMETER statement in this routine.

!     ALGORITHM AS 192  APPL. STATIST. (1983) VOL.32, NO.3

!     Computes approximate significance points of a Pearson curve with given
!     first four moments, or first three moments and left or right boundary.

! Error codes
! IFAULT = 0  successful completion
!        = 1 SD < 0.0
!        = 2 ITYPE < 1 or > 3
!        = 3 | RB1 | > 2.0
!        = 4 XBAR value impossible with the value entered for BNDRY
!        = 5 B2 cannot be computed for the first 3 moments + left boundary
!        = 6 B2 out of range, that is:
!            If | RB1 | <= 1.0 then
!               B2 < 1.5 * | RB1 | + 1.5  or  B2 > 0.2 * | RB1 | + 10.8
!            If | RB1 | > 1.0 then
!               B2 < 3.9 * | RB1 | - 0.9  or  B2 > 4.8 * | RB1 | + 6.2

IMPLICIT NONE

REAL, INTENT(IN)      :: xbar
REAL, INTENT(IN)      :: sd
INTEGER, INTENT(IN)   :: itype
REAL, INTENT(IN)      :: rb1
REAL, INTENT(IN OUT)  :: b2
REAL, INTENT(IN)      :: bndry
REAL, INTENT(OUT)     :: sigpt(:)
INTEGER, INTENT(OUT)  :: ifault

REAL    :: b(9), rbeta1, ca, ca2, carb1, denom, absca, bnd1, bnd2, sgn, &
           pi1, pi2
INTEGER :: i, j, k, l
REAL, PARAMETER :: zero = 0.0, point2 = 0.2, rthalf = 0.70710678,  &
                   point9 = 0.9, one = 1.0, onept5 = 1.5, two = 2.0, &
                   three = 3.0, thrpt9 = 3.9, four = 4.0, forpt8 = 4.8, &
                   sixpt2 = 6.2, tenpt8 = 10.8
REAL, PARAMETER :: a(418) = (/    &
-1.9336, -1.6061, 2.6955, -1.7036, 1.7236, -1.3209, -2.5464, 1.6812, -0.11812,
0.050875, 0.16042, -1.0616, 0.43929, &
-0.35799, 0.51040, 1.0958, -0.63453, 0.053175, -0.019945, 0.25597, 0.32520,
-0.094571, 2.0418, -2.5786, 0.61148, -0.64242, &
 1.1968, -0.33145, 0.015508, -4.1444, 0.13884, 4.6625, 0.37661, -0.20447,
-1.9508, -0.20214, 0.11921, -0.0085268, &

-1.6453, -1.8494, 2.3915, -1.5844, 1.8290, -1.1167, -3.6091, 2.7150, -0.45857,
0.048102, 0.89117, -1.2700, 0.37653, &
-0.81542, 0.56815, 1.6562, -1.1908, 0.24574, -0.024375, -1.5063, 4.4876,
-0.60765, -6.5584, 2.8944, -0.42381, 2.2664, &
-1.3425, 0.25806, -0.010421, -1.9526, 0.21332, 2.1317, -1.1996, 0.21033,
-0.52154, 0.53597, -0.12355, 0.0060658, &

-1.1044, -1.1300, 1.7681, 0.10947, 0.30566, -0.90598, -0.98814, 0.49355,
0.30625, -0.018707, 0.74941, -1.3174, 0.23974, &
-0.12433, 0.59105, 0.85714, -0.47292, -0.16028, 0.012684, -1.4645, 4.5349,
```

```
   -0.71053, -7.9213, 4.0018, -0.61932, 2.7320, &
   -1.7752, 0.35919, -0.014944, -2.0999, 0.30754, 3.6865, -2.3140, 0.39961,
   -1.1530, 1.0257, -0.22981, 0.011032, &

   -0.85842, 0.78929, 1.2196, -0.55088, -0.96385, -0.57312, -0.076974, 0.34882,
   0.47311, -0.056120, -0.63153, -1.2697, 0.64649, &
    0.92712, 0.49088, 0.48118, -0.46407, -0.39753, 0.052413, -0.92873, 2.2039,
   0.25828, 0.74312, -2.4307, 0.49934, 0.029833, &
    0.68249, -0.19687, 0.0068136, -2.9758, -0.050738, 0.067069, 2.0835, -0.45211,
   -0.28149, -0.58012, 0.17621, -0.0067195, &

   -2.4031, -1.4891, 0.75590, -2.3309, 9.2314, -3.8930, -7.0192, 2.3648, 0.44308,
   -0.029437, 16.616, -4.9826, 1.0626, &
   -18.888, 7.2830, 14.642, -2.6394, -1.2739, 0.060862, -1.6092, 0.81385,
   -0.059847, -0.18180, 1.2679, -0.36205, 0.29683, &
   -0.83310, 0.26630, -0.015081, 0.43536, 0.27965, 1.2516, -3.0536, 0.65936,
   -1.0579, 1.7886, -0.47962, 0.025311, &

   -2.0711E-4, 0.0068277, -7.5482E-6, 0.47086, -0.21213, 1.1003E-4, -0.22462,
   0.11733, -0.079682, -1.0565E-5, 2.5087, -2.6169, &
    0.94103, -2.3261, 1.6482, 2.3024, -1.4821, -0.020294, 9.9168E-4, -0.097078,
   0.58192, -0.019936, -0.7025, 0.11526, 0.0013959, &
    0.1595, -0.062382, 0.01338, 1.8557E-4, -1.8383, -0.54124, 0.68041, 1.7654,
   -0.37606, -1.113, 0.044447, 0.042456, -0.0017448, &

    0.39672, -0.17140, -0.74246, 0.48553, 0.19650, 0.58942, -1.5768, 0.37082,
   -0.32481, 0.025296, -0.60818, -1.9005, 0.45451, &
    0.79612, 1.0544, -1.9010, 0.29849, -0.51728, 0.050926, 0.45223, -2.4004,
   0.38372, 3.6817, -1.0796, 0.016619, -1.6829, &
    0.81147, -0.10376, 0.0052116, -4.3837, 0.67456, 6.3029, -1.8042, 0.023516,
   -2.9320, 1.4537, -0.20680, 0.010634, &

    0.77212, 0.028272, -1.3932, 0.95628, -0.37918, 0.85487, 0.38225, -1.3009,
   0.42400, -0.051933, 0.17133, -1.5330, 1.2721, &
   -0.58446, 0.78993, 0.33412, -1.2988, 0.39155, -0.046683, 2.2548, -1.7005,
   -0.65377, -2.9256, 2.1397, 0.093244, 3.2094, &
   -2.2896, 0.24580, -0.0020536, 0.30478, -0.58544, -4.0524, 1.8239, 0.083952,
   3.4072, -2.0163, 0.20345, -7.5748E-4, &

    1.1504, 0.22674, -1.7407, 1.1356, -0.58596, 0.88839, 0.34743, -1.2531, 0.42319,
   -0.014362, 0.17074, -1.2892, 1.1653, &
   -0.46699, 0.58479, 0.33975, -1.0005, 0.25271, -0.0091547, 2.1265, -8.7599,
   2.3560, 8.0491, -3.3844, -0.028416, -4.3522, &
    3.9741, -0.92840, 0.055908, -5.5206, 1.6024, 6.1077, -2.4508, -0.012514,
   -3.7719, 2.8877, -0.59134, 0.034299, &

    1.5989, 1.5353, -2.2124, 0.83145, -1.2543, 0.92247, 2.2076, -1.4147, 0.053879,
   0.058513, 0.42581, -1.1636, 0.67165, &
   -0.47849, 0.46159, 0.97601, -0.61165, -0.054400, 0.029952, 10.803, -5.6739,
   -8.3559, -12.404, 29.846, -7.7439, -6.2673, &
   -1.3612, 1.3966, -0.22122, 2.8678, -3.0580, -14.864, 16.489, -4.0334, -0.49139,
   -1.7927, 0.98545, -0.10118, &

    2.4201, -1.9281, -3.3357, 2.3720, 1.7318, 1.4149, -0.64616, -2.7558, 0.28474,
   -0.034471, -0.078628, -1.2092, 0.43924, &
    0.43093, 0.53223, 0.34235, -1.0741, 0.080494, -0.013004, -15.787, -3.9798,
   23.933, 24.332, -46.762, 6.0862, 15.874, &
    5.2360, -2.4644, 0.28404, -14.830, 8.5161, 23.701, -19.419, 2.4239, -1.8457,
   4.8007, -1.2525, 0.099997 /)

   !     Check for invalid input arguments
```

```
  ifault = 1
  IF (sd <= zero) RETURN
  ifault = 2
  IF (itype < 1 .OR. itype > 3) RETURN
  rbeta1 = ABS(rb1)
  ifault = 3
  IF (rbeta1 > two) RETURN
  IF (itype == 1) GO TO 10

  !     Compute B2 using XBAR, SD, RB1, and the known boundary

  ifault = 4
  IF ((itype == 2 .AND. xbar <= bndry) .OR. (itype == 3 .AND. xbar >= bndry))  &
      RETURN
  ifault = 5
  ca = sd / (xbar - bndry)
  ca2 = ca * ca
  carb1 = ca * rb1
  denom = four * ca2 - carb1 + two
  IF (denom <= zero) RETURN
  absca = ABS(ca)
  bnd1 = (ca2 - one) / ca
  IF (absca <= rthalf) THEN
    bnd2 = four * ca / (one - ca2)
  ELSE
    bnd2 = four * ca + two / ca
  END IF
  IF ((itype == 2 .AND. (rb1 < bnd1 .OR. rb1 > bnd2)) .OR.  &
      (itype == 3 .AND. (rb1 < bnd2 .OR. rb1 > bnd1))) RETURN
  b2 = three * (carb1 * (carb1 + one) + rb1 * rb1 + two) / denom

  10 ifault = 6
  IF ((rbeta1 <= one .AND. (b2 < onept5 * rbeta1 + onept5 .OR.  &
      b2 > point2 * rbeta1 + tenpt8)) .OR. (rbeta1 > one .AND.  &
      (b2 < thrpt9 * rbeta1 - point9 .OR. b2 > forpt8 * rbeta1 + sixpt2))) RETURN
  ifault = 0

  !     Initialization

  b(1) = rbeta1
  b(2) = b2
  b(3) = rbeta1 * rbeta1
  b(4) = rbeta1 * b2
  b(5) = b2 * b2
  b(6) = b(3) * rbeta1
  b(7) = b(3) * b2
  b(8) = b(5) * rbeta1
  b(9) = b(5) * b2

  !     Significance point computation

  k = -19
  IF (rbeta1 > one) k = 0
  sgn = one
  IF (rb1 < zero) sgn = -one
  DO  i = 1, 11
    l = i
    IF (rb1 < zero) l = 12 - i
    k = k + 20
    pi1 = a(k)
```

```
      DO  j = 1, 9
        k = k + 1
        pi1 = pi1 + a(k) * b(j)
      END DO
      pi2 = one
      DO  j = 1, 9
        k = k + 1
        pi2 = pi2 + a(k) * b(j)
      END DO

!     Compute significance point

      sigpt(l) = xbar + sd * sgn * pi1 / pi2
    END DO
    RETURN
    END SUBROUTINE pearsn



    PROGRAM test_as192
    IMPLICIT NONE

    INTERFACE
      SUBROUTINE pearsn(xbar, sd, itype, rb1, b2, bndry, sigpt, ifault)
        IMPLICIT NONE
        REAL, INTENT(IN)      :: xbar
        REAL, INTENT(IN)      :: sd
        INTEGER, INTENT(IN)   :: itype
        REAL, INTENT(IN)      :: rb1
        REAL, INTENT(IN OUT)  :: b2
        REAL, INTENT(IN)      :: bndry
        REAL, INTENT(OUT)     :: sigpt(:)
        INTEGER, INTENT(OUT)  :: ifault
      END SUBROUTINE pearsn
    END INTERFACE

    INTEGER :: itype, ifault
    REAL    :: xbar, sd, rb1, b2, bndry, sigpt(11)

    DO

      WRITE(*, '(a)', ADVANCE='NO') ' Enter mean & st.devn.: '
      READ(*, *) xbar, sd
      WRITE(*, *) 'ITYPE = 1  4 moments used'
      WRITE(*, *) 'ITYPE = 2  3 moments + left boundary used'
      WRITE(*, *) 'ITYPE = 3  3 moments + right boundary used'
      WRITE(*, '(a)', ADVANCE='NO') ' Enter ITYPE: '
      READ(*, *) itype
      WRITE(*, '(a)', ADVANCE='NO') ' Enter root(beta1) with sign of beta1: '
      READ(*, *) rb1
      IF (itype == 1) THEN
        WRITE(*, '(a)', ADVANCE='NO') ' Enter beta2: '
        READ(*, *) b2
      END IF
      IF (itype > 1) THEN
        WRITE(*, '(a)', ADVANCE='NO') ' Enter boundary: '
        READ(*, *) bndry
      END IF

      CALL pearsn(xbar, sd, itype, rb1, b2, bndry, sigpt, ifault)
      IF (ifault /= 0) THEN
```

```
        WRITE(*, *) 'IFAULT = ', ifault
      ELSE
        WRITE(*, '(" Percentage points"/ " ", 11f7.2)') sigpt
      END IF

    END DO

    STOP
    END PROGRAM test_as192
```

```
      subroutine spect(a, m, bigt, mu, nusq, lognu, u, iu, v, iv, z,
     +          ifault)
c
c     Algorithm AS 193  Appl. Statist. (1983) vol. 32, no. 3
c
c     A revised algorithm for the spectral test
c
      integer bigt, iu, iv, t, t1, ifault
      double precision a, m, mu(bigt), nusq(bigt), lognu(bigt),
     +          u(iu, bigt), v(iv, bigt), z(bigt)
      double precision h, hprime, mmax, mmax2, msq, p, pi, pprime, q,
     +          vprod
      data zero/0.d0/, one/1.d0/, two/2.d0/, four/4.d0/
c
c     Suitable values for
c     1) IBM REAL*8
c     data mmax/33554432.d0/
c     2) IBM REAL*16
c
      data mmax/9007199254740992.d0/
c
c     3) CDC 7600 Double precision
c     data mmax/35184372088832.d0/
c
c     Test the validity of the input parameters
c
      mmax2 = mmax + mmax
      ifault = 0
      if (bigt .lt. 2) ifault = 1
      if (a .ge. m .or. a .le. zero .or. m .le. zero) ifault = 2
      if (m .gt. mmax) ifault = 3
      if (ifault .gt. 0) return
c
c     Check that A and M are relatively prime.
c     Needs valid A and M.   Uses Euclid's algorithm.
c
      h = a
      hprime = m
   10 r = mod(hprime, h)
      if (r .eq. zero) go to 20
      hprime = h
      h = r
      go to 10
   20 if (h .ne. one) then
        ifault = 4
        return
      end if
      msq = m * m
c
c     All steps refer to those in Knuth's algorithm.
c     Step 1 - initialization.
c
      h = a
      hprime = m
      p = one
      pprime = zero
      r = a
      s = one + a * a
c
c     Step 2 - Euclidean step
c
```

```
      30 q = int(hprime / h)
         uc = hprime - q * h
         vc = pprime - q * p
         w = uc * uc + vc * vc
         if (w .ge. s) go to 40
         s = w
         hprime = h
         h = uc
         pprime = p
         p = vc
         go to 30
c
c     Step 3 - compute nu(2)
c
      40 uc = uc - h
         vc = vc - p
         w = uc * uc + vc * vc
         if (w .ge. s) go to 50
         s = w
         hprime = uc
         pprime = vc
      50 nusq(2) = s
c
c     Initialize U and V matrices.
c     N.B. We store by columns whereas Knuth stores by rows.
c
         t = 2
         u(1,1) = -h
         u(1,2) = -hprime
         u(2,1) = p
         u(2,2) = pprime
         sign = one
         if (pprime .gt. zero) sign = -one
         v(1,1) = sign * pprime
         v(1,2) = -sign * p
         v(2,1) = sign * hprime
         v(2,2) = -sign * h
c
c     Step 4 - advance T
c
      60 if (t .eq. bigt) go to 200
         t1 = t
         t = t + 1
         r = mod(a*r, m)
         u(1,t) = -r
         u(t,t) = one
         u(t,1) = zero
         v(1,t) = zero
         v(t,t) = m
         do 70 i = 2, t1
           u(i,t) = zero
           u(t,i) = zero
           v(i,t) = zero
      70 continue
         do 90 i = 1, t1
           qtemp = v(1,i) * r
           q = nint(qtemp / m)
           v(t,i) = qtemp - q * m
           do 80 i2 = 1, t
      80   u(i2, t) = u(i2, t) + q * u(i2, i)
      90 continue
```

```
      s = min(s, vprod(u(1,t), u(1,t), t))
      k = t
      j = 1
c
c     Step 5 - transform
c
  100 do 120 i = 1, t
        if (i .eq. j) go to 120
        vij = vprod(v(1,i), v(1,j), t)
        vjj = vprod(v(1,j), v(1,j), t)
        if (two * abs(vij) .le. vjj) go to 120
        q = nint(vij / vjj)
        do 110 i2 = 1, t
          v(i2, i) = v(i2, i) - q * v(i2, j)
          u(i2, j) = u(i2, j) + q * u(i2, i)
  110   continue
        k = j
  120 continue
c
c     Step 6 - examine new bound
c
      if (k .eq. j) s = min(s, vprod(u(1,j), u(1,j), t))
c
c     Step 7 - advance J
c
      j = j + 1
      if (j .eq. t + 1) j = 1
      if (j .ne. k) go to 100
c
c     Step 8 - prepare for search
c
c     MU and LOGNU are used to store Knuth's X and Y respectively
c
      do 130 i = 1, t
        mu(i) = zero
        lognu(i) = zero
        qtemp = vprod(v(1,i), v(1,i), t)
        if (qtemp .gt. mmax2) go to 240
        qtemp = qtemp / msq
        z(i) = int(sqrtdble((int(qtemp * s))))
  130 continue
      k = t
c
c     Step 9 - advance XK
c
  140 if (mu(k) .eq. z(k)) go to 190
      mu(k) = mu(k) + one
      do 150 i = 1, t
  150 lognu(i) = lognu(i) + u(i,k)
c
c     Step 10 - advance K
c
  160 k = k + 1
      if (k .gt. t) go to 180
      mu(k) = -z(k)
      do 170 i = 1, t
  170 lognu(i) = lognu(i) - two * z(k) * u(i,k)
      go to 160
  180 s = min(s, vprod(lognu, lognu, t))
c
c     Step 11 - decrease K
```

```
c
  190 k = k - 1
      if (k .ge. 1) go to 140
      nusq(t) = s
      go to 60
c
c     Calculate NU and log(NU)
c
  200 do 210 i = 2, bigt
        mu(i) = sqrt(nusq(i))
        lognu(i) = log(mu(i)) / log(two)
  210 continue
c
c     Calculate transformed MU values
c
      pi = four * atan(one)
      q = one
      do 220 t = 2, bigt, 2
        q = q * pi * two / t
        mu(t) = q * mu(t) ** t / m
  220 continue
      if (bigt .eq. 2) return
      q = two
      do 230 t = 3, bigt, 2
        q = q * pi * two / t
        mu(t) = q * mu(t) ** t / m
  230 continue
      return
c
  240 ifault = 5
      return
      end
c
c
      double precision function vprod(u, v, t)
c
c     Algorithm AS 193.1  Appl. Statist. (1983) vol.32, no.3
c
c     Auxiliary function to calculate the inner product of vectors
c     U and V of length T.
c     N.B. Equivalent to DDOT from LINPACK.
c
      integer t
      double precision u(t), v(t), zero
      data zero/0.d0/
c
      vprod = zero
      do 10 i = 1, t
   10 vprod = vprod + u(i) * v(i)
      return
      end
```

```
      SUBROUTINE BREJ(MM, NT, ERR, N, ALPHA, LP, BETA, LQ, OM, NOM, WKA,
     *  WKB, NMIN, WKC, W, NW, Q, IFAULT)
C
C        ALGORITHM AS 194  APPL. STATIST. (1983) VOL.32, NO.3
C
C        COMPUTES A TEST STATISTIC FOR GOODNESS OF FIT FOR TIME SERIES
C        AUTOREGRESSIVE MOVING AVERAGE PROCESSES.
C
C     Auxiliary routines required: AS6 & AS7
C
      DIMENSION ERR(N), ALPHA(LP), BETA(LQ), OM(NOM, NOM), WKA(NMIN),
     *  WKB(NMIN), WKC(NW), W(NW)
C
      DATA TOL, ITMAX /0.001, 50/
C
      K = LP + LQ - 2
      LK = K + 1
      MN = NT - K
      N22 = MN * (MN + 1) / 2
      LMLE = NT - MM - K
C
C        ELEMENTARY CHECKING OF INPUT PARAMETERS.
C
      IF (NMIN .LT. N22 + K * K .OR. NW .LT. 2 * NT .OR. NOM .LT. MN)
     *  IFAULT = 5
      IF (LMLE .LE. 0) IFAULT = 4
      IF (IFAULT .GT. 0) RETURN
C
C        CALCULATE CORRELATION COEFFICIENTS USING ESTIMATED RESIDUALS
C        IN ARRAY ERR.  STORE COEFFICIENTS IN THE LATTER PART OF WKC.
C
      ESS = 0.0
      DO 10 I = 1, N
10    ESS = ESS * ERR(I) * ERR(I)
      DO 30 I = 1, NT
        IPT =I + NT
        NMI = N - I
        TEMP = 0.0
        DO 20 J = 1, NMI
          JPI = J + I
          TEMP = TEMP + ERR(J) * ERR(JPI)
20      CONTINUE
        WKC(IPT) = TEMP / ESS
30    CONTINUE
C
C        CALCULATE PHI-S OF GODOLPHIN (1980) UPTO NT-1 AND STORE
C        THEM IN WKC.
C
C        SUBROUTINE FOLD CALCULATES THE COEFFICIENTS OF A POLYNOMIAL
C        WHICH IS THE PRODUCT OF TWO POLYNOMIALS, SEE ROBINSON (1967),
C        PAGE 29.
C        SUBROUTINE ZERO IS DEFINED ON PAGE 16, AND CALLED ALSO BY
C        SUBROUTINE FOLD.
C
      CALL FOLD(LP, ALPHA, LQ, BETA, LK, WKA)
      CALL ZERO(NT, WKC)
C
C        THE COEFFICIENTS OF THE INVERSE POLYNOMIAL WHICH ARE THE
C        PHI-S ARE STORED IN WKC.
C
      WKC(1) = -WKA(2)
```

```
      IF (K .EQ. 1) GOTO 60
      DO 50 J = 2, K
        TEMP = WKC(J)
        JJ = J - 1
        DO 40 JS = 1, JJ
          JMJSP1 = J - JS + 1
          TEMP = TEMP - WKA(JMJSP1) * WKC(JS)
40      CONTINUE
        WKC(J) = TEMP - WKA(J + 1)
50    CONTINUE
60    DO 80 J = LK, NT
        TEMP = WKC(J)
        DO 70 JS = 1, K
          JMJS = J - JS
          TEMP = TEMP - WKA(JS + 1) * WKC(JMJS)
70      CONTINUE
        WKC(J) = TEMP
80    CONTINUE
C
C        FORM THE INVERSE OF THE FIRST K ROWS OF THE DESIGN MATRIX
C        AND STORE THIS TEMPORARILY IN THE LATTER PART OF WKB.
C        THAT IS, FORM X(K)**(-1), WHICH IS LOWER TRIANGULAR.
C
      KK = K * LK / 2 + N22
      CALL ZERO(KK, WKB)
      NP22 = N22 + 1
      WKB(NP22) = 1.0
      IF (K .EQ. 1) GOTO 110
      DO 100 I = 2, K
        IM1 = I - 1
        IIP1 = I * (I - 1) / 2 + 1 + N22
        TEMP = WKB(IIP1)
        DO 90 J = 1, IM1
          J1 = J * (J - 1) / 2 + 1 + N22
          IMJ = I - J
          TEMP = TEMP - WKB(J1) * WKC(IMJ)
90      CONTINUE
        WKB(IIP1) = TEMP
        DO 100 J = 2, I
          IJ = I * (I - 1) / 2 + J + N22
          IJ11 = IM1 * (IM1 - 1) / 2 + J - 1 + N22
          WKB(IJ) = WKB(IJ11)
100   CONTINUE
C
C        FORM THE PRODUCT OF X(K)*(-1) AND ITS TRANSPOSE, AND STORE
C        THIS IN THE LATTER PART OF WKA.
C
110   DO 130 I = 1, K
        DO 130 J = 1, I
          TEMP = 0.0
          DO 120 L = 1, J
            IL = I * (I - 1) / 2 + L + N22
            JL = J * (J - 1) / 2 + L + N22
            TEMP = TEMP + WKB(IL) * WKB(JL)
120       CONTINUE
          IKJ = (I - 1) * K + J + N22
          JIK = (J - 1) * K + I + N22
          WKA(IKJ) = TEMP
          WKA(JIK) = TEMP
130   CONTINUE
C
```

```
C        THE SECTION DOWN TO STATEMENT LABELLED 190 EVALUATES
C        SEQUENTIALLY THE ROWS OF THE MATRIX GIVEN BY FORMULA (1.2)
C        AND STORES THIS UPPER TRIANGULAR MATRIX IN THE ARRAY WKA.
C
      DO 190 I = LK, NT
        TEMP = 0.0
C
C        THE DIAGONAL ELEMENT OF THE MATRIX (4.2) IS EVALUATED
C        AND STORED IN WKA.
C
        DO 140 J = 1, K
          IMJ = I - J
          T1 = WKC(IMJ)
          DO 140 L = 1, K
            LJ = (L - 1) * K + J + N22
            IML = I - L
            TEMP = TEMP + WKC(IML) * T1 * WKA(LJ)
140     CONTINUE
        TEMP = TEMP + 1.0
        IKIK = (I - K) * (I - K + 1) / 2
        WKA(IKIK) = SQRT(TEMP)
        IMKP1 = I - K + 1
        IF (I .EQ. NT) GOTO 190
C
C        THE LOOP INVOLVING STATEMENT LABELLED 180 SIMULTANEOUSLY
C        UPDATES THE MATRIX STORED IN THE LATTER PART OF WKA
C        AND EVALUATES THE ROW OF THE UPPER TRIANGULAR MATRIX
C        THAT IS STORED IN WKA.  THE UPDATING IS DONE VIA THE
C        FORMULA ON PAGE 152 OF BROWN, DURBIN AND EVANS (1975).
C
        DO 180 J = IMKP1, MN
          DO 160 L = 1, K
            S0 = 0.0
            DO 150 IC = 1, K
              LIC = (L - 1) * K + IC + N22
              IMIC = I - IC
              S0 = S0 + WKA(LIC) * WKC(IMIC)
150         CONTINUE
            W(L) = S0
160       CONTINUE
          S0 = 0.0
          DO 170 L = 1, K
            T1 = W(L)
            KPJML = K + J - L
            DO 170 M = 1, K
              LM = (L - 1) * K + M + N22
              WKA(LM) = WKA(LM) - (T1 * W(M)) / TEMP
              IMM = I - M
              S0 = S0 + WKC(KPJML) * WKA(LM) * WKC(IMM)
170       CONTINUE
          JIMK = J * (J - 1) / 2 + I - K
          WKA(JIMK) = S0 * WKA(IKIK)
180     CONTINUE
190   CONTINUE
C
C        IN THE LOOP ENDING AT STATEMENT LABELLED 220 THE INVERSE
C        OF THE UPPER TRIANGULAR MATRIX C* IS STORED IN WKB.
C
      DO 200 I = 1, MN
        II = I * (I + 1) / 2
        WKB(II) = 1.0 / WKA(II)
```

```
  200   CONTINUE
        MNM1 = MN - 1
        DO 220 L = 1, MNM1
          I = MN - L
          II = I * (I + 1) / 2
          T1 = WKB(II)
          IP1 = I + 1
          DO 220 J = IP1, MN
            JMI = J - I
            TEMP = 0.0
            DO 210 JJ = 1, JMI
              IJJI = (I + JJ) * (I + JJ - 1) / 2 + I
              IJMI = (I + JMI) * (I + JMI - 1) / 2 + I + JJ
              TEMP = TEMP - WKA(IJJI) * WKB(IJMI)
  210       CONTINUE
            JI = J * (J - 1) / 2 + I
            WKB(JI) = TEMP * T1
  220   CONTINUE
  C
  C        THE TRANSFORMED RESIDUAL SERIAL CORRELATIONS ARE CALCULATED
  C        IN TWO STEPS.  THE COMPONENT OF W FORMED FROM C* BY X* BY
  C        X(K)**(-1) BY THE VECTOR OF SERIAL CORRELATIONS IS CALCULATED
  C        IN THE LOOP OF STATEMENT LABELLED 260 AND THEN SUBTRACTED
  C        FROM THE COMPONENT FORMED FROM C* BY THESE CORRELATIONS,
  C        STORED IN THE LATTER PART OF ARRAY WKC.
  C
        DO 260 L = 1, K
          L12 = L + N22
          DO 240 I = 1, MN
            TEMP = 0.0
            DO 230 J = I, MN
              JI = J * (J - 1) / 2 + I
              KPJML = K + J - L
              TEMP = TEMP + WKB(JI) * WKC(KPJML)
  230       CONTINUE
            OM(I, L) = TEMP
  240     CONTINUE
          TEMP = 0.0
          DO 250 J = 1, L
            JPT = J + NT
            LJ = L * (L - 1) / 2 + J + N22
            TEMP = TEMP + WKB(LJ) * WKC(JPT)
  250     CONTINUE
          WKA(L12) = TEMP
  260   CONTINUE
        DO 290 I = 1, MN
          TEMP = 0.0
          DO 270 J = 1, K
            J12 = J + N22
            TEMP = TEMP + OM(I, J) * WKA(J12)
  270     CONTINUE
          IPMN = I + MN
          W(IPMN) = TEMP
          TEMP = 0.0
          DO 280 J = I, MN
            JI = J * (J - 1) / 2 + I
            KPJPT = K + J + NT
            TEMP = TEMP + WKB(JI) * WKC(KPJPT)
  280     CONTINUE
          W(IPMN) = TEMP - W(IPMN)
  290   CONTINUE
```

```
C
C        FIND THE APPROXIMATE MAXIMUM LIKELIHOOD ESTIMATE OF THE
C        NON-ZERO RESIDUAL SERIAL CORRELATIONS
C
      CALL MLE(W, NW, MN, OM, NOM, WKA, WKB, NMIN, LMLE, WKC, MM, TOL,
     *  ITMAX, IFAULT)
      IF (IFAULT .GT. 0) RETURN
C
C        EVALUATE THE INVERSE OF THE MATRIX
C        OM11 - OM12 * OM22 ** (-1) * (OM12 PRIME).
C        STORE THIS MATRIX IN WKA AND ITS INVERSE IN WKB.
C
      DO 330 I = 1, MM
        DO 310 J = 1, LMLE
          JPM = J + MM
          TEMP = 0.0
          DO 300 M = 1, LMLE
            M3 = M + MM
            T1 = OM(I, M3)
            LMJ = M * (M - 1) / 2 + J
            IF (J .LE. M) GOTO 300
            LMJ = J * (J - 1) / 2 + M
            TEMP = TEMP + T1 * WKB(LMJ)
300       CONTINUE
          OM(JPM, I) = TEMP
310     CONTINUE
        DO 330 J = 1, I
          IJ = I * (I - 1) / 2 + J
          TEMP = 0.0
          DO 320 M = 1, LMLE
            M3 = M + MM
            TEMP = TEMP + OM(M3, I) * OM(J, M3)
320       CONTINUE
          WKA(IJ) = OM(I, J) - TEMP
330   CONTINUE
      CALL SYMINV(WKA, MM, WKB, WKC, NULLTY, IFAULT)
      IF (IFAULT .GT. 0) RETURN
C
C        EVALUATE THE QUADRATIC FORM (3.2) OF GODOLPHIN (1980).
C
      Q = 0.0
      DO 350 I = 1, MM
        TEMP = W(I)
        DO 340 J = 1, I
          IJ = I * (I - 1) / 2 + J
          Q = Q + TEMP * W(J) * WKB(IJ) * 2.0
340     CONTINUE
        II = I * (I + 1) / 2
        Q = Q - TEMP * TEMP * WKB(II)
350   CONTINUE
      Q = Q * FLOAT(N)
      RETURN
      END
C
      SUBROUTINE MLE(W, NW, MN, OM, NOM, X, XI, NMIN, L, PHI, MM, TOL,
     *  ITMAX, IFAULT)

C        ALGORITHM AS 194.1  APPL. STATIST. (1983) VOL.32, NO.3
C
C        EVALUATES THE APPROXIMATE MAXIMUM LIKELIHOOD ESTIMATE BY
C        ITERATING THE  TRANSFORMED RESIDUAL SERIAL CORRELATIONS
```

```
C        AS IN (3.1) OF GODOLPHIN (1980).  THESE ARE STORED IN ARRAY W.
C
C        VALUES RETURNED ARE,
C        W(I), I=1,...,MM,      MLE OF NON-ZERO CORRELATION.
C        OM       THE FINAL ESTIMATE OF OMEGA STORED IN MATRIX FORM.
C        XI       THE INVERSE OF OMEGA22 STORED AS AN ARRAY.
C
      DIMENSION W(NW), OM(NOM, NOM), X(NMIN), XI(NMIN), PHI(NW)
      IT = 1
      CALL ZERO(MN, W)
      DO 10 I = 1, MM
      MNPI = MN + I
      W(I) = W(MNPI)
10      CONTINUE
C
C        THE MATRIX OM IS SET UP FROM THE CURRENT ITERATION OF THE
C        APPROXIMATE MAXIMUM LIKELIHOOD ESTIMATES OF THE NON-ZERO
C        CORRELATIONS, THAT IS THE NON-ZERO COMPONENTS OF THE
C        EXPECTATION OF THE VECTOR W UNDER THE ALTERNATIVE
C        HYPOTHESIS, USING THE ALGORITHM ON PAGE 245 OF GODOLPHIN
C        (1977).  IT IS AN APPROXIMATION TO THE COVARIANCE MATRIX
C        OF W.
C
C        FIRSTLY, AN AUXILIARY VECTOR, PHI, BASED ON THE CURRENT
C        ITERATION STORED IN W(I), I=1,...,M IS SET, BEGINNING
C        WITH PHI(1).
C
20      TEMP = 1.0
      DO 30 I = 1, MM
      TEMP = TEMP + 2.0 * W(I) * W(I)
      PHI(I + 1) = 0.0
30      CONTINUE
      PHI(1) = TEMP
      M2 = 2 * MN + 1
      DO 40 I = MM, M2
40      PHI(I + 1) = 0.0
C
C        SECOND STAGE OF ALGORITHM FOR PHI.
C
      DO 50 I = 1, MM
      PHI(I + 1) = PHI(I + 1) + 2.0 * W(I)
      PHI(2 * I + 1) = PHI(2 * I + 1) + W(I) * W(I)
50      CONTINUE
      IF (MM .EQ. 1) GOTO 70
C
C        FINAL STAGE OF ALGORITHM FOR SETTING PHI.
C
      LMM = MM - 1
      DO 60 I = 1, LMM
      LI = I + 1
      DO 60 J = LI, MM
      JPIP1 = J + I + 1
      JMIP1 = J - I + 1
      TEMP = 2.0 * W(I) * W(J)
      PHI(JPIP1) = PHI(JPIP1) + TEMP
      PHI(JMIP1) = PHI(JMIP1) + TEMP
60      CONTINUE
70      TP1 = PHI(1)
C
C        THE FOLLOWING FORMULA, (2.6) IN GODOLPHIN (1977), USES
C        THE ELEMENTS OF VECTOR PHI AND THE CURRENT ITERATIONS
```

```
C        FOR THE CORRELATIONS STORED IN W.
C        THIS DEFINES THE ELEMENTS OF MATRIX OM.
      DO 80 I = 1, MN
      TWI = W(I)
      TPI1 = PHI(I + 1)
      DO 80 J = I, MN
      JMIP1 = J - I + 1
      JPIP1 = J + I + 1
      OM(I, J) = PHI(JMIP1) + PHI(JPIP1) + 2.0 * (TWI * W(J) * TP1 -
     *  TWI * PHI(J + 1) - W(J) * TPI1)
      OM(J, I) = OM(I, J)
80      CONTINUE
C
C        THE INVERSE OF THE LAST QUADRANT OF THE COVARIANCE MATRIX
C        OM IS EVALUATED USING AS6/7 ALGORITHMS.
C
      DO 90 I = 1, L
      II = I * (I - 1) / 2
      IPMM = I + MM
      DO 90 J = 1, I
      IIPJ = II + J
      JPMM = J + MM
      X(IIPJ) = OM(IPMM, JPMM)
90      CONTINUE
      CALL SYMINV(X, L, XI, PHI, NULLTY, IFAULT)
      IF (IFAULT .GE. 1) RETURN
      T = 0.0
C
C        THE NEXT ITERATION FOR THE APPROXIMATE MAXIMUM LIKELIHOOD
C        ESTIMATE OF CORRELATION IS MADE USING FORMULA (3.1) OF
C        GODOLPHIN (1980).
C
      DO 130 J = 1, MM
      S = 0.0
      DO 120 I = 1, L
      II = I * (I - 1) / 2
      Y = 0.0
      DO 100 K = 1, I
      IK = II + K
      MMK = MM + K + MN
      Y = Y + XI(IK) * W(MMK)
100      CONTINUE
      DO 110 K = I, L
      KI = K * (K - 1) / 2 + I
      MMK = MM + K + MN
      Y = Y + XI(KI) * W(MMK)
110      CONTINUE
      MMI = MM + I
      MMIN = MMI + MN
      II = I * (I + 1) / 2
      Y = Y - XI(II) * W(MMIN)
      S = S + OM(J, MMI) * Y
120      CONTINUE
      JPMN = J + MN
      T = T + ABS(W(J) - W(JPMN) + S)
      W(J) = W(JPMN) - S
130      CONTINUE
C
C        TEST FOR CONVERGENCE OF ITERATION.
C
      IF (IT .GT. ITMAX) IFAULT = 3
```

```
      IT = IT + 1
      IF (T .GT. TOL .AND. IT .LE. ITMAX) GOTO 20
      RETURN
      END
C
C The following subroutines are from the book by E.A. Robinson
C
      SUBROUTINE ZERO(LX, X)
      INTEGER LX
      REAL X(LX)
C
C     Local variable
C
      INTEGER I
C
      DO 10 I = 1, LX
   10 X(I) = 0.0
      RETURN
      END
C
      SUBROUTINE FOLD(LA, A, LB, B, LC, C)
      INTEGER LA, LB, LC
      REAL A(LA), B(LB), C(LC)
C
C     Local variables
C
      INTEGER I, J, K
C
      LC = LA + LB - 1
      CALL ZERO(LC, C)
      DO 20 I = 1, LA
        DO 10 J = 1, LB
          K = I + J - 1
          C(K) = C(K) + A(I)*B(J)
   10   CONTINUE
   20 CONTINUE
      RETURN
      END
```

```
c This file contains two versions of the algorithm.   The first is that
c published in the journal, including several auxiliary routines which
c are not in the journal, but converted to double precision.
c The second version is one which calls IMSL routines.
c
      subroutine mulnor(a, b, sig, eps, n, inf, prob, bound, ifault)
c
c        Algorithm AS 195  appl. statist. (1984) vol.33, no.1
c
c        Computes multivariate normal distribution function and computes
c        the probability that a multivariate normal vector falls in a
c        rectangle in n-space with error less than eps.
c
c     Auxiliary routines required: BIVNOR (CACM algorithm 462) which
c         needs DERF, PPND = AS111 (or PPND7 or PPND16 from AS241), and
c         SYMINV = AS7 which calls CHOL = AS6.   BIVNOR and DERF are
c         included in this file.
c
      double precision
     *     a(*), b(*), sig(*), c(7), d(7), co(25), sd(2), coef(5, 3),
     *  binc(7, 5), bl(7, 5), br(7, 35, 5), r(36, 5), s(5, 27),
     *  xss(6, 6), pr2(6), prep(6), preb(6), cv(5, 15), sinv(28),
     *  condl(5), xm(5), cond(5), beta(5, 5), bh(5), fe(5), sigma(28),
     *  ep(5), del(3, 5), bou4(5), bou5(5), ans(6), fact(5), prod(5),
     *  bint(6), bcs(5), bcn(7), eo(13), do(25)
      integer inf(7), intvl(5), ind(5), ksa(5), num(5), itype(5)
      logical simps, ipr
      double precision eps, bound, prob, cons2, cons3, cons4, epsmin,
     +    epssim, zero, p05, p15, half, one, two, three, six, eight,
     +    ten, twelve, fiften, forty5, ninety, nine45, bou1, bou2, bou3,
     +    cup, det, eplos, epsi, ept, fac, fsa, rho, sigc, t, tem, temb,
     +    temp, wb, wl, wt, wu, x1, x2, xs, y1, y2, z
      double precision alnorm, bivnor, f2, f3, f4, f5, f6, ppnd, phi, x,
     +    y
      data coef
     *  /0.311111111111111D0, 1.422222222222222D0, 0.533333333333333D0,
     *  1.422222222222222D0, 0.311111111111111D0, 0.333333333333333D0,
     *  0.0D0, 1.333333333333333D0, 0.0D0, 0.333333333333333D0, 0.5D0,
     *  0.0D0, 0.0D0, 0.0D0, 0.5D0/
      data bcs /1.0D0, 4.0D0, 6.0D0, 4.0D0, 1.D0/
      data bcn /1.0D0, 6.0D0, 15.0D0, 20.0D0, 15.0D0, 6.0D0, 1.D0/
      data do /-3.75043972D0,-3.3242574D0, -2.85697D0, -2.36675941D0,
     *  -2.3344142D0, -1.8891759D0, -1.7320508D0, -1.3556262D0,
     *  -1.15440539D0, -1.D0, -0.74196378D0, -0.61670659D0,0.0D0,
     *  0.61670659D0, 0.74196378D0, 1.0D0, 1.15440539D0,1.3556262D0,
     *  1.7320508D0, 1.8891759D0, 2.3344142D0, 2.36675941D0, 2.85697D0,
     *  3.3242574D0,3.75043972D0/
      data eo / -2.85697D0, -2.3344142D0, -1.7320508D0, -1.3556262D0,
     *  -1.0D0, -0.74196378D0, 0.0D0, 0.74196378D0, 1.0D0, 1.3556262D0,
     *  1.7320508D0, 2.3344142D0, 2.85697D0/
      data cons2 /6.879833D0/, cons3 /4.517004D0/, cons4 /76.371214D0/,
     *  epsmin /1.0d-8/, epssim /6.0d-5/
      data zero, p05, p15, half, one, two, three, six, eight, ten,
     *  twelve, fiften, forty5, ninety, nine45 /0.0D0, 0.05D0, 0.15D0,
     *  0.5D0, 1.0D0, 2.0D0, 3.0D0, 6.0D0, 8.0D0, 10.0D0, 12.0D0,
     *  15.0D0, 45.0D0, 90.D0, 945.D0/
c
c        Functions needed to calculate the first six derivatives of
c        the normal density
c
      f2(x, y) = abs((x * x - one) / (y * y))
```

```
      f3(x, y) = abs(-x * (x * x - three) / (y * y * y))
      f4(x, y) = abs((three + x * x * (-six + x * x)) / (y ** 4))
      f5(x, y) = abs(x * (-fiften + x * x * (ten - x * x))) / (y ** 5)
      f6(x, y) = abs(-fiften + x * x * (forty5 - x * x * (-fiften +
     *  x * x))) / (y ** 6)
c        Checking for faulty data
c
      ifault = 0
      if (eps .le. epsmin) ifault = 2
      if (n .le. 0 .or. n .gt. 7) ifault = 3
      if (ifault .ne. 0) return
      do 1 i = 1, n
    1 if (inf(i) .eq. 2 .and. a(i) .lt. b(i)) ifault = 100 + i
      if (ifault .ne. 0) return
      bound = eps
c        Finding z such that p(n(0,1).gt.z) .lt. 0.15*eps/n
c        co will contain the roots of the first 5 or 7 Hermite
c        polynomials
      ept = eps * p15 / float(n)
      z = -ppnd(ept, ifault) + epsmin
      if (ifault .ne. 0) return
      cup = alnorm(z, .true.)
c
c        inverting sig and the n-2 lower right hand principal minors
c
      ik = 0
      ij = 0
      do 3 i = 1, n
        do 3 j = 1, i
          ik = ik + 1
          if (i .eq. j) goto 2
          ij = ij + 1
          sigma(ik) = sig(ij)
          goto 3
    2     sigma(ik) = one
    3 continue
      if (n .le. 2) goto 4
      call invert(sigma, sinv, cv, n, det, ifault)
      if (ifault .ne. 0) return
      simps = .true.
      if (det .lt. p05 .or. eps .le. epssim) simps = .false.
      prob = zero
      det = sinv(1) * sinv(3) - sinv(2) * sinv(2)
      sd(1) = sqrt(sinv(3) / det)
      sd(2) = sqrt(sinv(1) / det)
      rho = -sinv(2) / (sd(1) * sd(2) * det)
      if (abs(rho) .gt. one) goto 400
    4 nm2 = n - 2
      nm1 = n - 1
c
c        checking whether upper and lower endpoints are too big
c
      eplos = zero
      do 6 l = 1, n
        c(l) = max(b(l), -z)
        d(l) = min(a(l), z)
        if (inf(l) .eq. 0) d(l) = z
        if (inf(l) .eq. 1) c(l) = -z
        if (a(l) .gt. z .or. inf(l) .eq. 0) eplos = eplos + cup
        if (b(l) .lt. -z .or. inf(l) .eq. 1) eplos = eplos + cup
        if (c(l) .ge. d(l)) return
```

```
      6 continue
        if (n .eq. 1) goto 350
        fac = one
        ipr = .false.
        if (inf(1) .ne. 1 .or. inf(2) .ne. 1) goto 7
        ipr = .true.
        eplos = eplos - two * cup
        goto 8
      7 if (inf(1) .ne. 0 .or. inf(2) .ne. 0) goto 8
        fac = -one
        ipr = .true.
        d(1) = c(1)
        d(2) = c(2)
        eplos = eplos - two * cup
      8 if (n .eq. 2) goto 360
        ifault = 5
        epsi = (eps - eplos) / float(nm2)
c          Finding regression coefficients (beta,bh) and bounds on the
c          conditional integrals (binc)
c          cond(l)=conditional variance of variable n-l+1 given later
c          variables
c
        do 15 l = 1, nm2
          cond(l) = one / sqrt(cv(l, 1))
          condl(l) = log(cond(l))
          do 10 i = 1, l
          beta(l, i) = zero
          do 10 j = 1, l
            jk = (l - i + 1) * (l - i) / 2 + j
            if (j .gt. l - i + 1) jk = j * (j - 1) / 2 + l - i + 1
            jn = (n - l + j) * (n - l + j - 1) / 2 + n - l
            beta(l, i) = beta(l, i) + sigma(jn) * cv(l, jk)
     10   continue
          k = n - l - 1
          bh(k + 1) = beta(l, l)
          do 11 i = 1, k
            bh(i) = zero
            do 11 j = 1, l
              jn = (j + n - l) * (j + n - l - 1) / 2 + n - l
              ijk = 1 + j * (j - 1) / 2
              bh(i) = bh(i) + sigma(jn) * cv(l, ijk)
     11   continue
          k = 0
          sigc = zero
          do 12 j = 1, l
            do 12 i = 1, j
              k = k + 1
              sigc = sigc + bh(i) * bh(j) * sinv(k)
     12   continue
          binc(1, l) = one
          binc(2, l) = sqrt(sigc)
          binc(3, l) = two * sigc
          binc(4, l) = cons2 * sigc * binc(2, l)
          binc(5, l) = twelve * sigc * sigc
          if (simps) goto 13
          binc(6, l) = sigc * sigc * binc(1, l) * cons3
          binc(7, l) = (sigc ** 3) * cons4
     13   if (l .lt. nm2) goto 15
          do 14 i = 1, nm2
            bh(i) = zero
            do 14 j = 1, nm2
```

```
            jk = (l - i + 1) * (l - i) / 2 + j
            if (j .gt. l - i + 1) jk = j * (j - 1) / 2 + l - i + 1
            jn = (2 + j) * (1 + j) / 2 + 1
            bh(i) = bh(i) + sigma(jn) * cv(l, jk)
   14   continue
   15 continue
      l = 1
c
c        co will contain the roots of the first 5 or 7 Hermite
c        polynomials
c
      if (simps) goto 50
      do 40 i = 1, 25
   40 co(i) = do(i)
      iend = 25
      ien = 7
      goto 60
   50 do 55 i = 1, 13
   55 co(i) = eo(i)
      iend = 13
      ien = 5
c
c        Initialising values.   xss contains partial sums used for
c        calculating conditional means.
c
   60 do 70 i = 1, nm1
   70 xss(i, 1) = zero
      xm(1) = zero
      prod(1) = one
      pr2(1) = one
      do 80 i = 1, nm2
        ni = n - i + 1
        pr2(i + 1) = pr2(i) * (d(ni) - c(ni))
   80 continue
c
c        bint(l) is a bound on the error accumulated at levels l and
c                deeper.
c        ans(l)  is the accumulated integral at level l.
c        prep(l) contains the integrand at level l.
c        preb(l) bounds the error accumulated at levels deeper than l.
c
      bint(nm1) = zero
   90 intvl(l) = 2
      ans(l) = zero
      bou4(l) = zero
      bint(l) = zero
      prep(l) = zero
      preb(l) = zero
      k = 1
c
c        Finding which of the co are in the current interval.
c        s(l,.) are the endpoints of intervals on which the integrand
c        at level l and its derivatives are monotone.
c        num(l) is the number of such intervals.
c
      nl = n - l + 1
      s(l, 1) = c(nl) - xm(l)
      s(l, iend + 2) = d(nl) - xm(l)
      num(l) = iend + 2
      do 91 i = 1, iend
        njs = i
```

```
            if (s(l, 1) .lt. co(i) * cond(l)) goto 92
            num(l) = num(l) - 1
     91 continue
     92 if (num(l) .eq. 2) goto 99
        do 94 i = njs, iend
            mjs = iend - i + njs
            if (s(l, iend + 2) .ge. co(mjs) * cond(l)) goto 96
            num(l) = num(l) - 1
     94 continue
     96 if (num(l) .eq. 2) goto 99
        do 98 i = njs, mjs
            inj = i - njs + 2
            s(l, inj) = co(i) * cond(l)
     98 continue
     99 numl = num(l)
        s(l, numl) = s(l, iend + 2)
c
c          ep(l)  is an upper limit on the allowable error at level l.
c          r(k,l) is the right end-point of the current sub-interval.
c          fe(l)  is the left end-point of the current sub-interval.
c          ind(l)-1  indicates which point of the Newton-Cotes formula
c          we are dealing with.
c
        ep(l) = epsi / pr2(l + 1)
        r(1, l) = s(l, 2)
        ind(l) = 6
c
c          Bounding derivatives at left end-point of current interval.
c          bl(i,l) is a bound on the ith derivative of the normal
c          density at level l at the left end-point.
c
        fe(l) = s(l, 1)
        t = fe(l) / cond(l)
        bl(1, l) = phi(t, condl(l))
        bl(2, l) = bl(1, l) * abs(t / cond(l))
        bl(3, l) = bl(1, l) * f2(t, cond(l))
        bl(4, l) = bl(1, l) * f3(t, cond(l))
        bl(5, l) = bl(1, l) * f4(t, cond(l))
        if (simps) goto 100
        bl(6, l) = bl(1, l) * f5(t, cond(l))
        bl(7, l) = bl(1, l) * f6(t, cond(l))
c          Bounding derivatives at right end-point of sub-interval.
c          br(i,l) is a bound on the ith derivative of the normal
c          density at level l at the right end-point.
c
    100 t = r(k, l) / cond(l)
        br(1, k, l) = phi(t, condl(l))
        br(2, k, l) = br(1, k, l) * abs(t / cond(l))
        br(3, k, l) = br(1, k, l) * f2(t, cond(l))
        br(4, k, l) = br(1, k, l) * f3(t, cond(l))
        br(5, k, l) = br(1, k, l) * f4(t, cond(l))
        if (simps) goto 104
        br(6, k, l) = br(1, k, l) * f5(t, cond(l))
        br(7, k, l) = br(1, k, l) * f6(t, cond(l))
    104 r(k + 1, l) = (fe(l) + r(k, l)) * half
        bou5(l) = ep(l) * (r(k, l) - s(l, 1))
        del(2, l) = r(k + 1, l) - fe(l)
c
c          Checking the bound for the trapezoidal rule
c
        del(3, l) = two * del(2, l)
```

```
      bou1 = max(br(1, k, l), bl(1, l)) * binc(3, l) +
     *  two * max(br(2, k, l), bl(2, l)) * binc(2, l) +
     *  max(br(3, k, l), bl(3, l)) * binc(1, l)
      bou3 = bou4(l) + bou1 * (del(3, l) ** 3) * prod(l) / twelve
      itype(l) = 3
      if (bou3 .le. bou5(l)) goto 200
c
c        Checking the bound for Simpsons rule.
c
      bou1 = zero
      do 110 ij = 1, 5
        jk = 6 - ij
        bou2 = max(br(ij, k, l), bl(ij, l))
        bou1 = bou1 + bou2 * binc(jk, l) * bcs(ij)
  110 continue
      bou3 = bou4(l) + bou1 * (del(2, l) ** 5) * prod(l) / ninety
      itype(l) = 2
      if (bou3 .le. bou5(l)) goto 200
      if (simps) goto 130
c
c        Checking the bound for boules rule, if necessary.
c
      del(1, l) = half * del(2, l)
      bou1 = zero
      itype(l) = 1
      do 120 ij = 1, 7
        jk = 8 - ij
        bou2 = max(br(ij, k, l), bl(ij, l))
        bou1 = bou1 + bou2 * binc(jk, l) * bcn(ij)
  120 continue
      bou3 = bou4(l) + bou1 * (del(1, l) ** 7) * prod(l) * eight /
     *  nine45
      if (bou3 .le. bou5(l)) goto 200
c
c        Sub-dividing further at level l when the bound is too big.
c
  130 k = k + 1
      if (k .gt. 35) return
      goto 100
  200 bint(l) = bint(l) + bou3 - bou4(l)
      bou4(l) = bou3
      ksa(l) = k
      if (ind(l) .eq. 6) goto 202
      if (itype(l) - 2) 205, 206, 210
c
c        The next 30 lines condition on the value xs and go to level l+1
c
  202 ind(l) = 5
      xs = fe(l)
      fact(l) = bl(1, l)
  203 xss(nm1, l + 1) = xss(nm1, l) + bh(l) * (xs + xm(l))
      do 204 ll = l, nm2
  204 xss(ll, l + 1) = xss(ll, l) + beta(ll, l) * (xs + xm(l))
      if (l .eq. nm2) goto 300
c
c        xm is the mean of the next variable given those fixed so far.
c
      xm(l + 1) = xss(l, l + 1)
      prod(l + 1) = prod(l) * fact(l)
      l = l + 1
      goto 90
```

```
  205 ind(l) = 4
      k = ksa(l)
      xs = half * (fe(l) + r(k + 1, l))
      goto 207
  206 ind(l) = 3
      k = ksa(l)
      xs = r(k + 1, l)
  207 t = xs / cond(l)
      fact(l) = phi(t,condl(l))
      goto 203
  208 ind(l) = 2
      k = ksa(l)
      xs = half * (r(k, l) + r(k + 1, l))
      goto 207
  210 ind(l) = 1
      k = ksa(l)
      xs = r(k, l)
      fact(l) = br(1, k, l)
      goto 203
c
c        Evaluate conditional bivariate probabilities at deepest level
c
  300 x1 = fac * (xss(nm1, nm1) - d(1)) / sd(1)
      x2 = fac * (xss(nm2, nm1) - d(2)) / sd(2)
      l = nm1
      ans(l) = bivnor(x1, x2, rho)
      if (ipr) goto 310
      y1 = (xss(nm1, nm1) - c(1)) / sd(1)
      y2 = (xss(nm2, nm1) - c(2)) / sd(2)
      wu = bivnor(y1, y2, rho)
      wt = bivnor(x1, y2, rho)
      wb = bivnor(y1, x2, rho)
      ans(l) = ans(l) + wu - wt - wb
310   if (l .eq. 1) goto 340
      l = l - 1
c
c        Advancing the integration at the current level
c
      indl = ind(l)
      numl = num(l)
      ity = itype(l)
      temp = fact(l) * ans(l + 1)
      temb = bint(l + 1)
      fsa = one
      if (indl .ne. 1) goto 315
      tem = temp
      temp = temp + prep(l)
      temb = temb + preb(l)
      prep(l) = tem
      preb(l) = bint(l + 1)
      fsa = two
  315 ans(l) = ans(l) + coef(indl, ity) * temp * del(ity, l)
      bint(l) = bint(l) + coef(indl, ity) * temb * del(ity, l)
c
c        Making use of the error which did not accumulate at level l+1
c
      ep(l) = ep(l) + del(ity, l) * coef(indl, ity) * (fsa * float(nm2 -
     *  l) * epsi / pr2(l + 1) - temb) / (s(l, numl) - s(l, 1))
      if (indl .eq. 1) goto 320
      igo = indl - (1 + (itype(l) * (itype(l) - 1)) / 2)
      goto (210, 208, 206, 205), igo
```

```
c
c        Un-subdividing at level l.
  320 k = ksa(l)
      do 322 i = 1, ien
  322 bl(i, l) = br(i, k, l)
      ind(l) = 5
      fe(l) = r(k, l)
      if (k .eq. 1) goto 326
      k = k - 1
      goto 104
  326 if (intvl(l) .eq. num(l)) goto 310
      intvl(l) = intvl(l) + 1
      intl = intvl(l)
      r(1, l) = s(l, intl)
      goto 100
c
c        Completion of integration and bounding.
c
 340  ifault = 0
      prob = ans(1)
      bound = bint(1) + eplos
      return
c
c        special cases --
c        label 350 - n=1
c        label 360 - n=2
c
 350  prob = alnorm(d(1), .false.) - alnorm(c(1), .false.)
      return
 360  rho = sigma(2)
      if (abs(rho) .gt. one) goto 400
      y1 = -d(1) * fac
      y2 = -d(2) * fac
      prob = bivnor(y1, y2, rho)
      if (ipr) return
      x1 = -c(1)
      x2 = -c(2)
      wl = bivnor(x1, x2, rho)
      wt = bivnor(x1, y2, rho)
      wb = bivnor(y1, x2, rho)
      prob = wl - wt - wb + prob
      return
c
c        Error return for covariance not positive definite.
c
  400 ifault = 4
      return
      end
c
c
c
      double precision function phi(x, y)
c
c        algorithm as 195.1  appl. statist. (1984) vol.33, no.1
c
c        computes univariate normal density
c        xlow=log(smallest floating point number)
c        sq2p=log(sqrt(two*pi))
c
      double precision arg, half, sq2p, x, xlow, y, zero
c
```

```
      data xlow /-87.0D0/, sq2p /0.91893853320467274D0/, zero /0.D0/,
     *  half /0.5D0/
      phi = zero
      arg = -half * x * x - sq2p - y
      if (arg .gt. xlow) phi = exp(arg)
      return
      end
C
C
C
      DOUBLE PRECISION FUNCTION BIVNOR(AH, AK, R)
C
C     BIVNOR is a controlled precision Fortran function to calculate
C     the bivariate normal upper right area, viz. the probability for
C     two normal variates X and Y whose correlation is R, that X > AH
C     and Y > AK.
C     The accuracy is specified as the number of decimal digits, IDIG.
C
C     Reference:
C        Donnelly, T.G. (1973) Algorithm 462, Bivariate normal
C        distribution, Comm. A.C.M., vol.16, p. 638.
C
C     Corrected for the case AH = 0, AH non-zero by reversing AH & AK
C     in such cases.
C
      DOUBLE PRECISION AH, AK, R
C
C     Local variables
C
      DOUBLE PRECISION TWOPI, B, XAH, XAK, GH, GK, RR, GAUSS, DERF, H2,
     +   A2, H4, EX, W2, AP, S2, SP, S1, SN, SGN, SQR, CON, WH, WK, GW,
     +   T, G2, CONEX, CN, TWO, ZERO, ONE, FOUR, QUART, HALF, EXPLIM
      INTEGER IDIG, IS
      DATA TWO/2.D0/, ZERO/0.D0/, ONE/1.D0/, FOUR/4.D0/, QUART/0.25D0/,
     +   HALF/0.5D0/
      GAUSS(T) = (ONE + DERF(T/SQRT(TWO)))/TWO
C
C     GAUSS is a univariate lower normal tail area calculated here from
C     the central error function, DERF.
C     It may be replaced by the algorithm in Hill, I.D. and Joyce, S.A.
C     Algorithm 304, Normal curve integral (S15), Comm. A.C.M. (10),
C     (June 1967), p.374 or with Applied Statistics algorithm AS66.
C     Source: Owen, D.B. Ann. Math. Statist., vol.27 (1956), p.1075.
C
      DATA TWOPI/6.2831 85307 17958 7D0/, IDIG/15/, EXPLIM/80.D0/
C
      B = ZERO
      IF (AH .EQ. ZERO) THEN
        XAH = AK
        XAK = AH
      ELSE
        XAH = AH
        XAK = AK
      END IF

      GH = GAUSS(-XAH) / TWO
      GK = GAUSS(-XAK) / TWO
      IF (R .EQ. ZERO) THEN
        B = FOUR * GH * GK
        GO TO 350
      END IF
```

```
      RR = ONE - R*R
      IF (RR .LT. ZERO) THEN
        WRITE(*, *)'Error in BIVNOR, R = ', R
        GO TO 390
      END IF
      IF (RR .GT. ZERO) GO TO 100
C
C     R^2 = 1.0
C
      IF (R .GE. ZERO) GO TO 70
      IF (XAH + XAK .GE. ZERO) GO TO 350
      B = TWO * (GH + GK) - ONE
      GO TO 350
   70 IF (XAH - XAK .LT. ZERO) THEN
        B = TWO * GK
      ELSE
        B = TWO * GH
      END IF
      GO TO 350
C
C     Regular case, R^2 < 1
C
  100 SQR = SQRT(RR)
      IF (IDIG .EQ. 15) THEN
        CON = TWOPI * 1.D-15 / TWO
      ELSE
        CON = TWOPI / TWO / 10**IDIG
      END IF

      IF (XAH .NE. ZERO) GO TO 170
      IF (XAK .NE. ZERO) GO TO 190
      B = ATAN(R/SQR) / TWOPI + QUART
      GO TO 350
  170 B = GH

      IF (XAH*XAK) 180, 200, 190
  180 B = B - HALF
  190 B = B + GK
  200 WH = -XAH
      WK = (XAK/XAH - R)/SQR
      GW = TWO * GH
      IS = -1
  210 SGN = -ONE
      T = ZERO
      IF (WK .EQ. ZERO) GO TO 320
      IF (ABS(WK) - ONE) 270, 230, 240
  230 T = WK * GW * (ONE - GW) / TWO
      GO TO 310
  240 SGN = -SGN
      WH = WH * WK
      G2 = GAUSS(WH)
      WK = ONE / WK
      IF (WK .LT. ZERO) B = B + HALF
      B = B - (GW + G2)/TWO + GW*G2
  270 H2 = WH * WH
      A2 = WK * WK
      H4 = H2 / TWO
      IF (H4 .LT. EXPLIM) THEN
        EX = EXP(-H4)
      ELSE
```

```
          EX = ZERO
        END IF
        W2 = H4 * EX
        AP = ONE
        S2 = AP - EX
        SP = AP
        S1 = ZERO
        SN = S1
        CONEX = ABS(CON / WK)
        GO TO 290

  280 SN = SP
        SP = SP + ONE
        S2 = S2 - W2
        W2 = W2 * H4 / SP
        AP = -AP*A2
  290 CN = AP * S2 / (SN + SP)
        S1 = S1 + CN
        IF (ABS(CN) - CONEX .GT. ZERO) GO TO 280

        T = (ATAN(WK) - WK*S1) / TWOPI
  310 B = B + SGN*T
  320 IF (IS .GE. 0) GO TO 350
        IF (XAK .NE. ZERO) THEN
          WH = -XAK
          WK = (XAH/XAK - R) / SQR
          GW = TWO * GK
          IS = 1
          GO TO 210
        END IF

  350 IF (B .LT. ZERO) B = ZERO
        IF (B .GT. ONE) B = ONE
  390 BIVNOR = B
C
        RETURN
        END


C
C
C
        SUBROUTINE INVERT(A, AINV, C, N, DET, IFAULT)
C
C     Invert the NxN symmetric matrix, A, of which only the lower
C     triangle is stored by rows.   The inverse is returned in AINV.
C     The inverse of the last I rows and columns, for I = 1, 2, ...,
C     N-2, is returned in C(I,*).   DET = the determinant of A.
C     IFAULT = 4 if A is not positive definite.
C
        DOUBLE PRECISION A(*), AINV(*), C(5,*), DET
        INTEGER N, IFAULT
C
C     Local variables
C
        DOUBLE PRECISION ONE, WK(15), W(7)
        INTEGER NULLTY, NN, I, ROW1, WKPOS, ROW, APOS, COL
        DATA ONE/1.D0/
C
        DO 50 I = 1, N-2
C
C     Copy the last I rows and columns of A into WK.
```

```
C
        NN = I * (I + 1) / 2
        ROW1 = N + 1 - I
        WKPOS = 1
        DO 20 ROW = ROW1, N
          APOS = ROW * (ROW - 1) / 2 + ROW1
          DO 10 COL = ROW1, ROW
            WK(WKPOS) = A(APOS)
            WKPOS = WKPOS + 1
            APOS = APOS + 1
   10     CONTINUE
   20   CONTINUE
C
C     Call SYMINV to invert WK in situ.
C
        CALL SYMINV(WK, I, NN, WK, W, NULLTY, IFAULT)
        IF (IFAULT .NE. 0) GO TO 70
C
C     Copy the inverse into the I-th row of C.
C
        DO 40 APOS = 1, NN
   40   C(I, APOS) = WK(APOS)
   50 CONTINUE
C
C     Use AS6 (CHOL) to calculate the determinant.
C     This is inefficient as AS7 will call AS6 again to repeat the same
C     calculations.   AS6/7 can easily be converted to calculate the
C     determinant.
C
      NN = N * (N + 1) / 2
      CALL CHOL(A, N, NN, AINV, NULLTY, IFAULT)
      IF (IFAULT .NE. 0) GO TO 70
      DET = ONE
      APOS = 1
      DO 60 ROW = 1, N
        DET = DET * AINV(APOS)
        APOS = APOS + ROW + 1
   60 CONTINUE
      DET = DET * DET
C
C     Invert the full matrix.
C
      CALL SYMINV(A, N, NN, AINV, W, NULLTY, IFAULT)
      IF (IFAULT .EQ. 0) RETURN
C
   70 IFAULT = 4
      RETURN
      END
C
C
C
      SUBROUTINE CALERF(ARG,RESULT,JINT)
C------------------------------------------------------------------
C
C THIS PACKET COMPUTES THE ERROR AND COMPLEMENTARY ERROR FUNCTIONS
C   FOR REAL ARGUMENTS  ARG.  IT CONTAINS TWO FUNCTION TYPE
C   SUBPROGRAMS,  ERF  AND  ERFC  (OR  DERF  AND  DERFC),  AND ONE
C   SUBROUTINE TYPE SUBPROGRAM,  CALERF.  THE CALLING STATEMENTS
C   FOR THE PRIMARY ENTRIES ARE
C
C                  Y=ERF(X)     (OR   Y=DERF(X) )
```

```
C   AND
C                  Y=ERFC(X)    (OR   Y=DERFC(X) ).
C
C   THE ROUTINE  CALERF  IS INTENDED FOR INTERNAL PACKET USE ONLY,
C   ALL COMPUTATIONS WITHIN THE PACKET BEING CONCENTRATED IN THIS
C   ROUTINE.  THE FUNCTION SUBPROGRAMS INVOKE  CALERF  WITH THE
C   STATEMENT
C          CALL CALERF(ARG,RESULT,JINT)
C   WHERE THE PARAMETER USAGE IS AS FOLLOWS
C
C      FUNCTION                     PARAMETERS FOR CALERF
C       CALL             ARG                  RESULT          JINT
C     ERF(ARG)      ANY REAL ARGUMENT         ERF(ARG)          0
C     ERFC(ARG)     ABS(ARG) .LT. XMAX        ERFC(ARG)         1
C
C   THE MAIN COMPUTATION EVALUATES NEAR MINIMAX APPROXIMATIONS
C   FROM "RATIONAL CHEBYSHEV APPROXIMATIONS FOR THE ERROR FUNCTION"
C   BY W. J. CODY, MATH. COMP., 1969, PP. 631-638.  THIS
C   TRANSPORTABLE PROGRAM USES RATIONAL FUNCTIONS THAT THEORETICALLY
C   APPROXIMATE  ERF(X)  AND  ERFC(X)  TO AT LEAST 18 SIGNIFICANT
C   DECIMAL DIGITS.  THE ACCURACY ACHIEVED DEPENDS ON THE ARITHMETIC
C   SYSTEM, THE COMPILER, THE INTRINSIC FUNCTIONS, AND PROPER
C   SELECTION OF THE MACHINE-DEPENDENT CONSTANTS.
C
C------------------------------------------------------------------
C
C EXPLANATION OF MACHINE-DEPENDENT CONSTANTS
C
C   XSMALL = ARGUMENT BELOW WHICH ERF(X) MAY BE REPRESENTED
C            BY   2*X/SQRT(PI)  AND ABOVE WHICH  X*X  WILL
C            NOT UNDERFLOW.  A CONSERVATIVE VALUE IS THE
C            LARGEST X SUCH THAT   1.0 + X = 1.0   TO MACHINE
C            PRECISION.
C   XMAX   = LARGEST ARGUMENT ACCEPTABLE TO  ERFC;  SOLUTION TO
C            EQUATION:  W(X) * (1-0.5/X**2) = XMIN,  WHERE
C            W(X) = EXP(-X*X)/(X*SQRT(PI)),  AND XMIN IS THE
C            SMALLEST POSITIVE MACHINE NUMBER (SEE TABLE BELOW).
C
C     APPROXIMATE VALUES FOR SOME IMPORTANT MACHINES ARE:
C
C                           XSMALL     XMAX     XMIN
C
C    IBM 195     (D.P.)   1.39D-17   13.306   5.40D-79
C    CDC 7600    (S.P.)   7.11E-15   25.922   3.13E-294
C    CRAY-1      (S.P.)   7.11E-15   75.326   4.58E-2467
C    UNIVAC 1108 (D.P.)   1.73D-18   26.582   2.78D-309
C    VAX 11/780  (S.P.)   5.96E-8     9.269   2.94E-39
C    VAX 11/780  (D.P.)   1.39D-17    9.269   2.94D-39
C    IBM PC      (S.P.)   5.96E-8     9.194   1.18E-38
C    IBM PC      (D.P.)   1.11D-16   26.543   2.23D-308
C
C------------------------------------------------------------------
C
C ERROR RETURNS
C
C  THE PROGRAM RETURNS  ERFC = 0  FOR  ARG .GT. XMAX.
C
C
C OTHER SUBPROGRAMS REQUIRED (SINGLE PRECISION VERSION)
C
C     ABS, EXP
```

```
C
C OTHER SUBPROGRAMS REQUIRED (DOUBLE PRECISION VERSION)
C
C     DABS, DEXP
C
C
C  AUTHOR: W. J. CODY
C          MATHEMATICS AND COMPUTER SCIENCE DIVISION
C          ARGONNE NATIONAL LABORATORY
C          ARGONNE, IL 60439
C
C  LATEST MODIFICATION: JANUARY 8, 1985
C
C------------------------------------------------------------------
      INTEGER I,JINT
CS    REAL             A,ARG,B,C,D,FOUR,HALF,P,ONE,Q,RESULT,SQRPI,
CS   1               TWO,THRESH,X,XMAX,XDEN,XNUM,XSMALL,Y,YSQ,ZERO
      DOUBLE PRECISION A,ARG,B,C,D,FOUR,HALF,P,ONE,Q,RESULT,SQRPI,
     1               TWO,THRESH,X,XMAX,XDEN,XNUM,XSMALL,Y,YSQ,ZERO
      DIMENSION A(5),B(4),C(9),D(8),P(6),Q(5)
C------------------------------------------------------------------
C  MATHEMATICAL CONSTANTS
C------------------------------------------------------------------
CS    DATA FOUR,ONE,HALF,TWO,ZERO/4.0E0,1.0E0,0.5E0,2.0E0,0.0E0/
CS    DATA SQRPI/5.6418958354775628695E-1/,THRESH/0.46875E0/
      DATA FOUR,ONE,HALF,TWO,ZERO/4.0D0,1.0D0,0.5D0,2.0D0,0.0D0/
      DATA SQRPI/5.6418958354775628695D-1/,THRESH/0.46875D0/
C------------------------------------------------------------------
C  MACHINE-DEPENDENT PARAMETERS
C------------------------------------------------------------------
CS    DATA XSMALL/4.2E-16/, XMAX/9.269E0/
      DATA XSMALL/4.2D-16/, XMAX/9.269D0/
C------------------------------------------------------------------
C  COEFFICIENTS FOR APPROXIMATION TO DERF IN FIRST INTERVAL
C------------------------------------------------------------------
CS    DATA A/3.16112374387056560E00,1.13864154151050156E02,
CS   1       3.77485237685302021E02,3.20937758913846947E03,
CS   2       1.85777706184603153E-1/
CS    DATA B/2.36012909523441209E01,2.44026637934444173E02,
CS   1       1.28261652607737228E03,2.84423683343917062E03/
      DATA A/3.16112374387056560D00,1.13864154151050156D02,
     1       3.77485237685302021D02,3.20937758913846947D03,
     2       1.85777706184603153D-1/
      DATA B/2.36012909523441209D01,2.44026637934444173D02,
     1       1.28261652607737228D03,2.84423683343917062D03/
C------------------------------------------------------------------
C  COEFFICIENTS FOR APPROXIMATION TO DERFC IN SECOND INTERVAL
C------------------------------------------------------------------
CS    DATA C/5.64188496988670089E-1,8.88314979438837594E0,
CS   1       6.61191906371416295E01,2.98635138197400131E02,
CS   2       8.81952221241769090E02,1.71204761263407058E03,
CS   3       2.05107837782607147E03,1.23033935479799725E03,
CS   4       2.15311535474403846E-8/
CS    DATA D/1.57449261107098347E01,1.17693950891312499E02,
CS   1       5.37181101862009858E02,1.62138957456669019E03,
CS   2       3.29079923573345963E03,4.36261909014324716E03,
CS   3       3.43936767414372164E03,1.23033935480374942E03/
      DATA C/5.64188496988670089D-1,8.88314979438837594D0,
     1       6.61191906371416295D01,2.98635138197400131D02,
     2       8.81952221241769090D02,1.71204761263407058D03,
     3       2.05107837782607147D03,1.23033935479799725D03,
```

```
     4       2.15311535474403846D-8/
      DATA D/1.57449261107098347D01,1.17693950891312499D02,
     1       5.37181101862009858D02,1.62138957456669019D03,
     2       3.29079923573345963D03,4.36261909014324716D03,
     3       3.43936767414372164D03,1.23033935480374942D03/
C------------------------------------------------------------------
C  COEFFICIENTS FOR APPROXIMATION TO DERFC IN THIRD INTERVAL
C------------------------------------------------------------------
CS    DATA P/3.05326634961232344E-1,3.60344899949804439E-1,
CS   1       1.25781726111229246E-1,1.60837851487422766E-2,
CS   2       6.58749161529837803E-4,1.63153871373020978E-2/
CS    DATA Q/2.56852019228982242E00,1.87295284992346047E00,
CS   1       5.27905102951428412E-1,6.05183413124413191E-2,
CS   2       2.33520497626869185E-3/
      DATA P/3.05326634961232344D-1,3.60344899949804439D-1,
     1       1.25781726111229246D-1,1.60837851487422766D-2,
     2       6.58749161529837803D-4,1.63153871373020978D-2/
      DATA Q/2.56852019228982242D00,1.87295284992346047D00,
     1       5.27905102951428412D-1,6.05183413124413191D-2,
     2       2.33520497626869185D-3/
C------------------------------------------------------------------
      X = ARG
CS    Y = ABS(X)
      Y = DABS(X)
      IF (Y .GT. FOUR) GO TO 200
      IF (Y .GT. THRESH) GO TO 100
C------------------------------------------------------------------
C  EVALUATE ERF FOR ABS(X) .LE. 0.46875
C------------------------------------------------------------------
      YSQ = ZERO
      IF (Y .GT. XSMALL) YSQ = Y * Y
      XNUM = A(5)*YSQ
      XDEN = YSQ
      DO 20 I = 1, 3
         XNUM = (XNUM + A(I)) * YSQ
         XDEN = (XDEN + B(I)) * YSQ
   20 CONTINUE
      RESULT = X * (XNUM + A(4)) / (XDEN + B(4))
      IF (JINT .NE. 0) RESULT = ONE - RESULT
      GO TO 800
C------------------------------------------------------------------
C  EVALUATE ERFC FOR 0.46875 .LT. ABS(X) .LE. 4.0
C------------------------------------------------------------------
  100 YSQ = Y * Y
      XNUM = C(9)*Y
      XDEN = Y
      DO 120 I = 1, 7
         XNUM = (XNUM + C(I)) * Y
         XDEN = (XDEN + D(I)) * Y
  120 CONTINUE
CS    RESULT = EXP(-YSQ) * (XNUM + C(8)) / (XDEN + D(8))
      RESULT = DEXP(-YSQ) * (XNUM + C(8)) / (XDEN + D(8))
      GO TO 300
C------------------------------------------------------------------
C  EVALUATE ERFC FOR ABS(X) .GT. 4.0
C------------------------------------------------------------------
  200 RESULT = ZERO
      IF (Y .GE. XMAX) GO TO 300
  220 YSQ = ONE / (Y * Y)
      XNUM = P(6)*YSQ
      XDEN = YSQ
```

```
      DO 240 I = 1, 4
         XNUM = (XNUM + P(I)) * YSQ
         XDEN = (XDEN + Q(I)) * YSQ
  240 CONTINUE
      RESULT = YSQ *(XNUM + P(5)) / (XDEN + Q(5))
CS    RESULT = (EXP(-Y*Y) / Y) * (SQRPI - RESULT)
      RESULT = (DEXP(-Y*Y) / Y) * (SQRPI - RESULT)
C---------------------------------------------------------------------
C  FIX UP FOR NEG. ARG., ERF, ETC.
C---------------------------------------------------------------------
  300 IF (JINT .EQ. 0) GO TO 350
      IF (X .LT. ZERO) RESULT = TWO - RESULT
      GO TO 800
  350 RESULT = (HALF - RESULT) + HALF
      IF (X .LT. ZERO) RESULT = -RESULT
C---------------------------------------------------------------------
  800 RETURN
C---------- LAST CARD OF CALERF ----------
      END




CS    REAL FUNCTION ERF(X)
      DOUBLE PRECISION FUNCTION DERF(X)
C---------------------------------------------------------------------
C
C  PROGRAM TO COMPUTE THE ERROR FUNCTION
C
C   AUTHOR - W. J. CODY
C
C   DATE - JANUARY 8, 1985
C
C---------------------------------------------------------------------
      INTEGER JINT
CS    REAL             X, RESULT
      DOUBLE PRECISION X, RESULT
C---------------------------------------------------------------------
      JINT = 0
      CALL CALERF(X,RESULT,JINT)
CS    ERF = RESULT
      DERF = RESULT
      RETURN
C---------- LAST CARD OF DERF ----------
      END




CS    REAL FUNCTION ERFC(X)
      DOUBLE PRECISION FUNCTION DERFC(X)
C---------------------------------------------------------------------
C
C  PROGRAM TO COMPUTE THE COMPLEMENTARY ERROR FUNCTION
C
C   AUTHOR - W. J. CODY
C
C   DATE - JANUARY 8, 1985
C
C---------------------------------------------------------------------
      INTEGER JINT
```

```
CS    REAL             X, RESULT
      DOUBLE PRECISION X, RESULT
C------------------------------------------------------------------
      JINT = 1
      CALL CALERF(X,RESULT,JINT)
CS    ERFC = RESULT
      DERFC = RESULT
      RETURN
C---------- LAST CARD OF DERFC ----------
      END
c
c-----------------------------------------------------------------------
c
      subroutine mulnor(a, b, sig, eps, n, inf, prob, bound, ifault)
c
c        algorithm as 195  appl. statist. (1984) vol.33, no.1
c
c        computes multivariate normal distribution function and computes
c        the probability that a multivariate normal vector falls in a
c        rectangle in n-space with error less than eps.
c
c     Auxiliary functions required: ANORDF, ANORIN, BNRDF, LFDRG, LFTRG
c     and LINRG from the IMSL Stat/Library.
c
        dimension
     *     a(*), b(*), sig(*), c(7), d(7), co(25), sd(2), coef(5, 3),
     *  binc(7, 5), bl(7, 5), br(7, 35, 5), r(36, 5), s(5, 27),
     *  xss(6, 6), pr2(6), prep(6), preb(6), cv(5, 15), sinv(28),
     *  condl(5), xm(5), cond(5), beta(5, 5), bh(5), fe(5), sigma(28),
     *  ep(5), del(3, 5), bou4(5), bou5(5), ans(6), fact(5), prod(5),
     *  bint(6), bcs(5), bcn(7), eo(13), do(25)
       dimension inf(7), intvl(5), ind(5), ksa(5), num(5), itype(5)
       logical simps, ipr
       data coef(1, 1), coef(2, 1), coef(3, 1), coef(4, 1), coef(5, 1),
     *  coef(1, 2), coef(2, 2), coef(3, 2), coef(4, 2), coef(5, 2),
     *  coef(1, 3), coef(2, 3), coef(3, 3), coef(4, 3), coef(5, 3)
     *  /0.311111111111111, 1.422222222222222, 0.533333333333333,
     *  1.422222222222222, 0.311111111111111, 0.333333333333333, 0.0,
     *  1.333333333333333, 0.0, 0.333333333333333, 0.5, 0.0, 0.0, 0.0,
     *  0.5/
       data bcs(1), bcs(2), bcs(3), bcs(4), bcs(5) /1.0, 4.0, 6.0, 4.0,
     *  1.0/
       data bcn(1), bcn(2), bcn(3), bcn(4), bcn(5), bcn(6), bcn(7)
     *  /1.0, 6.0, 15.0, 20.0, 15.0, 6.0, 1.0/
       data do(1), do(2), do(3), do(4), do(5), do(6), do(7), do(8),
     *  do(9), do(10), do(11), do(12), do(13), do(14), do(15), do(16),
     *  do(17), do(18), do(19), do(20), do(21), do(22), do(23), do(24),
     *  do(25) / - 3.75043972,-3.3242574, -2.85697, -2.36675941,
     *  -2.3344142, -1.8891759, -1.7320508, -1.3556262,-1.15440539,-1.,
     *  -0.74196378, -0.61670659,0.0,
     *  0.61670659, 0.74196378, 1.0, 1.15440539,1.3556262,  1.7320508,
     *  1.8891759,2.3344142, 2.36675941,2.85697, 3.3242574,3.75043972/
       data eo(1), eo(2), eo(3), eo(4), eo(5), eo(6), eo(7), eo(8),
     *  eo(9), eo(10), eo(11), eo(12), eo(13) / - 2.85697, -2.3344142,
     *  -1.7320508, -1.3556262, -1.0, -0.74196378, 0.0, 0.74196378, 1.0,
     *  1.3556262, 1.7320508, 2.3344142, 2.85697/
       data cons2 /6.879833/, cons3 /4.517004/, cons4 /76.371214/,
     *  epsmin /1.0e-8/, epssim /6.0e-5/
       data zero, p05, p15, half, one, two, three, six, eight, ten,
     *  twelve, fiften, forty5, ninety, nine45 /0.0, 0.05, 0.15, 0.5,
     *  1.0, 2.0, 3.0, 6.0, 8.0, 10.0, 12.0, 15.0, 45.0, 90, 945/
```

```
c
c       functions needed to calculate the first six derivatives of
c       the normal density
c
      f2(x, y) = abs((x * x - one) / (y * y))
      f3(x, y) = abs(-x * (x * x - three) / (y * y * y))
      f4(x, y) = abs((three + x * x * (-six + x * x)) / (y ** 4))
      f5(x, y) = abs(x * (-fiften + x * x * (ten - x * x))) / (y ** 5)
      f6(x, y) = abs(-fiften + x * x * (forty5 - x * x * (-fiften +
     *  x * x))) / (y ** 6)
c       checking for faulty data
c
      ifault = 0
      if (eps .le. epsmin) ifault = 2
      if (n .le. 0 .or. n .gt. 7) ifault = 3
      if (ifault .ne. 0) return
      do 1 i = 1, n
    1 if (inf(i) .eq. 2 .and. a(i) .lt. b(i)) ifault = 100 + i
      if (ifault .ne. 0) return
      bound = eps
c       finding z such that p(n(0,1).gt.z).lt. 0.15*eps/n
c       co will contain the roots of the first 5 or 7 hermite
c       polynomials
      ifault=0.
      ept = eps * p15 / float(n)
      z = -anorin(ept) + epsmin
      cup=1-anordf(z)
c
c       inverting sig and the n-2 lower right hand principal minors
c
      ik = 0
      ij = 0
      do 3 i = 1, n
        do 3 j = 1, i
          ik = ik + 1
          if (i .eq. j) goto 2
          ij = ij + 1
          sigma(ik) = sig(ij)
          goto 3
    2     sigma(ik) = one
    3 continue
      if (n .le. 2) goto 4
      call invert(sigma, sinv, cv, n, det, ifault)
      if (ifault .ne. 0) return
      simps = .true.
      if (det .lt. p05 .or. eps .le. epssim) simps = .false.
      prob = zero
      det = sinv(1) * sinv(3) - sinv(2) * sinv(2)
      sd(1) = sqrt(sinv(3) / det)
      sd(2) = sqrt(sinv(1) / det)
      rho = -sinv(2) / (sd(1) * sd(2) * det)
      if (abs(rho) .gt. one) goto 400
    4 nm2 = n - 2
      nm1 = n - 1
c
c       checking whether upper and lower endpoints are too big
c
      eplos = zero
      do 6 l = 1, n
        c(l) = max(b(l), -z)
        d(l) = min(a(l), z)
```

```
          if (inf(l) .eq. 0) d(l) = z
          if (inf(l) .eq. 1) c(l) = -z
          if (a(l) .gt. z .or. inf(l) .eq. 0) eplos = eplos + cup
          if (b(l) .lt. -z .or. inf(l) .eq. 1) eplos = eplos + cup
          if (c(l) .ge. d(l)) return
        6 continue
          if (n .eq. 1) goto 350
          fac = one
          ipr = .false.
          if (inf(1) .ne. 1 .or. inf(2) .ne. 1) goto 7
          ipr = .true.
          eplos = eplos - two * cup
          goto 8
        7 if (inf(1) .ne. 0 .or. inf(2) .ne. 0) goto 8
          fac = -one
          ipr = .true.
          d(1) = c(1)
          d(2) = c(2)
          eplos = eplos - two * cup
        8 if (n .eq. 2) goto 360
          ifault = 5
          epsi = (eps - eplos) / float(nm2)
c         finding regression coefficients (beta,bh) and bounds on the
c         conditional integrals (binc)
c         cond(l)=conditional variance of variable n-l+1 given later
c         variables
c
          do 15 l = 1, nm2
            cond(l) = one / sqrt(cv(l, 1))
            condl(l) = log(cond(l))
            do 10 i = 1, l
            beta(l, i) = zero
            do 10 j = 1, l
              jk = (l - i + 1) * (l - i) / 2 + j
              if (j .gt. l - i + 1) jk = j * (j - 1) / 2 + l - i + 1
              jn = (n - l + j) * (n - l + j - 1) / 2 + n - l
              beta(l, i) = beta(l, i) + sigma(jn) * cv(l, jk)
       10   continue
            k = n - l - 1
            bh(k + 1) = beta(l, l)
            do 11 i = 1, k
              bh(i) = zero
              do 11 j = 1, l
                jn = (j + n - l) * (j + n - l - 1) / 2 + n - l
                ijk = 1 + j * (j - 1) / 2
                bh(i) = bh(i) + sigma(jn) * cv(l, ijk)
       11   continue
            k = 0
            sigc = zero
            do 12 j = 1, l
              do 12 i = 1, j
                k = k + 1
                sigc = sigc + bh(i) * bh(j) * sinv(k)
       12   continue
            binc(1, l) = one
            binc(2, l) = sqrt(sigc)
            binc(3, l) = two * sigc
            binc(4, l) = cons2 * sigc * binc(2, l)
            binc(5, l) = twelve * sigc * sigc
            if (simps) goto 13
            binc(6, l) = sigc * sigc * binc(1, l) * cons3
```

```
           binc(7, l) = (sigc ** 3) * cons4
      13   if (l .lt. nm2) goto 15
           do 14 i = 1, nm2
             bh(i) = zero
             do 14 j = 1, nm2
               jk = (l - i + 1) * (l - i) / 2 + j
               if (j .gt. l - i + 1) jk = j * (j - 1) / 2 + l - i + 1
               jn = (2 + j) * (1 + j) / 2 + 1
               bh(i) = bh(i) + sigma(jn) * cv(l, jk)
      14   continue
      15 continue
         l = 1
c
c          co will contain the roots of the first 5 or 7 hermite
c          polynomials
c
         if (simps) goto 50
         do 40 i = 1, 25
      40 co(i) = do(i)
         iend = 25
         ien = 7
         goto 60
      50 do 55 i = 1, 13
      55 co(i) = eo(i)
         iend = 13
         ien = 5
c
c          initialising values.   xss contains partial sums used for
c          calculating conditional means.
c
      60 do 70 i = 1, nm1
      70 xss(i, 1) = zero
         xm(1) = zero
         prod(1) = one
         pr2(1) = one
         do 80 i = 1, nm2
           ni = n - i + 1
           pr2(i + 1) = pr2(i) * (d(ni) - c(ni))
      80 continue
c
c          bint(l) is a bound on the error accumulated at levels l and
c                  deeper.
c          ans(l)  is the accumulated integral at level l.
c          prep(l) contains the integrand at level l.
c          preb(l) bounds the error accumulated at levels deeper than l.
c
         bint(nm1) = zero
      90 intvl(l) = 2
         ans(l) = zero
         bou4(l) = zero
         bint(l) = zero
         prep(l) = zero
         preb(l) = zero
         k = 1
c
c          finding which of the co are in the current interval.
c          s(l,.) are the endpoints of intervals on which the integrand
c          at level l and its derivatives are monotone.
c          num(l) is the number of such intervals.
c
         nl = n - l + 1
```

```
      s(l, 1) = c(nl) - xm(l)
      s(l, iend + 2) = d(nl) - xm(l)
      num(l) = iend + 2
      do 91 i = 1, iend
        njs = i
        if (s(l, 1) .lt. co(i) * cond(l)) goto 92
        num(l) = num(l) - 1
   91 continue
   92 if (num(l) .eq. 2) goto 99
      do 94 i = njs, iend
        mjs = iend - i + njs
        if (s(l, iend + 2) .ge. co(mjs) * cond(l)) goto 96
        num(l) = num(l) - 1
   94 continue
   96 if (num(l) .eq. 2) goto 99
      do 98 i = njs, mjs
        inj = i - njs + 2
        s(l, inj) = co(i) * cond(l)
   98 continue
   99 numl = num(l)
      s(l, numl) = s(l, iend + 2)
c
c       ep(l)  is an upper limit on the allowable error at level l.
c       r(k,l) is the right end-point of the current sub-interval.
c       fe(l)  is the left end-point of the current sub-interval.
c       ind(l)-1  indicates which point of the newton-cotes formula
c       we are dealing with.
c
      ep(l) = epsi / pr2(l + 1)
      r(1, l) = s(l, 2)
      ind(l) = 6
c
c       bounding derivatives at left end-point of current interval.
c       bl(i,l) is a bound on the ith derivative of the normal
c       density at level l at the left end-point.
c
      fe(l) = s(l, 1)
      t = fe(l) / cond(l)
      bl(1, l) = phi(t, condl(l))
      bl(2, l) = bl(1, l) * abs(t / cond(l))
      bl(3, l) = bl(1, l) * f2(t, cond(l))
      bl(4, l) = bl(1, l) * f3(t, cond(l))
      bl(5, l) = bl(1, l) * f4(t, cond(l))
      if (simps) goto 100
      bl(6, l) = bl(1, l) * f5(t, cond(l))
      bl(7, l) = bl(1, l) * f6(t, cond(l))
c       bounding derivatives at right end-point of sub-interval.
c       br(i,l) is a bound on the ith derivative of the normal
c       density at level l at the right end-point.
c
  100 t = r(k, l) / cond(l)
      br(1, k, l) = phi(t, condl(l))
      br(2, k, l) = br(1, k, l) * abs(t / cond(l))
      br(3, k, l) = br(1, k, l) * f2(t, cond(l))
      br(4, k, l) = br(1, k, l) * f3(t, cond(l))
      br(5, k, l) = br(1, k, l) * f4(t, cond(l))
      if (simps) goto 104
      br(6, k, l) = br(1, k, l) * f5(t, cond(l))
      br(7, k, l) = br(1, k, l) * f6(t, cond(l))
  104 r(k + 1, l) = (fe(l) + r(k, l)) * half
      bou5(l) = ep(l) * (r(k, l) - s(l, 1))
```

```
      del(2, l) = r(k + 1, l) - fe(l)
c
c        checking the bound for the trapezoidal rule
c
      del(3, l) = two * del(2, l)
      bou1 = max(br(1, k, l), bl(1, l)) * binc(3, l) +
     *  two * max(br(2, k, l), bl(2, l)) * binc(2, l) +
     *  max(br(3, k, l), bl(3, l)) * binc(1, l)
      bou3 = bou4(l) + bou1 * (del(3, l) ** 3) * prod(l) / twelve
      itype(l) = 3
      if (bou3 .le. bou5(l)) goto 200
c
c        checking the bound for simpsons rule.
c
      bou1 = zero
      do 110 ij = 1, 5
        jk = 6 - ij
        bou2 = max(br(ij, k, l), bl(ij, l))
        bou1 = bou1 + bou2 * binc(jk, l) * bcs(ij)
  110 continue
      bou3 = bou4(l) + bou1 * (del(2, l) ** 5) * prod(l) / ninety
      itype(l) = 2
      if (bou3 .le. bou5(l)) goto 200
      if (simps) goto 130
c
c        checking the bound for boules rule, if necessary.
c
      del(1, l) = half * del(2, l)
      bou1 = zero
      itype(l) = 1
      do 120 ij = 1, 7
        jk = 8 - ij
        bou2 = max(br(ij, k, l), bl(ij, l))
        bou1 = bou1 + bou2 * binc(jk, l) * bcn(ij)
  120 continue
      bou3 = bou4(l) + bou1 * (del(1, l) ** 7) * prod(l) * eight /
     *  nine45
      if (bou3 .le. bou5(l)) goto 200
c
c        sub-dividing further at level l when the bound is too big.
c
  130 k = k + 1
      if (k .gt. 35) return
      goto 100
  200 bint(l) = bint(l) + bou3 - bou4(l)
      bou4(l) = bou3
      ksa(l) = k
      if (ind(l) .eq. 6) goto 202
      if (itype(l) - 2) 205, 206, 210
c
c        the next 30 lines condition on the value xs and go to level l+1
c
  202 ind(l) = 5
      xs = fe(l)
      fact(l) = bl(1, l)
  203 xss(nm1, l + 1) = xss(nm1, l) + bh(l) * (xs + xm(l))
      do 204 ll = l, nm2
  204 xss(ll, l + 1) = xss(ll, l) + beta(ll, l) * (xs + xm(l))
      if (l .eq. nm2) goto 300
c
c        xm is the mean of the next variable given those fixed so far.
```

```
c
      xm(l + 1) = xss(l, l + 1)
      prod(l + 1) = prod(l) * fact(l)
      l = l + 1
      goto 90
  205 ind(l) = 4
      k = ksa(l)
      xs = half * (fe(l) + r(k + 1, l))
      goto 207
  206 ind(l) = 3
      k = ksa(l)
      xs = r(k + 1, l)
  207 t = xs / cond(l)
      fact(l) = phi(t,condl(l))
      goto 203
  208 ind(l) = 2
      k = ksa(l)
      xs = half * (r(k, l) + r(k + 1, l))
      goto 207
  210 ind(l) = 1
      k = ksa(l)
      xs = r(k, l)
      fact(l) = br(1, k, l)
      goto 203
c
c        evaluate conditional bivariate probabilities at deepest level
c
  300 x1 = fac * (xss(nm1, nm1) - d(1)) / sd(1)
      x2 = fac * (xss(nm2, nm1) - d(2)) / sd(2)
      l = nm1
      ans(l) = bivnor(x1, x2, rho)
      if (ipr) goto 310
      y1 = (xss(nm1, nm1) - c(1)) / sd(1)
      y2 = (xss(nm2, nm1) - c(2)) / sd(2)
      wu = bivnor(y1, y2, rho)
      wt = bivnor(x1, y2, rho)
      wb = bivnor(y1, x2, rho)
      ans(l) = ans(l) + wu - wt - wb
310   if (l .eq. 1) goto 340
      l = l - 1
c
c        advancing the integration at the current level
c
      indl = ind(l)
      numl = num(l)
      ity = itype(l)
      temp = fact(l) * ans(l + 1)
      temb = bint(l + 1)
      fsa = one
      if (indl .ne. 1) goto 315
      tem = temp
      temp = temp + prep(l)
      temb = temb + preb(l)
      prep(l) = tem
      preb(l) = bint(l + 1)
      fsa = two
  315 ans(l) = ans(l) + coef(indl, ity) * temp * del(ity, l)
      bint(l) = bint(l) + coef(indl, ity) * temb * del(ity, l)
c
c        making use of the error which did not accumulate at level l+1
c
```

```
      ep(l) = ep(l) + del(ity, l) * coef(indl, ity) * (fsa * float(nm2 -
     *  l) * epsi / pr2(l + 1) - temb) / (s(l, numl) - s(l, 1))
      if (indl .eq. 1) goto 320
      igo = indl - (1 + (itype(l) * (itype(l) - 1)) / 2)
      goto (210, 208, 206, 205), igo
c
c        un-subdividing at level l.
  320 k = ksa(l)
      do 322 i = 1, ien
  322 bl(i, l) = br(i, k, l)
      ind(l) = 5
      fe(l) = r(k, l)
      if (k .eq. 1) goto 326
      k = k - 1
      goto 104
  326 if (intvl(l) .eq. num(l)) goto 310
      intvl(l) = intvl(l) + 1
      intl = intvl(l)
      r(1, l) = s(l, intl)
      goto 100
c
c        completion of integration and bounding.
c
 340  ifault = 0
      prob = ans(1)
      bound = bint(1) + eplos
      return
c
c        special cases --
c        label 350 - n=1
c        label 360 - n=2
c
350   prob=anordf(d(1))-anordf(c(1))
      return
  360 rho = sigma(2)
      if (abs(rho) .gt. one) goto 400
      y1 = -d(1) * fac
      y2 = -d(2) * fac
      prob = bivnor(y1, y2, rho)
      if (ipr) return
      x1 = -c(1)
      x2 = -c(2)
      wl = bivnor(x1, x2, rho)
      wt = bivnor(x1, y2, rho)
      wb = bivnor(y1, x2, rho)
      prob = wl - wt - wb + prob
      return
c
c        error return for covariance not positive definite.
c
  400 ifault = 4
      return
      end
c
c
c
      function phi(x, y)
c
c        algorithm as 195.1  appl. statist. (1984) vol.33, no.1
c
c        computes univariate normal density
```

```
c        xlow=log(smallest floating point number)
c        sq2p=log(sqrt(two*pi))
c
      real arg, half, sq2p, x, xlow, y, zero
c
      real exp
c
      data xlow /-87.0/, sq2p /0.91893853320467274/, zero /0.0/,
     *  half /0.5/
      phi = zero
      arg = -half * x * x - sq2p - y
      if (arg .gt. xlow) phi = exp(arg)
      return
      end
c
c
c
        real function bivnor(x,y,r)
        data range/25.0/
c
        xx=max(-range,min(range,x))
        yy=max(-range,min(range,y))
        bivnor=1+bnrdf(xx,yy,r)-anordf(xx)-anordf(yy)
        return
        end
c
c
c
        subroutine invert(a,ai,c,n,det,ier)
        dimension a(*),ai(*),c(5,1),s(5,5),ipiv(20),aa(5,5)
c
        l=0
        do 3 i=1,n
          do 3 j=1,i
            l=l+1
            aa(i,j)=a(l)
3       aa(j,i)=aa(i,j)
        call linrg(n,aa,5,s,5)
        l=0
        do 4 i=1,n
          do 4 j=1,i
            l=l+1
4       ai(l)=s(i,j)
        do 1 nm=1,(n-2)
          ii=n+1-nm
          l1=0
          do 2 i=ii,n
            l1=l1+1
            l2=0
            do 2 j=ii,n
              l2=l2+1
2         s(l1,l2)=aa(i,j)
          call linrg(nm,s,5,s,5)
          l=0
          do 1 i=1,l1
            do 1 j=1,i
              l=l+1
1       c(nm,l)=s(i,j)
        call lftrg(n,aa,5,aa,5,ipiv)
        call lfdrg(n,aa,5,ipiv,det1,det2)
        det=det1*10.0**det2
```

```
      ier=0
      return
      end
c
c
c
      subroutine matinv(a,nd,n,ifault)
      dimension a(nd,1)
c
      ifault=0
      call linrg(n,a,nd,a,nd)
      return
      end
```

```
      SUBROUTINE LOGCCH(NS, NCA, NCT, NIMAX, NMAX, NMAX1, NVMAX, NVMAX1,
     *  NV, Z, IVAR, COVI, CNTR, W, WB, WDB, WD2B, U, INS, DB, D2B, DL,
     *  B, COV, CHI2, ST, IFAULT)
C
C       ALGORITHM AS 196 APPL. STATIST. (1984) VOL.33, NO.1
C
C       LOGISTIC ANALYSIS OF CASE-CONTROL STUDIES
C
c     *** WARNING  This file has been input using a scanner and may
c                  contain errors.
c
      INTEGER NCA(NS), NCT(NS), IVAR(NVMAX), INS(NS)
      REAL Z(NVMAX, NIMAX), COVI(NVMAX1), CNTR(NVMAX, NS), W(NVMAX),
     *  WB(NMAX1), WDB(NVMAX, NMAX1), WD2B(NVMAX1, NMAX1), U(NMAX),
     *  DB(NVMAX), D2B(NVMAX1), DL(NVMAX), B(NVMAX), COV(NVMAX1)
C
      REAL CHI2, EPS, ZERO, ONE, TWO, BMN, CONST, C1, FM, RLIK, RLIKP,
     *  RLIKS, ST, T, TTT
C
      REAL ABS, ALGFAC, ALOG, FLOAT
C
      DATA MAXIT /20/, EPS /0.00001/, ZERO /0.0/, ONE /1.0/, TWO /2.0/
C
C       INITIAL SETTINGS
C
      RLIKP = ONE
      IFAULT = 0
      ITS = 0
      IF (NV .GT. NVMAX) GOTO 21
      IF (NVMAX * (NVMAX + 1) / 2 .GT. NVMAX1) GOTO 21
      IF (NMAX + 1 .GT. NMAX1) GOTO 21
      INS(1) = 0
      IF (NCA(1) + NCT(1) .GT. NMAX .OR. NCA(1) + NCT(1) .GT. NIMAX)
     *  GOTO 21
      IF (NS .EQ. 1) GOTO 2
      DO 1 I = 2, NS
      IF (NCA(I) + NCT(I) .GT. NMAX) GOTO 21
      I1 = I - 1
      INS(I) = NCA(I1) + NCT(I1) + INS(I1)
    1 CONTINUE
      IF (INS(NS) + NCA(NS) + NCT(NS) .GT. NIMAX) GOTO 21
C
C       CENTRE THE INDEPENDENT VARIABLES ABOUT THE MEAN OF THE
C       COVARIATES FOR THE CASES (C.F. S.HOWARDS COMMENT TO COX (1972))
C
    2 DO 6 I = 1, NS
      IF (NCA(I) * NCT(I) .EQ. 0) GOTO 6
      M = NCA(I)
      N = M + NCT(I)
      FM = 1.0 / FLOAT(M)
      I1 = INS(I)
      DO 5 K = 1, NV
      CNTR(K, I) = ZERO
      K1 = IVAR(K)
      J1 = I1
      DO 3 J = 1, M
      J1 = J1 + 1
      CNTR(K, I) = CNTR(K, I) + Z(K1, J1)
    3 CONTINUE
      CNTR(K, I) = CNTR(K, I) * FM
      J1 = I1
```

```
      DO 4 J = 1, N
      J1 = J1 + 1
      Z(K1, J1) = Z(K1, J1) - CNTR(K, I)
    4 CONTINUE
    5 CONTINUE
    6 CONTINUE
    7 ITS = ITS + 1
      IF (ITS .GT. MAXIT) GOTO 23
      RLIK = ZERO
      K = 0
      DO 8 J = 1, NV
      DL(J) = ZERO
      DO 8 JJ = 1, J
      K = K + 1
      COVI(K) = ZERO
    8 CONTINUE
C
C        LOOP THROUGH STRATA
C
      DO 14 I = 1, NS
      IF (NCA(I) * NCT(I) .EQ. 0) GOTO 14
      M = NCA(I)
      N = M + NCT(I)
      NID = INS(I)
C
C        FIND U(J)=EXP(Z(J)*BETA) FOR EACH INDIVIDUAL J IN THE STRATA.
C        ALSO, FIHD TTT, THE TOTAL OF THE (Z(J)*BETA)S.
C
      TTT = ZERO
      DO 10 J = 1, N
      J1 = J + NID
      T = ZERO
      DO 9 K = 1, NV
      L = IVAR(K)
      T = T + B(K) * Z(L, J1)
    9 CONTINUE
      TTT = TTT + T
      U(J) = EXP(T)
   10 CONTINUE
C
C        CALC LATE THE CONSTANT CONST=
C        ((N(C)M) * EXP(M*BETA*XBAR))**(1/M)
C        WHERE XBAR IS THE MEAN OF THE COVARIATES OVER THE CASES AND
C        CONTROLS, AHD DIVIDE EACH U(J) BY THIS CONSTANT.
C        THIS KEEPS THE SUMS CALCULATED BY SUBROUTINE HOWARD FROM
C        BECOMING TOO LARGE IN ABSOLUTE VALUE.
C        NOTE - ALGFAC(X)=LN((X)FACTORIAL)
C
      C1 = ALGFAC(N) - (ALGFAC(M) + ALGFAC(N - M)) + TTT * FLOAT(M) /
     *   FLOAT(N)
      CONST = EXP(-C1 / FLOAT(M))
      DO 11 J = 1, N
   11 U(J) = U(J) * CONST
C
C        CALL TO HOWARD TO CALCULATE SUM(EXP(S(L)*BETA)) OVER ALL
C        COMBINATIONS OF N LABELS TAKEN M AT A TIME, AND ITS FIRST AND
C        SECOND DERIVATIVES WITH RESPECT TO BETA.
C        NOTE - THE VALUE FOR THE SUM AND ITS DERIVATIVES RETURNED BY
C        HOWARD ARE THE TRUE VALUES DIVIDED BY THE CONSTANT CONST**M.
C        THEREFORE RLIK IS CORRECTED FOR THIS CONSTANT SO THAT THE
C        LIKELIHOOD RATIO TEST STATISTIC WILL BE CORRECT.
```

```
C        NOTE - SC=BETA*(SUM(Z(J)) OVER THE CASES) = 0 BY DEFINITION.
C
      CALL HOWARD(M, N, U, Z, NID, IVAR, NVMAX, NIMAX, NMAX, NVMAX1, NV,
     *   NMAX1, WB, WDB, WD2B, DB, D2B, BMN)
      RLIK = RLIK - ALOG(BMN) - C1
      L = 0
      IR = 1
      IS = 0
C
C        CALCULATE THE CUMULATIVE SCORE UP TO THIS STRATUM.
C
      DO 13 K = 1, NV
      DL(K) = DL(K) - DB(K) / BMN
      DO 13 KK = 1, K
      L = L + 1
      IS = IS + 1
      IF (IS .LE. IR) GOTO 12
      IR = IR + 1
      IS = 1
C
C       CALCULATE THE CUMULATIVE INFORMATION UP TO THIS STRATUM.
C
   12 COVI(L) = COVI(L) + D2B(L) / BMN - DB(IR)
   13 CONTINUE
   14 CONTINUE
      IF (ITS .EQ. 1) RLIKS = RLIK
C
C        CALCULATE THE INVERSE OF THE INFORMATION MATRIX
C
      CALL SYMINV(COVI, NV, COV, W, NULLTY, IFAULT, NVMAX1)
      IF (IFAULT .NE. 0) GOTO 22
C
C        CALCULATE NEW PARAMETER ESTIMATES
C
      DO 17 I = 1, NV
      W(I) = ZERO
      I2 = I * (I - 1) / 2
      DO 15 J = 1, I
      K = I2 + J
      W(I) = W(I) + DL(J) * COV(K)
   15 CONTINUE
      I1 = I + 1
      IF (I1 .GT. NV) GOTO 17
      DO 16 K = I1, NV
      J = K * (K - 1) / 2 + I
      W(I) = W(I) + DL(K) * COV(J)
   16 CONTINUE
   17 CONTINUE
      DO 18 I = 1, NV
   18 B(I) = B(I) + W(I)
      IF (ITS .NE. 1) GOTO 20
C
C        CALCULATE THE TEST SCORE
C
      ST = ZERO
      DO 19 I = 1, NV
   19 ST = ST + W(I) * DL(I)
C
C        TEST FOR CONVERGENCE
C
   20 RLIK = RLIK - RLIKS
```

```
      IF (ABS(RLIKP - RLIK) .LE. EPS) GOTO 24
      RLIKP = RLIK
      GOTO 7
   21 IFAULT = 1
      GOTO 25
   22 IFAULT = 2
      GOTO 25
   23 IFAULT = 3
      GOTO 25
   24 CHI2 = TWO * RLIK
C
C        RETURN MATRIX Z TO THE FORM IT WAS IN WHEN LOGCCH WAS CALLED
C
   25 DO 27 I = 1, NS
      IF (NCA(I) * NCT(I) .EQ. 0) GOTO 27
      N = NCA(I) + NCT(I)
      I1 = INS(I)
      DO 26 K = 1, NV
      K1 = IVAR(K)
      J1 = I1
      DO 26 J = 1, N
      J1 = J1 + 1
      Z(K1, J1) = Z(K1, J1) + CNTR(K, I)
   26 CONTINUE
   27 CONTINUE
      RETURN
      END
C
      SUBROUTINE HOWARD(M, N, U, Z, NID, IVAR, NVMAX, NIMAX, NMAX,
     *   NVMAX1, NV, NMAX1, WB, WDB, WD2B, DB, D2B, BMN)
C
C        ALGORITHM AS 122.1 APPL. STATIST. (1984) VOL.33, NO.1
C
      INTEGER IVAR(NVMAX)
      REAL U(NMAX), Z(NVMAX, NIMAX), WB(NMAX1), WDB(NVMAX, NMAX1),
     * WD2B(NVMAX1, NMAX1), DB(NVMAX), D2B(NVMAX1)
C
      REAL BMN, ZERO, ONE
C
      DATA ZERO /0.0/, ONE /1.0/
C
      NV1 = NV * (NV + 1) / 2
C
C        INITIALISE THE REQUIRED MATRICES AND VECTORS FOR THE RECURSION
C
      WB(1) = ONE
      I = 0
      DO 1 L = 1, NV
      WDB(L, 1) = ZERO
      DO 1 LL = 1, L
      I = I + 1
      WD2B(I, 1) = ZERO
    1 CONTINUE
      M1 = M + 1
      IF (M .EQ. 0) GOTO 9
      DO 3 J = 2, M1
      WB(J) = ZERO
      I = 0
      DO 2 L = 1, NV
      WDB(L, J) = ZERO
      DO 2 LL = 1, L
```

```
        I = I + 1
        WD2B(I, J) = ZERO
      2 CONTINUE
      3 CONTINUE
        NMMP1 = N - M + 1
        DO 8 IM = 1, NMMP1
        DO 7 J = 1, M
        J1 = J + 1
        I = J + IM - 1
        I1 = I + NID
        IR = 1
        IS = 0
        DO 5 L = 1, NV1
        IS = IS + 1
        IS1 = IVAR(IS)
        IR1 = IVAR(IS)
        IF (IS .LE. IR) GOTO 4
        IR = IR + 1
        IS = 1
        IS1 = IVAR(IS)
        IR1 = IVAR(IR)
C
C          CALCULATE SUM(EXP(S(L)*BETA)) OVER ALL COMBINATIONS OF N
C          LABELS TAKEN M AT A TIME?  AND ITS FIRST AND SECOND DERIVATIVES
C          WITH RESPECT TO BETA.
C          NOTE - THE VALUES RETURNED BT THIS ROUTINE ARE THE TRUE VALUES
C          OF THE SUMS DIVIDED BY TME CONSTANT CONST**M, WHERE CONST
C          IS AS CALCULATED IN 2UBROUTINE LOGCCH.
C
      4 WD2B(L, I1) = WD2B(L, J1) + U(I) * WD2B(L, J) + Z(IR1, I1) *
       *  Z(IS1, I1) * U(I) * WB(J) + Z(IR1, I1) * U(I) * WDB(IS, J) +
       *  Z(IS1, I1) * U(I) * WDB(IR, J)
      5 CONTINUE
        DO 6 L = 1, NV
        L1 = IVAR(L)
        WDB(L, J1) = WDB(L, J1) + U(I) * WDB(L, J) + Z(L1, I1) * U(I) *
       *  WB(J)
      6 CONTINUE
        WB(J1) = WB(J1) + U(I) * WB(J)
      7 CONTINUE
      8 CONTINUE
      9 BMN = WB(M1)
        I = 0
        DO 10 L = 1, NV
        DB(L) = WDB(L, M1)
        DO 10 LL = 1, L
        I = I + 1
        D2B(I) = WD2B(I, M1)
     10 CONTINUE
        RETURN
        END

        REAL FUNCTION ALGFAC(I)
C
C          ALGORITHM AS 196.2 APPL. STATIST. (1984) VOL.33, NO.1
C
C          EVALUATES THE NATURAL LOGARITHM OF GAMMA(I+1)=LN(I-FACTORIAL)
C          (FORTRAN VERSION OF ACM291 - M.C.PIKE AND I.D.HILL,CACM,9,1966)
C
        REAL HALF, CONST, ONE, TWELVE, TREE60, AA, AK, B
C
```

```
      REAL ALOG, FLOAT
C
      DATA HALF, CONST, ONE, TWELVE, TREE60 /0.5, 0.9189385333, 1.0,
     *  12.0, 360.0/

      IF (I .GT. 7) GOTO 3
      B = ONE
      IF (I .LE. 1) GOTO 2
      DO 1 K = 2, I
    1 B = B * FLOAT(K)
    2 ALGFAC = ALOG(B)
      RETURN
    3 AA = FLOAT(I)
      AK = ONE / AA
      ALGFAC = (AA + HALF) * ALOG(AA) + AK * (ONE / TWELVE - (AK * AK)
     *   / TREE60) - AA + CONST
      RETURN
      END
```

```
      SUBROUTINE FLIKAM(P,MP,Q,MQ,W,E,N,SUMSQ,FACT,VW,VL,
     *                  MRP1,VK,MR,TOLER,IFAULT)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
C     ALGORITHM AS 197  APPLIED STATISTICS (1984) VOL.33, NO.1
C
C     COMPUTES THE LIKELIHOOD FUNCTION OF AN AUTOREGRESSIVE-MOVING
C     AVERAGE PROCESS, EXPRESSED AS FACT * SUMSQ.
C
C     3/96 CAC  FROM STATLIB; CONVERTED TO DOUBLE PRECISION, NO TABS,
C     UPPERCASE, MIN --> MIN0, MAX --> MAX0 TO AGREE WITH JOURNAL
C     LISTING.  FIXED 4 BUGS (SEE COMMENTS WITH ORIGINAL BELOW).
C
C     TESTED SUCCESSFULLY ON THE FOLLOWING MODELS AND BENCHMARK RESULTS:
C      MA(1)  BOX-JENKINS(1976) SERIES A  -  ARIMA(0,1,1).
C      MA(2)  OSBORN(1976) USING 28-OBS SECTIONS OF B-J SERIES C
C      ARMA(1,1)  BOX-JENKINS(1976) SERIES A  (MEAN REMOVED FIRST).
C      ARMA(2,1)  DAVID CUSHMAN'S SERIES.  COMPARED WITH TSP PROC CODE
C                 THAT IMPLEMENTED NEWBOLD(1974) METHOD.
C      ARMA(3,1), ARMA(4,1), ARMA(5,1)   COMPARED WITH GAUSS RESULTS USING
C                 ANSLEY(1979) METHOD, WHICH IS NOT COMPLETELY EXACT ML
C                 (IS MISSING THE QUADRATIC FORM IN THE INITIAL RESIDUALS
C                  MENTIONED IN JUDGE, ET AL (1980) THEORY AND PRACTICE).
C
C     ORIGINAL CODE FROM STATLIB USED P(MP),Q(MQ) BELOW, BUT THIS
C     FAILS ON COMPUTERS LIKE VAX/VMS WHEN MP OR MQ ARE ZERO.
      DIMENSION P(1),Q(1),W(N),E(N),VW(MRP1),VL(MRP1),VK(MR)
C
      DATA EPSIL1 /1.D-10/
      DATA ZERO, P0625, ONE, TWO, FOUR, SIXTEN /
     *   0.D0, 0.0625D0, 1.D0, 2.D0, 4.D0, 16.D0/
C
      FACT = ZERO
      DETMAN = ONE
      DETCAR = ZERO
      SUMSQ = ZERO
      MXPQ = MAX0(MP, MQ)
      MXPQP1 = MXPQ + 1
      MQP1 = MQ + 1
      MPP1 = MP + 1
C
C      CALCULATION OF THE AUTOCOVARIANCE FUNCTION OF A PROCESS WITH
C      UNIT INNOVATION VARIANCE (VW) AND THE COVARIANCE BETWEEN THE
C      VARIABLE AND THE LAGGED INNOVATIONS (VL).
C
      CALL TWACF(P,MP,Q,MQ,VW,MXPQP1,VL,MXPQP1,VK,MXPQ,IFAULT)
      IF (MR .NE. MAX0(MP, MQP1)) IFAULT = 6
      IF (MRP1 .NE. MR + 1) IFAULT = 7
      IF (IFAULT .GT. 0) RETURN
C
C      COMPUTATION OF THE FIRST COLUMN OF MATRIX P (VK)
C
      VK(1) = VW(1)
      IF (MR .EQ. 1) GO TO 150
      DO 140 K = 2, MR
        VK(K) = ZERO
        IF (K .GT. MP) GO TO 120
        DO 110 J = K, MP
          JP2MK = J + 2 - K
C          ORIGINAL CODE FROM STATLIB
C          VK(K) = VK(K) + P(J) + VW(JP2MK)
```

```
C          FIXED TO AGREE WITH JOURNAL LISTING AND EQN (10.2)
           VK(K) = VK(K) + P(J) * VW(JP2MK)
  110    CONTINUE
  120    IF (K .GT. MQP1) GO TO 140
         DO 130 J = K, MQP1
           JP1MK = J + 1 - K
           VK(K) = VK(K) - Q(J-1) * VL(JP1MK)
  130    CONTINUE
  140 CONTINUE
C
C      COMPUTATION OF THE INITIAL VECTORS L AND K (VL, VK).
C
  150 R = VK(1)
      VL(MR) = ZERO
      DO 160 J = 1, MR
        VW(J) = ZERO
        IF (J .NE. MR) VL(J) = VK(J+1)
        IF (J .LE. MP) VL(J) = VL(J) + P(J) * R
        VK(J) = VL(J)
  160 CONTINUE
C
C      INITIALIZATION
C
      LAST = MPP1 - MQ
      LOOP = MP
      JFROM = MPP1
      VW(MPP1) = ZERO
      VL(MXPQP1) = ZERO
C
C      EXIT IF NO OBSERVATION, ELSE LOOP ON TIME.
C
      IF (N .LE. 0) GO TO 500
      DO 290 I = 1, N
C
C      TEST FOR SKIPPED UPDATING
C
        IF (I .NE. LAST) GO TO 170
        LOOP = MIN0(MP, MQ)
        JFROM = LOOP + 1
C
C      TEST FOR SWITCHING
C
        IF (MQ .LE. 0) GO TO 300
  170   IF (R .LE. EPSIL1) GO TO 400
        IF (DABS(R - ONE) .LT. TOLER .AND. I .GT. MXPQ) GO TO 300
C
C      UPDATING SCALARS
C
        DETMAN = DETMAN * R
  190   IF (DABS(DETMAN) .LT. ONE) GO TO 200
        DETMAN = DETMAN * P0625
        DETCAR = DETCAR + FOUR
        GO TO 190
  200   IF (DABS(DETMAN) .GE. P0625) GO TO 210
        DETMAN = DETMAN * SIXTEN
        DETCAR = DETCAR - FOUR
        GO TO 200
  210   VW1 = VW(1)
        A = W(I) - VW1
        E(I) = A / DSQRT(R)
        AOR = A / R
```

```
          SUMSQ = SUMSQ + A * AOR
          VL1 = VL(1)
          ALF = VL1 / R
          R = R - ALF * VL1
          IF (LOOP .EQ. 0) GO TO 230
C
C       UPDATING VECTORS
C
          DO 220 J = 1, LOOP
            FLJ = VL(J+1) + P(J) * VL1
            VW(J) = VW(J+1) + P(J) * VW1 + AOR * VK(J)
            VL(J) = FLJ - ALF * VK(J)
            VK(J) = VK(J) - ALF * FLJ
  220     CONTINUE
  230     IF (JFROM .GT. MQ) GO TO 250
          DO 240 J = JFROM, MQ
            VW(J) = VW(J+1) + AOR * VK(J)
            VL(J) = VL(J+1) - ALF * VK(J)
            VK(J) = VK(J) - ALF * VL(J+1)
  240     CONTINUE
  250     IF (JFROM .GT. MP) GO TO 270
          DO 260 J = JFROM, MP
  260     VW(J) = VW(J+1) + P(J) * W(I)
  270     CONTINUE
  290   CONTINUE
        GO TO 390
C
C       QUICK RECURSIONS
C
  300   NEXTI = I
        IFAULT = -NEXTI
        DO 310 I = NEXTI, N
  310   E(I) = W(I)
        IF (MP .EQ. 0) GO TO 340
        DO 330 I = NEXTI, N
          DO 320 J = 1, MP
            IMJ = I - J
            E(I) = E(I) - P(J) + W(IMJ)
  320     CONTINUE
  330   CONTINUE
  340   IF (MQ .EQ. 0) GO TO 370
        DO 360 I = NEXTI, N
          DO 350 J = 1, MQ
            IMJ = I - J
            E(I) = E(I) + Q(J) * E(IMJ)
  350     CONTINUE
  360   CONTINUE
C
C       RETURN SUM OF SQUARES AND DETERMINANT
C
  370   DO 380 I = NEXTI, N
  380   SUMSQ = SUMSQ + E(I) * E(I)
C
C       CODE FOR CONDITIONAL SUM OF SQUARES.
C       REPLACES ALL EXECUTABLE UP TO AND INCLUDING THAT LABELLED 380.
C
C             FACT = ZERO
C             DETMAN = ONE
C             DETCAR = ZERO
C             SUMSQ = ZERO
C             MXPQ = MAX0(MP, MQ)
```

```
C            DO 380 I = MXPQ, N
C              E(I) = W(I)
C              IF (MP .LE. 0) GO TO 340
C              DO 320 J = 1, MP
C                IMJ = I - J
C                E(I) = E(I) - P(J) * W(IMJ)
C        320        CONTINUE
C        340        IF (MQ .LE. 0) GO TO 380
C              DO 350 J = 1, MQ
C                IMJ = I - J
C                E(I) = E(I) + Q(J) * E(IMJ)
C        350        CONTINUE
C        380      SUMSQ = SUMSQ + E(I) * E(I)
C
  390 FN = N
      FACT = DETMAN ** (ONE / FN) * TWO ** (DETCAR / FN)
      RETURN
C
C      EXECUTION ERRORS
C
  400 IFAULT = 8
      RETURN
  500 IFAULT = 9
      RETURN
      END
C-----------------------
      SUBROUTINE TWACF(P,MP,Q,MQ,ACF,MA,CVLI,MXPQP1,ALPHA,MXPQ,IFAULT)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
C
C      ALGORITHM AS 197.1 APPLIED STATISTICS (1984) VOL.33, NO. 1
C
C      IMPLEMENTATION OF THE ALGORITHM OF G. TUNNICLIFFE WILSON
C      (J. STATIST. COMPUT. SIMUL. 8, 1979, 301-309) FOR THE COMPUTATION
C      OF THE AUTOCOVARIANCE FUNCTION (ACF) OF AN ARMA PROCESS OF ORDER
C      (MP, MQ) AND UNIT INNOVATION VARIANCE.   THE AUTOREGRESSIVE AND
C      MOVING AVERAGE COEFFICIENTS ARE STORED IN VECTORS P AND Q, USING
C      BOX AND JENKINS NOTATION.   ON OUTPUT, VECTOR CVLI CONTAINS THE
C      COVARIANCES BETWEEN THE VARIABLE AND THE (K-1)-LAGGED INNOVATION
C      FOR K = 1, ..., MQ+1.
C
C     ORIGINAL CODE FROM STATLIB USED P(MP),Q(MQ) BELOW, BUT THIS
C     FAILS ON COMPUTERS LIKE VAX/VMS WHEN MP OR MQ ARE ZERO.
      DIMENSION P(1), Q(1), ACF(MA), CVLI(MXPQP1), ALPHA(MXPQ)
C
      DATA EPSIL2 /1.D-10/
      DATA ZERO, HALF, ONE, TWO /0.D0, 0.5D0, 1.D0, 2.D0/
C
      IFAULT = 0
      IF (MP .LT. 0 .OR. MQ .LT. 0) IFAULT = 1
      IF (MXPQ .NE. MAX0(MP, MQ)) IFAULT = 2
      IF (MXPQP1 .NE. MXPQ + 1) IFAULT = 3
      IF (MA .LT. MXPQP1) IFAULT = 4
      IF (IFAULT .GT. 0) RETURN
C
C      INITIALIZATION, AND RETURN IF MP = MQ = 0
C
      ACF(1) = ONE
      CVLI(1) = ONE
      IF (MA .EQ. 1) RETURN
      DO 10 I = 2, MA
   10 ACF(I) = ZERO
```

```
      IF (MXPQP1 .EQ. 1) RETURN
      DO 20 I = 2, MXPQP1
   20 CVLI(I) = ZERO
      DO 90 K = 1, MXPQ
   90 ALPHA(K) = ZERO
C
C      COMPUTATION OF THE A.C.F. OF THE MOVING AVERAGE PART,
C      STORED IN ACF.
C
      IF (MQ .EQ. 0) GO TO 180
      DO 130 K = 1, MQ
        CVLI(K+1) = -Q(K)
        ACF(K+1) = -Q(K)
        KC = MQ - K
        IF (KC .EQ. 0) GO TO 120
        DO 110 J = 1, KC
          JPK = J + K
          ACF(K+1) = ACF(K+1) + Q(J) * Q(JPK)
  110   CONTINUE
  120   ACF(1) = ACF(1) + Q(K) * Q(K)
  130 CONTINUE
C
C      INITIALIZATION OF CVLI = T.W.-S PHI -- RETURN IF MP = 0.
C
  180 IF (MP .EQ. 0) RETURN
      DO 190 K = 1, MP
        ALPHA(K) = P(K)
        CVLI(K) = P(K)
  190 CONTINUE
C
C      COMPUTATION OF T.W.-S ALPHA AND DELTA (DELTA STORED IN ACF
C      WHICH IS GRADUALLY OVERWRITTEN).
C
      DO 290 K = 1, MXPQ
        KC = MXPQ - K
        IF (KC .GE. MP) GO TO 240
        DIV = ONE - ALPHA(KC+1) * ALPHA(KC+1)
        IF (DIV .LE. EPSIL2) GO TO 700
        IF (KC .EQ. 0) GO TO 290
        DO 230 J = 1, KC
          KCP1MJ = KC + 1 - J
          ALPHA(J) = (CVLI(J) + ALPHA(KC+1) * CVLI(KCP1MJ)) / DIV
  230   CONTINUE
  240   IF (KC .GE. MQ) GO TO 260
        J1 = MAX0(KC + 1 - MP, 1)
        DO 250 J = J1, KC
          KCP1MJ = KC + 1 - J
C          ORIGINAL CODE FROM STATLIB
C          ACF(J+1) = ACF(J+1) + ACF(K+2) * ALPHA(KCP1MJ)
C          FIXED TO AGREE WITH JOURNAL LISTING
          ACF(J+1) = ACF(J+1) + ACF(KC+2) * ALPHA(KCP1MJ)
  250   CONTINUE
  260   IF (KC .GE. MP) GO TO 290
        DO 270 J = 1, KC
  270   CVLI(J) = ALPHA(J)
  290 CONTINUE
C
C      COMPUTATION OF T.W.-S NU (NU IS STORED IN CVLI, COPIED INTO
C      ACF).
C
      ACF(1) = HALF * ACF(1)
```

```
      DO 330 K = 1, MXPQ
        IF (K .GT. MP) GO TO 330
        KP1 = K + 1
        DIV = ONE - ALPHA(K) * ALPHA(K)
        DO 310 J = 1, KP1
          KP2MJ = K + 2 - J
          CVLI(J) = (ACF(J) + ALPHA(K) * ACF(KP2MJ)) / DIV
  310   CONTINUE
        DO 320 J = 1, KP1
  320   ACF(J) = CVLI(J)
  330 CONTINUE
C
C      COMPUTATION OF ACF (ACF IS GRADUALLY OVERWRITTEN).
C
      DO 430 I = 1, MA
        MIIM1P = MIN0(I-1, MP)
        IF (MIIM1P .EQ. 0) GO TO 430
        DO 420 J = 1, MIIM1P
          IMJ = I - J
          ACF(I) = ACF(I) + P(J) * ACF(IMJ)
  420   CONTINUE
  430 CONTINUE
      ACF(1) = ACF(1) * TWO
C
C      COMPUTATION OF CVLI - RETURN WHEN MQ = 0.
C
      CVLI(1) = ONE
      IF (MQ .LE. 0) GO TO 600
      DO 530 K = 1, MQ
        CVLI(K+1) = -Q(K)
        IF (MP .EQ. 0) GO TO 530
        MIKP = MIN0(K, MP)
        DO 520 J = 1, MIKP
          KP1MJ = K + 1 - J
          CVLI(K+1) = CVLI(K+1) + P(J) * CVLI(KP1MJ)
  520   CONTINUE
  530 CONTINUE

  600 RETURN
C
C      EXECUTION ERROR DUE TO (NEAR) NON-STATIONARITY.
C
  700 IFAULT = 5
      RETURN
      END
```

```
      SUBROUTINE PROBS(K, W, P, IFAULT)
C
C        ALGORITHM AS 198 APPL. STATIST. (1984) VOL.33,NO.1
C
C        CALCULATION OF THE PROBABILITIES P(L,K) FOR THE CASE OF
C        SIMPLE ORDER.
C
C     Auxiliary functions required: PR1 & F1 from AS 158
C
      DIMENSION W(20), P(20), Q(20, 20), CH(20, 20)
      REAL ZERO, HALF
      DATA C1 /1.0E-6/, ZERO/0.0/, HALF/0.5/
C
C        CHECK THAT K .GE. 3 AND .LE. 20
C
      IFAULT = 2
      IF (K .LT. 3 .OR. K .GT. 20) RETURN
C
C        CHECK THAT THE WEIGHTS ARE POSITIVE
C
      IFAULT = 1
      DO 1 I = 1, K
      IF (W(I) .LE. ZERO) RETURN
    1 CONTINUE
C
C        CHECK WEIGHTS FOR EQUALITY
C
      IFAULT = 0
      WW = W(1)
      DO 2 I = 2, K
        IF (ABS(WW - W(I)) .GT. C1) GOTO 7
    2 CONTINUE
C
C        EQUAL WEIGHTS
C
      CALL CHASE(K, Q, .TRUE.)
      DO 3 J = 1, K
    3 P(J) = A(J, K)
      RETURN
C
C        UNEQUAL WEIGHTS
C
    7 IF (K .GT. 5) GOTO 11
      K2 = K - 2
      GOTO (8, 9, 10), K2
    8 P(1) = PR1(1, 3, W)
      P(2) = HALF
      P(3) = HALF - P(1)
      RETURN
    9 P(1) = PR1(1, 4, W)
      P(4) = PR1(4, 4, W)
      P(2) = HALF - P(4)
      P(3) = HALF - P(1)
      RETURN
   10 P(5) = PR1(5, 5, W)
      P(4) = PR1(4, 5, W)
      P(2) = HALF - P(4)
      P(1) = PR1(1, 5, W)
      P(3) = HALF - P(1) - P(5)
      RETURN
   11 CALL CHASE(K, CH, .FALSE.)
```

```
      CALL CHASE(K, Q, .TRUE.)
      CALL PAPRX(K, W, CH, Q, P)
      RETURN
      END
C
      SUBROUTINE PAPRX(K, W, CH, P, PA)
C
C        ALGORITHM AS 198.1 APPL. STATIST. (1984) VOL.33, NO.1
C
C        THIS SUBROUNTINE COMPUTES THE APPROXIMATE PLK-S.
C
      REAL W(20), CH(20, 20), P(20, 20), PA(20), PB(20), PBP(20),
     *  SUMS(2), ZERO, ONE, THREE, PT65, PT35
      INTEGER INDEX(20), A, B
      DATA ZERO/0.0/, ONE/1.0/, THREE/3.0/, PT65/0.65/, PT35/0.35/
C
      ALPHA = ONE / THREE
C
C        INITIALISE PA,PB,PBP
C
      DO 10 I = 1, 20
        PA(I) = ZERO
        PB(I) = ZERO
        PBP(I) = ZERO
   10 CONTINUE
C
C         DETERMINE MASIMUM AND MINIMUM OF WEIGHTS
C
      WMAX = W(1)
      WMIN = W(1)
      DO 20 I = 2, K
        WW = W(I)
        IF (WW .LT. WMIN) WMIN = WW
        IF (WW .LT. WMAX) WMAX = WW
   20 CONTINUE
      CUT = PT65 * WMIN + PT35 * WMAX
C
C        DETERMINE THE INDICES OF THE WEIGHTS AND THE NUMBER OF
C        LARGE WEIGHTS (M)
C
      M = 0
      DO 40 I = 1, K
        IF (W(I) .LT. CUT) GOTO 30
        INDEX(I) = 1
        M = M + 1
        GOTO 40
   30   INDEX(I) = 0
   40 CONTINUE
C
C        IF A=0 AND B=0 SET PB=PLM
C
      N = INDEX(1) + INDEX(K)
      IF (N .NE. 2) GO TO 90
      DO 80 L = 1, M
   80 PB(L) = P(L, M)
      GOTO 240
C
C        DETERMINE A,B
C
   90 A = 1
      B = 1
```

```
      DO 100 I = 1, K
        IF (INDEX(I) .NE. 0) GOTO 110
        A = A + 1
  100 CONTINUE
  110 DO 120 I = 1, K
        J = K - I + 1
        IF (INDEX(J) .NE. 0) GOTO 130
        B = B + 1
  120 CONTINUE
  130 IF (N .EQ. 0) GOTO 190
      N = MAX0(A, B)
      DO 180 L = 1, K
        SUM = ZERO
        DO 170 I = 1, L
          J = L - I + 1
          SUM = SUM + P(I, M) * CH(J, N)
  170   CONTINUE
        PB(L) = SUM
  180 CONTINUE
      GOTO 240
  190 DO 210 L = 1, K
        SUM = ZERO
        DO 200 I = 1, L
          J = L - I + 1
          SUM = SUM + CH(I, A) * CH(J, B)
  200   CONTINUE
        PBP(L) = SUM
  210 CONTINUE
      DO 230 L = 1, K
        SUM = ZERO
        DO 220 I = 1, L
          J = L - I + 1
          SUM = SUM + PBP(I) * P(J, M)
  220   CONTINUE
        PB(L) = SUM
  230 CONTINUE
C
C       DETERMINE R
C
  240 SUMS(1) = ZERO
      SUMS(2) = ZERO
      DO 250 I = 1, K
        J = INDEX(I) + 1
        SUMS(J) = SUMS(J) + W(I)
  250 CONTINUE
      X = K - M
      Y = M
      R = SUMS(2) * X / (SUMS(1) * Y)
      RI = ONE / R ** ALPHA
      DO 260 L = 1, K
  260 PA(L) = (ONE - RI) * PB(L) + RI * P(L, K)
      RETURN
      END
C
      SUBROUTINE CHASE(K, CH, EQUAL)
C
C       ALGORITHM AS 198.2 APPL. STATIST. (1984) VOL.33, NO.1
C
C       THIS SUBROUTINE COMPUTES CHASE PLK-S IF EQUAL=.FALSE.
C       OR EQUAL WEIGHT PROBABILITIES IF EQUAL=.TRUE.
C
```

```
      LOGICAL EQUAL
      REAL ZERO, HALF, ONE, CH(20, 20)
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/
C
      DO 20 J = 1, K
        DO 10 I = 1, K
   10   CH(I, J) = ZERO
   20 CONTINUE
      CH(1, 1) = ONE
      CH(1, 2) = HALF
      CH(2, 2) = HALF
      KM = K - 1
      DO 60 J = 1, KM
        J1 = J + 1
        IF (EQUAL) GOTO 30
        X = J + J - 1
        Y = J + J
        GOTO 40
   30   X = J
        Y = J1
   40   Y1 = ONE / Y
        CH(1, J1) = X * Y1 * CH(1, J)
        CH(J1, J1) = Y1 * CH(J, J)
        DO 50 I = 2, J
          CH(I, J1)  = Y1 * CH(I - 1, J) + X * Y1 * CH(I, J)
   50   CONTINUE
   60 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE BAB(N, M, SI, T, S, A, MAX, CRIT, BEST, NEVAL, IFAULT)
C
C     ALGORITHM AS  199  APPL. STATIST. (1984) VOL.33, NO.2
C
C     Efficient branch and bound algorithm for determining the optimal
C     feature subset of given size.
C
C     Incorporates a corrected version of the corrections in ASR 67.
C     The second of those corrections caused a jump into a DO loop.
C
      REAL SI(N,N), T(N), S(N,N,N), CRIT, ZERO, D
      INTEGER A(N), MAX(M), BEST(M), NEVAL(N), AI8, AJ8, OMIT
      LOGICAL SKIP
C
      DATA ZERO /0.0/
C
C     Check for invalid values of M and N
C
      IFAULT = 1
      IF (M .LT. 2 .OR. N .LE. M) RETURN
      IFAULT = 0
C
C     Initialize arrays - A holds current M-subset, MAX holds upper
C     limit for possible member features in M-subset.
C     Initial optimal criterion is CRIT = 0
C     SKIP = .FALSE. indicates that current position on solution tree
C     follows consecutively in enumeration sequence rather than a jump
C     from a rejected branch.
C
      DO 4 I = 1, N
        DO 3 J = 1, N
    3   S(I,J,N) = SI(I,J)
        NEVAL(I) = 0
    4 CONTINUE
      SKIP = .FALSE.
      CRIT = ZERO
      DO 5 I = 1, M
        A(I) = I
        MAX(I) = N - M + I
        BEST(I) = 0
    5 CONTINUE
   10 K1 = A(M)
      K2 = MAX(M)
      DO 30 I = K1, K2
C
C     Enumerate members of next subset
C
        DO 13 J = I, N
          JJ = M - I + J
          A(JJ) = J
   13   CONTINUE
        MM = M + N - I
C
C     If subset of size N, skip to calculation of criterion for
C     subsequent descendent subsets.
C     If consecutive enumeration (SKIP = .FALSE.) next feature omitted
C     is A(M).
C     If following jump in enumeration sequence (SKIP = .TRUE.) next
C     feature omitted is A(M-L).
C
        IF (MM .EQ. N) GO TO 22
```

```
            OMIT = M
            IF (.NOT. SKIP) GO TO 15
            OMIT = M - L
            SKIP = .FALSE.
C
C     Evaluate criterion function for current subset from that of
C     previous subset, excluding feature A(OMIT).
C
   15     M9 = MM + 1
            D = ZERO
            DO 17 I8 = 1, MM
              I9 = I8
              AI8 = A(I8)
              IF (I8 .GE. OMIT) I9 = I8 + 1
              DO 16 J8 = 1, MM
                J9 = J8
                AJ8 = A(J8)
                IF (J8 .GE. OMIT) J9 = J8 + 1
                S(I8,J8,MM) = S(I9,J9,M9) - S(OMIT,I9,M9) * S(OMIT,J9,M9) /
     *                        S(OMIT,OMIT,M9)
                D = D + T(AI8) * T(AJ8) * S(I8,J8,MM)
   16         CONTINUE
   17       CONTINUE
            NEVAL(MM) = NEVAL(MM) + 1
C
C     Check criteraion of current subset, D, relative to optimal value
C     CRIT.
C
            IF (D .GT. CRIT .AND. MM .EQ. M) GO TO 28
            IF (D .GT. CRIT) GO TO 22
C
C     Optimal value not exceeded, so abandon downward search of branch.
C     Prepare for return to next lower level branching mode not yet
C     explored.   SKIP = .TRUE. indicates jump to lower level before
C     termination of branch.
C
   60     M1 = M - 1
            DO 18 L = 1, M1
              ML = M - L
              IF (A(ML) .LT. A(ML+1) - 1) GO TO 19
   18       CONTINUE
            L = M
   19       DO 20 LL = 1, L
            ML = M - LL + 1
            A(ML) = MAX(ML)
   20       CONTINUE
            SKIP = .TRUE.
            GO TO 35
C
C     Optimal value CRIT exceeded by lower level subset.   Reduce by
C     one feature at a time until either criterion falls below CRIT or
C     subset size reaches M.
C
   22     MX = MM - M
            DO 27 MO = 1, MX
              D = ZERO
              M8 = MM - MO
              M9 = M8 + 1
              DO 24 I8 = 1, M8
                I9 = I8
                IF (I8 .GT. M) I9 = I8 + 1
```

```
              A(I8) = A(I9)
              DO 23 J8 = 1, M8
                J9 = J8
                IF (J8 .GT. M) J9 = J8 + 1
                S(I8,J8,M8) = S(I9,J9,M9) - S(M+1,I9,M9) * S(M+1,J9,M9) /
     *                        S(M+1,M+1,M9)
   23         CONTINUE
   24       CONTINUE
            DO 26 I8 = 1, M8
              AI8 = A(I8)
              DO 25 J8 = 1, M8
                AJ8 = A(J8)
                D = D + T(AI8) * T(AJ8) * S(I8,J8,M8)
   25         CONTINUE
   26       CONTINUE
            NEVAL(M8) = NEVAL(M8) + 1
            IF (D .LE. CRIT) GO TO 30
   27     CONTINUE
C
C       Update optimal criterion function, CRIT, and corresponding
C       M-subset BEST(.).
C
   28     CRIT = D
          DO 29 L1 = 1, M
   29     BEST(L1) = A(L1)
          IF (I .EQ. K2) GO TO 60
   30   CONTINUE
C
C       Update feature subset required for return to lower level branching
C       mode of tree.   Search complete if first member A(1) has taken all
C       its possible feature values.
C
   35   DO 50 I = 2, M
          II = M - I + 1
          IF (A(II) .EQ. MAX(II)) GO TO 50
          A(II) = A(II) + 1
          I1 = II + 1
          DO 40 J = I1, M
   40     A(J) = A(J-1) + 1
          GO TO 10
   50   CONTINUE
C
C       Successful completion of search
C
        RETURN
        END
```

```
      SUBROUTINE SUMSQ(N, APPROX, BL, BU, IFAULT)
C
C     ALGORITHM AS 200  APPL. STATIST. (1984) VOL.33, NO.2
C
C     Approximates the sum of squares of normal scores and gives bounds
C     for the truncation error involved in the approximation.
C
C     Coefficients SA(K) are the squares of the Fourier coefficients
C     A(2*K-1).
C
      REAL APPROX, BL, BU, SA(14), ZERO, ONE, PROD, RI, RK, RN, SUM1,
     *     SUM2
C
      DATA SA/
     *  0.95492 96583, 0.03349 20160, 0.00667 47227, 0.00227 80925,
     *  0.00101 63627, 0.00053 26443, 0.00031 08437, 0.00019 58978,
     *  0.00013 08187, 0.00009 13941, 0.00006 63703, 0.00004 96799,
     *  0.00004 11471, 0.00003 49349/
      DATA ZERO/0.0/, ONE/1.0/, NMAX/27/
C
C     Value of zero is arbitrarily given in case of failure
C
      APPROX = ZERO
      BL = ZERO
      BU = ZERO
C
C     Check consistency of input parameter
C
      IFAULT = 1
      IF (N .LE. 1) RETURN
      IFAULT = 0
C
      RN = N
      NM = NMAX - 2
      IF (N .GT. NMAX) GO TO 1
      M = N / 2
      NM = 2 * M - 1
    1 SUM1 = ZERO
      SUM2 = ZERO
      KK = 0
      DO 3 K = 1, NM, 2
        RK = K
        PROD = ONE
        DO 2 I = 1, K
          RI = I
          PROD = PROD * (RN - RI) / (RN + RK - RI + ONE)
    2   CONTINUE
        KK = KK + 1
        PROD = PROD * RN
        SUM1 = SUM1 + SA(KK) * PROD
        SUM2 = SUM2 + SA(KK)
    3 CONTINUE
C
C     Approximate value is computed
C
      APPROX = SUM1
      IF (N .LE. NMAX) RETURN
      PROD = ONE
      DO 4 I = 1, NMAX
        RI = I
        PROD = PROD * (RN - RI) / (RN + RI)
```

```
      4 CONTINUE
        PROD = PROD * RN
C
C     Lower & upper bounds for the truncation error are computed
C
        BL = PROD * SA(14)
        BU = PROD * (ONE - SUM2)
C
        RETURN
        END
```

```
      SUBROUTINE COMICS(ALP, NP, P, TOL, PI, CPI, CP, SP, IFAULT)
C
C     ALGORITHM AS 201  APPL. STATIST. (1984) VOL.33, NO.2
C
C     Subroutine to combine predictions about a statistic based on the
C     orderings of a set of means with an F-test to test for differences
C     between the means.
C
      INTEGER NP, IFAULT, I, J
      REAL ALP, P(NP), TOL, PI(NP), CPI(NP), CP(NP), SP(NP), ZERO, ONE,
     *     CUM, PIJ, X, XALP, ABS
C
      DATA ZERO /0.0/, ONE /1.0/
C
C     Test ALP
C
      IFAULT = 1
      IF (ALP .LE. ZERO .OR. ALP .GE. ONE) RETURN
C
C     Compute cumulative distribution of statistic
C
      IFAULT = 2
      CP(1) = P(1)
      IF (P(1) .LE. ZERO) RETURN
      DO 10 I = 2, NP
        IF (P(I) .LE. ZERO) RETURN
        CP(I) = CP(I-1) + P(I)
   10 CONTINUE
C
C     Check that distribution sums to one
C
      IFAULT = 3
      IF (ABS(CP(NP) - ONE) .GT. TOL) RETURN
      IFAULT = 0
C
C     Compute distribution of combined likelihoods (PI) and cumulative
C     combined distribution (CPI)
C
      CUM = ZERO
      DO 30 J = 1, NP
        PIJ = ZERO
        DO 20 I = J, NP
   20   PIJ = PIJ + P(I) / CP(I)
        PI(J) = P(J) * PIJ
        CUM = CUM + PI(J)
        CPI(J) = CUM
   30 CONTINUE
C
C     Determine criteria for significance on combined dist X
C
      DO 40 I = 1, NP
        IF (CPI(I) .GT. ALP) GO TO 50
   40 CONTINUE
      I = NP
   50 XALP = CPI(I) - ALP
      X = -XALP * P(I) / PI(I) + CP(I)
C
C     Determine significance level required of F
C
      DO 60 I = 1, NP
        SP(I) = X / CP(I)
```

```
            IF (SP(I) .GT. ONE) SP(I) = ONE
       60 CONTINUE
C
          RETURN
          END
```

```
      SUBROUTINE MODLIK(NN, NNP2, Y, DEL, ALPHA, BETA, DELTA1, DELTA2,
     *  NGR, K, INK, NK, NFAIL, H, A, VML, CUMCEN, DAT, IFAULT)
C
C     ALGORITHM AS 202  APPL. STATIST. (1984) VOL.33, NO.2
C
C     MODLIK uses cross-validation to select parameters for the
C     3-parameter nonparametric hazard estimator.
C
      INTEGER NN, NNP2, DEL(NN), NGR, K, INK, NK, NFAIL, CUMCEN(NN),
     *  IFAULT, I, IM, ITEST, J, JM, JM1, JP1, KTH, KTH2, LKTH, MAXIT,
     *  NJ1, NT
      REAL Y(NN), ALPHA, BETA, DELTA1, DELTA2, H(NN,NN), A(NN,NN), VML,
     *  DAT(NNP2), ZERO, EPS2, BIG, ALEPH1, ALEPH2, ALEPH3, BET1, BET2,
     *  BET3, DHAA, FMAB, FMAX, SS, TEST
C
      DATA ZERO, EPS2, BIG /0.0, 1.0E-04, 1.0E25/, MAXIT/10/
C
      NT = NN - 1
C
C     Check for input faults
C
      NFAIL = 0
      IFAULT = 1
      IF (NN .LE. 0 .OR. ALPHA .LE. ZERO .OR. BETA .LE. ZERO .OR.
     *  DELTA1 .LE. ZERO .OR. DELTA2 .LE. ZERO .OR. NGR .LE. 0 .OR.
     *  K .LE. 0 .OR. INK .LE. 0 .OR. NK .LE. 0 .OR. NNP2 .NE. NN+2)
     *  RETURN
      IFAULT = 2
      DO 25 I = 1, NN
        J = DEL(I)**2
        IF (Y(I) .LT. ZERO .OR. J .NE. DEL(I)) RETURN
        NFAIL = NFAIL + DEL(I)
   25 CONTINUE
      DHAA = NFAIL
      IFAULT = 3
      DO 30 I = 1, NT
        IF (Y(I) .GT. Y(I+1)) RETURN
   30 CONTINUE
      IFAULT = 4
      LKTH = K + (NK-1)*INK
      ITEST = NFAIL - LKTH - 1
      IF (ITEST .LE. 0) RETURN
      IM = 1
      JM = 2
      DO 55 I = 1, NT
        KTH = 0
   35   IF (Y(IM) .LT. Y(JM) .OR. DEL(IM) .LT. DEL(JM)) GO TO 45
        IF (DEL(IM) .GT. DEL(JM)) RETURN
        IF (DEL(JM) .LT. 1) GO TO 45
        KTH = KTH + 1
        IF (K .LE. KTH) RETURN
        IF (JM .GE. NN) GO TO 60
        JM = JM + 1
        GO TO 35
   45   IM = JM
        IF (JM .GE. NN) GO TO 60
        JM = JM + 1
   55 CONTINUE
C
C     Set up the A matrix
C
```

```
   60 SS = DEL(1) / FLOAT(NN-1)
      DO 65 J = 2, NN
        NJ1 = NN - J + 1
        A(1,J) = SS
        A(NN,NJ1) = DEL(NN)
   65 CONTINUE
      DO 95 J = 2, NT
        SS = DEL(J) / FLOAT(NN-J)
        JP1 = J + 1
        DO 75 I = JP1, NN
   75   A(J,I) = SS
        JM1 = J - 1
        SS = DEL(J) / FLOAT(NN-JM1)
        DO 85 I = 1, JM1
   85   A(J,I) = SS
   95 CONTINUE
      DO 99 J = 1, NN
   99 A(J,J) = ZERO
C
C     Locate the maximizer of the log-likelihood function.
C     For a given value of K, evaluate the log-modified likelihood
C     function over a grid of alpha and beta points, then use the
C     maximizer as the starting point for the N-R algorithm.
C
      FMAB = - BIG
      DO 250 KTH = K, LKTH, INK
        CALL KTHNN(NN, NNP2, DEL, Y, KTH, NFAIL, 1, H, CUMCEN, DAT)
        FMAX = - BIG
        DO 150 IM = 1, NGR
          ALEPH1 = ALPHA + (IM - 1) * DELTA1
          DO 100 JM = 1, NGR
            BET1 = BETA + (JM - 1) * DELTA2
            CALL EVDER(NN, DEL, Y, A, H, 0, ALEPH1, BET1, DHAA, VML,
     *                 TEST, IFAULT)
            IF (VML .LT. FMAX .OR. IFAULT .GT. 0) GO TO 100
            FMAX = VML
            ALEPH2 = ALEPH1
            BET2 = BET1
  100     CONTINUE
  150   CONTINUE
C
C     This loop controls the maximum number (MAXIT) of N-R iterations.
C
        DO 175 IM = 1, MAXIT
          CALL EVDER(NN, DEL, Y, A, H, 1, ALEPH2, BET2, DHAA, VML, TEST,
     *               IFAULT)
C
C     The following IF statement tests for non-positive values of alpha
C     and beta.   If the condition is true, user must specify another
C     grid search for this value of K.   Irrespective of the outcome of
C     this test, the program proceeds to a larger (valid) value of K.
C
          IF (IFAULT .EQ. 5) GO TO 250
C
C     Test if the algorithm has converged.
C
          IF (TEST .LT. EPS2) GO TO 200
  175   CONTINUE
C
C     Convergence achieved or maximum number of iterations performed.
C     Compare with previous maximizer vector.   Update vector if nec.
```

```
C
  200   IF (VML .LT. FMAB) GO TO 250
        FMAB = VML
        KTH2 = KTH
        ALEPH3 = ALEPH2
        BET3 = BET2
  250 CONTINUE
      K = KTH2
      ALPHA = ALEPH3
      BETA = BET3
      VML = FMAB
C
      RETURN
      END
C
C
      SUBROUTINE KTHNN(NN, NNP2, DEL, Y, K, NFAIL, IKTH, H, CUMCEN, DAT)
C
C     ALGORITHM AS 202.1  APPL. STATIST. (1984) VOL.33, NO.2
C
C     KTHNN computes the matrix H of K-th nearest neighbour distances.
C
      INTEGER NN, NNP2, DEL(NN), K, NFAIL, IKTH, CUMCEN(NN), CENTRE,
     *   FSD, I, IDUM, J, K1, KP1, LOW, NJ1, NR
      REAL Y(NN), H(NN,NN), DAT(NNP2), ZERO, BIG, DIFF, KNN, KP1NN
C
      DATA ZERO, BIG /0.0, 1.0E25/
C
      CENTRE = 2
      LOW = 2
      DAT(1) = - BIG
      DAT(NFAIL+2) = BIG
      KP1 = K + 1
      NR = NN
      IF (IKTH .LT. 1) NR = 1
C
C     Collapse the observation vector, but record the cumulative number
C     of censored observations between failures.
C
      J = 0
      K1 = 0
      DO 150 I = 1, NFAIL
   50   J = J + 1
        IF (DEL(J) .GT. 0) GO TO 100
        K1 = K1 + 1
        GO TO 50
  100   DAT(I+1) = Y(J)
        CUMCEN(I) = K1
  150 CONTINUE
C
C     This loop moves the point of interest to the right.
C
      DO 750 I = 1, NFAIL
C
C     Move the window to the right.
C
  180   IDUM = LOW + K + 1
        DIFF = (DAT(IDUM) - DAT(CENTRE)) - (DAT(CENTRE) - DAT(LOW))
        IF (DIFF .GT. ZERO) GO TO 250
        LOW = LOW + 1
        GO TO 180
```

```
C
C     Determine which edge is the K-th nearest neighbour.
C
  250   IDUM = LOW + K
        DIFF = (DAT(IDUM) - DAT(CENTRE)) - (DAT(CENTRE) - DAT(LOW))
        KNN = DAT(IDUM)
        IF (DIFF .LE. ZERO) KNN = DAT(LOW)
        IF (IKTH .GT. 0) GO TO 400
        H(I,1) = ABS(KNN - DAT(CENTRE))
        GO TO 725
  400   FSD = LOW + CUMCEN(LOW-1) - 1
C
C     Find the K+1st nearest neighbour.
C
        IDUM = LOW + K + 1
        DIFF = (DAT(IDUM) - DAT(CENTRE)) - (DAT(CENTRE) - DAT(LOW-1))
        KP1NN = DAT(IDUM)
        IF (DIFF .GT. ZERO) KP1NN = DAT(LOW-1)
C
C     Compute the K-th nearest neighbour matrix H.   A typical row
C     consists of three segments - pre-window, window and post-window.
C
        DIFF = ABS(KNN - DAT(CENTRE))
        IF (FSD .LE. 1) GO TO 625
        K1 = FSD - 1
        DO 600 J = 1, K1
  600   H(I,J) = DIFF
  625   K1 = 1
        DO 650 J = FSD, NN
          IF (DEL(J) .GT. 0) GO TO 635
          H(I,J) = DIFF
          GO TO 650
  635     H(I,J) = ABS(KP1NN - DAT(CENTRE))
          K1 = K1 + 1
          IF (K1 .GT. KP1) GO TO 675
  650   CONTINUE
  675   IF (J .GE. NN) GO TO 725
        K1 = J + 1
        DO 700 J = K1, NN
  700   H(I,J) = DIFF
  725   IF (CENTRE .LE. NFAIL) CENTRE = CENTRE + 1
  750 CONTINUE
C
C     Expand the H matrix.
C
      J = 0
      DO 850 K1 = 1, NFAIL
  775   J = J + 1
        NJ1 = NN - J + 1
        IF (DEL(NJ1) .LT. 1) GO TO 775
        IDUM = NFAIL - K1 + 1
        DO 800 I = 1, NR
  800   H(NJ1,I) = H(IDUM,I)
  850 CONTINUE
      DO 950 I = 1, NN
        IF (DEL(I) .GT. 0) GO TO 950
        DO 900 J = 1, NR
  900   H(I,J) = ZERO
  950 CONTINUE
      IF (IKTH .LT. 1) GO TO 990
      DO 975 I = 1, NN
```

```
      975 H(I,I) = ZERO
      990 RETURN
          END
C
C
          SUBROUTINE EVDER(NN, DEL, Y, A, H, IFLAG, ALPHA, BETA, DHAA, VML,
         *  TEST, IFAULT)
C
C     ALGORITHM AS 202.2  APPL. STATIST. (1984) VOL.33, NO.2
C
C     EVDER evaluates the log-modified-likelihood function (ML) and its
C     gradient vector and Hessian matrix.   If FLAG = 1, an N-R step is
C     performed.
C
          INTEGER NN, DEL(NN), IFLAG, IFAULT, I, J, KM
          REAL Y(NN), A(NN,NN), H(NN,NN), ALPHA, BETA, DHAA, VML, TEST,
         *  ZERO, EPS1, ONFI, ONTH, TWTH, FS, ONE, TWO, THREE, FF, FOUR,
         *  FIVE, TBIG, BIG, A1, B1, CT, D1, D0K, D1K, D2K, DHB, D2HB, DSA,
         *  G1, G2, SK, SDK, SD2K, T, T1, T2, X1, X3, X5
C
          DATA ZERO /0.0/, EPS1 /1.0E-4/, ONFI /0.2/, ONTH /0.3333333333/,
         *  TWTH /0.6666666667/, FS /0.9375/, ONE /1.0/, TWO /2.0/, THREE
         *  /3.0/, FF /3.75/, FOUR /4.0/, FIVE /5.0/, TBIG /350.0/, BIG
         *  /1.0E25/
C
          IFAULT = 5
          D0K = ZERO
          D1K = ZERO
          D2K = ZERO
          DHB = ZERO
          D2HB = ZERO
          VML = ZERO
C
C     Delete each observation, one at a time.
C
          DO 150 KM = 1, NN
            SK = ZERO
            SDK = ZERO
            SD2K = ZERO
C
C     Compute the contribution of the deleted point to ML and its
C     first two derivatives for fixed values of alpha and beta.
C
            DO 100 I = 1, NN
              IF (A(I,KM) .LE. ZERO) GO TO 100
              T = BETA * H(I,KM)
              T2 = MIN(Y(KM), Y(I) + T)
              T1 = MAX(ZERO, Y(I) - T)
              IF (T1 .GT. T2) GO TO 100
              A1 = (T2 - Y(I)) / T
              B1 = (T1 - Y(I)) / T
              X1 = A1 - B1
              X3 = A1**3 - B1**3
              X5 = A1**5 - B1**5
              D0K = D0K + (X1 - X3*TWTH + X5*ONFI) * A(I,KM)
              IF (IFLAG .LT. 1) GO TO 50
              D1K = D1K + (X3*ONTH - X5*ONFI)* A(I,KM)
              D2K = D2K + (X5 - X3) * A(I,KM)
   50         IF (DEL(KM) .LT. 1) GO TO 100
              A1 = (Y(KM) - Y(I)) / T
              A1 = A1**2
```

```
            IF (A1 .GT. ONE) GO TO 100
            SK = SK + (ONE - A1)**2 / H(I,KM) * A(I,KM)
            IF (IFLAG .LT. 1) GO TO 100
            SDK = SDK + (ONE - A1) * A1 / H(I,KM) * A(I,KM)
            SD2K = SD2K + (FIVE*A1 - THREE) * A1 / H(I,KM) * A(I,KM)
  100     CONTINUE
          IF (DEL(KM) .LT. 1) GO TO 150
C
C     If SK <= 0 restart the N-R algorithm.
C
          IF (SK .LE. ZERO) RETURN
          VML = VML + LOG(SK * FS / ALPHA)
          IF (IFLAG .LT. 1) GO TO 150
          DHB = DHB + SDK / SK
          D2HB = D2HB + SD2K/SK - (SDK/SK)**2 * FOUR
  150   CONTINUE
        DSA = D0K * FS * BETA / ALPHA**2
        VML = VML - DSA * ALPHA
        IF (IFLAG .LT. 1) GO TO 300
C
C     Compute the gradient vector (G1, G2), the Hessian matrix
C     (H11=A1, H12=H21=B1, H22=D1), and update alpha and beta.
C
        G1 = -DHAA/ALPHA + DSA
        G2 = -D1K*FF/ALPHA + DHB*FOUR/BETA
        A1 = DHAA/ALPHA**2 - D0K*FS*BETA*TWO/ALPHA**3
        B1 = D1K*FF/ALPHA**2
        D1 = D2HB*FOUR/BETA**2 - D2K*FF/(ALPHA*BETA)
        CT = A1*D1 - B1**2
        ALPHA = ALPHA - (G1*D1 - G2*B1)/CT
        BETA = BETA + (G1*B1 - G2*A1)/CT
        TEST = SQRT(G1**2 + G2**2)
C
C     Test for negative alpha and beta values.
C     If condition is true, the algorithm is diverging.
C     Also test for unbounded increase of beta.
C
        IF (BETA .LT. EPS1 .OR. ALPHA .LT. EPS1) RETURN
        IF (BETA .LT. TBIG) GO TO 300
C
C     If beta is large, compute alpha and VML analytically.
C
        SD2K = ZERO
        VML = ZERO
        DO 250 I = 1, NN
          SDK = ZERO
          DO 200 J = 1, NN
  200     IF (A(J,I) .GT. ZERO) SDK = SDK + A(J,I)/H(J,I)
          IF (DEL(I) .GT. 0) VML = VML + LOG(SDK * FS)
          SD2K = SD2K + SDK*Y(I)
  250   CONTINUE
        ALPHA = FS * SD2K / DHAA
        BETA = BIG
        VML = -DHAA - DHAA*LOG(ALPHA) + VML
        TEST = ZERO
C
  300   IFAULT = 0
        RETURN
        END
```

```
      subroutine mixture(a, k, m, c, x, n, alpha, mean, sd, tol, nobvs,
     +    logl, counter, putout, ifault)
c
c     Translation from Algol 60 by Alan Miller of AS 203.
c
c     ALGORITHM AS 203 APPL. STATIST. (1984) VOL.33, NO. 3
c
c     This algorithm calculates the maximum likelihood estimates of the
c     parameters of a mixture of normal or exponential or Poisson or
c     binomial distributions.   These parameters are the mixing
c     proportions, the means and (in the normal distribution case)
c     standard deviations.   It also calculates the log-likelihood
c     function and the number of iterations taken to satisfy the
c     tolerance value.
c
c     Users must provide their own routine `putout' which prints the
c     estimated parameter values after each iteration.   The form of
c     this routine is:
c        subroutine putout(k, alpha, mean, sd, logl)
c        integer k
c        real alpha(k), mean(k), sd(k), logl
c
c     Notes:
c     The original (Algol) variable names have been retained.   This
c     means that some variable names have more than 6 characters, such
c     as `counter', `newalpha', etc.
c     The authors treat the normal distribution as if it were a discrete
c     distribution.
c     In Fortran 77 it is necessary to give explicit dimensions to some
c     of the temporary arrays which are not passed as arguments.
c     Maximum values have been set for k and m (kmax and mmax) in the
c     parameter statement under `Local variables' below.
c
      integer a, k, m, c, n(m), nobvs, counter, ifault
      real x(m), alpha(k), mean(k), sd(k), tol, logl
      external putout
c
c     Local variables
c
      integer kmax, mmax, i, j
      parameter (kmax = 20, mmax = 100)
      logical test
      real sumalpha, part, oldlogl, newalpha(kmax), newmean(kmax),
     +    newsd(kmax), dt(kmax), nt(kmax), vt(kmax), g(mmax),
     +    f(mmax, kmax), zero, one, half
      data zero /0.0/, one /1.0/, half /0.5/
c
      if (a .lt. 1 .or. a .gt. 4) then
        ifault = 1
        return
      else
        ifault = 0
      end if

      do 10 i = 1, m-1
        if (x(i) .gt. x(i+1)) then
          ifault = 5
          return
        end if
   10 continue
```

```
         if (nobvs .lt. 2 * m) then
           ifault = 6
           return
         end if

         do 20 i = 1, m
           if (n(i) .lt. 0) then
             ifault = 6
             return
           end if
           if (a .ne. 1) then
             if (x(i) .lt. zero) then
               ifault = 7
               return
             end if
           end if
      20 continue

         oldlogl = zero
         counter = 0
   c
   c     Start of iterative cycle
   c
      30 counter = counter + 1
         do 40 j = 1, k
           if (alpha(j) .gt. one .or. alpha(j) .lt. zero) then
             ifault = 2
             return
           end if
           if (mean(j) .ge. x(m) .or. mean(j) .le. x(1)) then
             ifault = 3
             return
           end if
           if (a .eq. 1) then
             if (sd(j) .le. zero) then
               ifault = 4
               return
             end if
           end if
      40 continue

         do 60 i = 1, k-1
           do 50 j = i+1, k
             if (mean(i) .eq. mean(j)) then
               if (a .eq. 1) then
                 if (sd(i) .eq. sd(j)) then
                   ifault = 9
                   return
                 end if
               else
                 ifault = 8
                 return
               end if
             end if
      50   continue
      60 continue

         logl = zero
         do 80 i = 1, m
           g(i) = zero
           do 70 j = 1, k
```

```
c
c     a = 1 denotes normal mixture
c     a = 2 denotes exponential mixture
c     a = 3 denotes Poisson mixture
c     a = 4 denotes binomial mixture
c
          if (a .eq. 1) then
            f(i,j) = exp(-half*((x(i) - mean(j))/sd(j))**2) / sd(j)
          else if (a .eq. 2) then
            f(i,j) = exp(-x(i)/mean(j)) / mean(j)
          else if (a .eq. 3) then
            if (x(i) .eq. x(1)) then
              f(i,j) = exp(-mean(j)) * mean(j)**x(i)
            else
              f(i,j) = f(i-1,j) * mean(j)
            end if
          else
            if (x(i) .eq. x(1)) then
              f(i,j) = (one - mean(j) / x(m))**x(m) * (mean(j) /
     +                  (x(m) - mean(j)))**x(i)
            else
              f(i,j) = f(i-1,j) * (mean(j) / (x(m) - mean(j)))
            end if
          end if
          g(i) = g(i) + alpha(j) * f(i,j)
   70   continue
        logl = logl + n(i) * log(g(i))
   80 continue
c
c     Calculation of the probability densities of the sub-populations
c     which form the mixture, and the log-likelihood function.
c
      test = .false.
      sumalpha = zero
      do 100 j = 1, k
        nt(j) = zero
        dt(j) = zero
        vt(j) = zero
        do 90 i = 1, m
          part = f(i,j) * n(i) / g(i)
          dt(j) = dt(j) + part
          nt(j) = nt(j) + part * x(i)
          if (a .eq. 1) vt(j) = vt(j) + part * (x(i) - mean(j))**2
   90   continue
c
c     Calculation of denominators and numerators of new estimates.
c
        newmean(j) = nt(j) / dt(j)
        if (j .ne. k) then
          newalpha(j) = alpha(j) * dt(j) / nobvs
          sumalpha = sumalpha + newalpha(j)
        else
          newalpha(k) = one - sumalpha
        end if
        if (a .eq. 1) newsd(j) = sqrt(vt(j) / dt(j))
c
c     Convergence test.
c
        if (abs(oldlogl - logl) .gt. tol) test = .true.
        oldlogl = logl
        alpha(j) = newalpha(j)
```

```
          mean(j) = newmean(j)
          if (a .eq. 1) sd(j) = newsd(j)
    100 continue
c
      if (c .gt. 0) then
        if ((counter/c)*c .eq. counter) then
          call putout(k, alpha, mean, sd, logl)
        end if
      end if

      if (test) go to 30
      return
      end
```

This version of AS204 is in Pascal. It does not use AS 66 for the calculation of normal probabilities, but one of Hastings' algorithms.

```
(* Program description: Journal of the Royal Statistical Society (Series C)
   Vol 33 No 3  1984, R.W. Farebrother,  Algorithm AS204, pp 332 - 339

   Use of Rubin's (1962) method to evaluate the
   expression  Pr[d k(i) K}(m(i),k(i)}) < c]   where k(i) and c are
   positive constants, and where K}(m(i),k(i)}) represents an independent
   chi-squared random variable with m(j) degrees of freedom and
   non-centrality parameter k(i)}.

     n =  number of chi-squared terms
     c =  Critical chi-squared value
     maxit = maximum number of iterations = 500 in Vol 33 No. 3 1984
     eps = degree of accuracy

     This includes an auxiliary algorithm, Function 26.2.17 from 'Handbook
     of Mathematical Functions', (1968), p 932, to obtain probabilities for
     the standard normal distribution.

     The program returns the cumulative probability value at the point c in
     the distribution.

*)
CONST
     n =  3;          (* number of terms in equation *)
     maxit = 500;     (* maximum number of iterations in eq. 3 of algorithm *)
     eps = 0.0001;    (* desired level of accuracy *)
     mode = 0.90625;  (* as in AS 204A  *)

VAR
    RUBEN, tol, beta, ao, aoinv   : Real;
    c, dnsty, z, rz               : Real;
    eps2, hold, hold2, sum, sum1  : Real;
    dans, lans, pans ,prbty, temp : Real;
    ifault, itemp, i, j, k, m     : Integer;
    lambda, delta                 : Array[1..n] of Real;
    mult                          : Array[1..n] of Integer;
    gamma, theta                  : Array[1..n] of Real;
    a, b                          : Array[1..maxit] of Real;
    exit, L                       : Boolean;

Procedure centnorm(VAR x,prob: Real);

const
  p  =  0.2316419;
  b1 =  0.319381530;
  b2 = -0.356563782;
  b3 =  1.781477937;
  b4 = -1.821255978;
  b5 =  1.330274429;

var
  constant, Z, t, Q        : real;
(*                           x       standardised value
                             Z       N(0,1) function value at x
                             Q       P(X > abs(x)) where X is N(0,1)
                          prob       P(-x < z < x)   *)

Begin
```

```
      constant := 1/sqrt(2*pi);
      If abs(x) < 4.5 then
      Begin
            if x < 0 then
                 x := -x;
            Z := exp(-x*x/2)*constant;
            t := 1/(1+p*x);
            Q := ((((b5*t+b4)*t+b3)*t+b2)*t+b1)*t*Z;
      end
      else
            Q := 0;
      prob := 1 - 2 * Q;
End;   (* centnorm  *)

BEGIN
      (* RUBEN evaluates the probability that a positive definite quadratic
         form in Normal variates is less than a given value *)

      c := 1.0;
      For i := 1 to n do
          delta[i] := 0.0;
      lambda[1] := 6;
      lambda[2] := 3;
      lambda[3] := 1;
      For i := 1 to n do
           mult[i] := 1;

      exit := false;
      L := false;
      If (n < 1) OR (c <= 0.0) OR (maxit < 1) OR (eps <= 0.0) then
      Begin
           RUBEN := -2.0;
           ifault := 2;
      End
      Else
      Begin
           tol := -200.0;
           (* preliminaries *)
           beta := lambda[1];
           sum := lambda[1];
           For i := 1 to n do
           Begin
                hold := lambda[i];
                If (hold <= 0.0) OR (mult[i] < 1) OR (delta[i] < 0.0) then
                Begin
                     RUBEN := -7.0;
                     ifault := -i;
                     exit := true;
                     i := n;
                End;
                If beta > hold then beta := hold;
                If sum < hold then sum := hold;
           End;
           If exit = false then
           Begin
                If mode > 0.0 then
                Begin
                     temp := mode * beta;
                     beta := temp;
                End
                Else
```

```
Begin
     temp := 2.0/(1.0/beta + 1.0/sum);
     beta := temp;
End;
k := 0;
sum := 1.0;
sum1 := 0.0;
for i := 1 to n do
Begin
     hold := beta/lambda[i];
     gamma[i] := 1.0 - hold;
     temp := 1;
     for j := 1 to mult[i] do
         temp := temp * hold;
     sum := sum * temp;
     sum1 := sum1 + delta[i];
     k := k + mult[i];
     theta[i] := 1.0;
End;
Writeln;
ao := EXP(0.5 * (LN(sum) - sum1));
If ao <= 0.0 then
Begin
     RUBEN := 0.0;
     dnsty := 0.0;
     ifault := 1;
End
Else
Begin
     z := c/beta;
     (* evaluate probability and density of chi-squared on
        k degrees of freedom.  The constant 0.22579135264473
        is ln(sqrt(pi/2))  *)
     itemp := (k DIV 2) * 2;
     If k = itemp then
     Begin
          i := 2;
          lans := -0.5 * z;
          dans := EXP(lans);
          pans := 1.0 - dans;
     end
     Else
     Begin
          i := 1;
          lans := -0.5 * (z + LN(z)) - 0.22579135264473;
          dans := EXP(lans);
          rz := SQRT(z);
          centnorm(rz,pans);
          (* centnorm(x) will be a procedure or function to
             evaluate the probability that a standard normal
             variable lies in the interval [-x,x]
             Algorithm AS66 recommended *)
     End;
     k := k - 2;
     While i <= k do
     Begin
          If lans < tol then
          Begin
               lans := lans + LN(z/i);
               dans := EXP(lans);
          End
```

```
                    Else
                    Begin
                         temp := dans;
                         dans := temp * z/i;
                    End;
                    temp := pans;
                    pans := temp - dans;
                    i := i + 2;
               End;
               { evaluate successive terms of expansion  }
               prbty := pans;
               dnsty := dans;
               eps2 := eps/ao;
               aoinv := 1.0/ao;
               sum := aoinv - 1.0;
               For m := 1 to maxit do
               Begin
                    sum1 := 0.0;
                    For i := 1 to n do
                    Begin
                         hold := theta[i];
                         theta[i] := hold * gamma[i];
                         hold2 := theta[i];
                         temp := hold2 * mult[i] + m * delta[i] * (hold -
hold2);
                         sum1 := sum1 + temp;
                    End;
                    b[m] := 0.5 * sum1;
                    sum1 := b[m];
                    itemp := m - 1;
                    (* itemp = 0 when m = 1  !!!  *)
                    If itemp > 0 then
                    For i := itemp downto 1 do
                         sum1 := sum1 + b[i] * a[m-i];
                    a[m] := sum1/m;
                    sum1 := a[m];
                    k := k + 2;
                    If lans < tol then
                    Begin
                         lans := lans + LN(z/k);
                         dans := EXP(lans);
                    End
                    Else
                    Begin
                         temp := dans;
                         dans := temp * z/k;
                    End;
                    pans := pans - dans;
                    sum := sum - sum1;
                    temp := dnsty;
                    dnsty := temp + dans * sum1;
                    temp := sum1;
                    sum1 := pans * temp;
                    temp := prbty;
                    prbty := temp + sum1;
                    If prbty < -aoinv then
                    Begin
                         RUBEN := -3.0;
                         ifault := 3;
                         exit := true;
                         m := maxit;
```

```
                         End;
                         If exit = false then
                         Begin
                              temp := ABS(pans* sum);
                              If temp < eps2 then
                              Begin
                                   temp := ABS(sum1);
                                   If temp < eps2 then
                                   Begin
                                        ifault := 0;
                                        L := true;
                                        m := maxit;
                                   End;
                              end;
                         end;
                    End;  { m }
                    If L = false then ifault := 4;
                    If exit = false then
                    Begin
                         temp := dnsty;
                         dnsty := ao * temp/(beta + beta);
                         temp := prbty;
                         prbty := ao * temp;
                         If (prbty < 0.0) OR (prbty > 1.0) then
                            ifault := ifault + 5
                         Else
                             If dnsty < 0.0 then ifault := ifault + 6;
                         RUBEN := prbty;
                    end;
               end;
          End;
     End;
     Writeln('probability =  ',RUBEN:10:6);
     Writeln('fault code =  ',ifault:3);
END.
```

```
      SUBROUTINE ENUM(R,C,N,M,IFAULT)
C
C
C       ALGORITHM AS 205 APPL. STATIST. (1984) VOL 33, NO. 3.
C       ENUMERATES ALL R*C CONTINGENCY TABLES WITH GIVEN ROW TOTALS N(I)
C       AND COLUMN TOTALS M(J) AND CALCULATES THE HYPERGEOMETRIC
C       PROBABILITY OF EACH TABLE.
C
C       FOR TABLES HAVING TWO OR MORE ROW SUMS REPEATED, EQUIVALENT
C       TABLES DIFFERING ONLY BY A ROW PERMUTATION ARE NOT SEPARATELY
C       ENUMERATED. A REPRESENTATIVE OF EACH EQUIVALENCE CLASS IS ENUMERATED
C       AND THE MULTIPLICITY OF EACH CLASS CALCULATED.
C
C       FOR EACH TABLE ENUMERATED, SUBROUTINE EVAL IS CALLED TO CARRY OUT
C       CALCULATIONS ON THE TABLE.
C
      INTEGER R,C,N(10),M(10),IFAULT
      INTEGER TABLE(10,10),BOUND(10,10),Z(10)
      INTEGER REPS(10),MULT(10),REPSR,REPSC,MULTR,MULTC,ONE
      INTEGER NTOTAL,NTOT,MAX,NMAX,RM,CM,LEFT,ROWBND,ROWSUM
      INTEGER I,J,II,JJ,IIP,IIM,JP,KEEP
      REAL PROB(10,10),FLOGM(201),FACTLM(201),ZERO
      REAL PROB0
      LOGICAL REPT(10),REPTC(10),IEQIM
C
C       LOCAL VARIABLES -
C         TABLE(I,J)   - (I,J)-TH ENTRY OF CURRENT TABLE
C         NTOTAL       - TOTAL OF TABLE ENTRIES
C         BOUND(I,J)   - CURRENT UPPER BOUND ON TABLE(I,J) TO SATISFY
C                        ROW AND COLUMN TOTALS
C         REPT(I)      - LOGICAL = TRUE IF ROW TOTALS N(I), N(I-1) ARE EQUAL
C                                = FALSE OTHERWISE
C         REPS(I)      - NUMBER OF PREVIOUS ROWS EQUAL TO ROW I
C         MULT(I)      - MAXIMUM NUMBER OF EQUIVALENT TABLES
C                        GIVEN FIRST I ROWS
C         Z(J)         - LOWER BOUND ON SUM OF ENTRIES USED BY ALGORITHM C
C         PROB(I,J)    - PARTIAL SUM OF TERMS IN LOG(P)
C
C
      DATA MAX/10/ , NMAX/201/ , ZERO/0.0E0/ , ONE/1/
C
C         MAX  - MAXIMUM DIMENSION OF TABLE
C         NMAX - MAXIMUM NUMBER OF OBSERVATIONS + 1
C
C
C       CHECK INPUT VALUES
C
      IFAULT=1
      IF(R.GT.MAX .OR. C.GT.MAX .OR. R.LE.0 .OR. C.LE.0) RETURN
C
      IFAULT=3
      NTOTAL=0
      DO 10 I=1,R
        IF(N(I) .LE. 0) RETURN
        NTOTAL=NTOTAL+N(I)
10    CONTINUE
      NTOT=0
      DO 20 J=1,C
        IF (M(J) .LE. 0) RETURN
        NTOT=NTOT+M(J)
20    CONTINUE
```

```
        IFAULT=2
      IF (NTOT.NE.NTOTAL) RETURN
C
      IFAULT=4
      IF (NTOTAL.GE.NMAX) RETURN
C
      IFAULT=0
C
C       INITIALISE FLOGM(K)=LOG(K-1), FACTLM(K)=LOG(K-1 FACTORIAL)
C
      FLOGM(1)=ZERO
      FACTLM(1)=ZERO
      DO 50 K=1,NTOTAL
        FLOGM(K+1)=ALOG(FLOAT(K))
        FACTLM(K+1)=FACTLM(K)+FLOGM(K+1)
50    CONTINUE
C
C       CONSTANTS
C
      RM=R-1
      CM=C-1
C
C       SORT ROWS AND COLUMNS INTO ASCENDING ORDER
C
      DO 60 I=1,RM
        IP=I+1
        DO 60 II=IP,R
          IF (N(I).LE.N(II)) GOTO 60
          KEEP=N(I)
          N(I)=N(II)
          N(II)=KEEP
60    CONTINUE
C
      DO 70 J=1,CM
        JP=J+1
        DO 70 JJ=JP,C
          IF(M(J).LE.M(JJ))GOTO 70
          KEEP=M(J)
          M(J)=M(JJ)
          M(JJ)=KEEP
70    CONTINUE
C
C       CALCULATE MULTIPLICITIES OF ROWS AND COLUMNS
C
C       REPTC(J) = .TRUE. IF COLUMNS J AND J-1 HAVE THE SAME TOTAL
C       REPT(I)  = .TRUE. IF ROWS I AND I-1 HAVE THE SAME TOTAL
C
      MULTC=ONE
      REPSC=ONE
      REPTC(1)=.FALSE.
      DO 80 J=2,C
        REPTC(J) = (M(J).EQ.M(J-1))
        IF(REPTC(J)) GOTO 79
        REPSC=ONE
        GOTO 80
79      REPSC=REPSC+ONE
        MULTC=MULTC*REPSC
80    CONTINUE
C
      MULTR=ONE
      REPSR=ONE
```

```
      REPT(1)=.FALSE.
      DO 85 I=2,R
        REPT(I)=(N(I).EQ.N(I-1))
        IF(REPT(I))GOTO 84
        REPSR=ONE
        GOTO 85
84      REPSR=REPSR+ONE
        MULTR=MULTR*REPSR
85    CONTINUE
C
C       IF COLUMN MULTIPLICITY EXCEEDS ROW MULTIPLICITY TRANSPOSE TABLE
C
      IF (MULTR.GE.MULTC) GOTO 100
      MAXRC=MAX0(R,C)
      DO 90 IJ=1,MAXRC
        KEEP=N(IJ)
        N(IJ)=M(IJ)
        M(IJ)=KEEP
90    CONTINUE
      KEEP=R
      R=C
      C=KEEP
        RM=R-1
        CM=C-1
      DO 95 I=1,R
95        REPT(I)=REPTC(I)
      MULTR=MULTC
C
C       SET UP INITIAL TABLE
C
C       MAXIMUM MULTIPLICITY
C
100   MULT(1)=MULTR
      REPS(1)=ONE
C
C       CONSTANT TERM IN PROBABILITY
C
      PROB0=-FACTLM(NTOTAL+1)
      DO 105 I=1,R
        II=N(I)
        PROB0=PROB0+FACTLM(II+1)
105   CONTINUE
      DO 106 J=1,C
        JJ=M(J)
        PROB0=PROB0+FACTLM(JJ+1)
106   CONTINUE
C
C       CALCULATE BOUNDS  ON ROW 1
C
      DO 110 J=1,C
110       BOUND(1,J) = M(J)
C
C       FOR EACH I FIND GREATEST I-TH ROW SATISFYING BOUNDS
C
      DO 150 I=1,R
        IF (I.NE.1) PROB0=PROB(I-1,C)
        LEFT=N(I)
C
C         ELEMENTS OF ROW I
C
        IEQIM = REPT(I)
```

```
          DO 120 J=1,CM
          IJ=MIN0(LEFT,BOUND(I,J))
            TABLE(I,J) = IJ
            IF (J.EQ.1) PROB(I,J) = PROB0 - FACTLM(IJ+1)
            IF (J.NE.1) PROB(I,J) = PROB(I,J-1) - FACTLM(IJ+1)
            LEFT=LEFT - TABLE(I,J)
            IF (I.LT.R) BOUND(I+1,J) = BOUND(I,J) - TABLE(I,J)
            IF (LEFT.EQ.0) GOTO 121
            IF (IEQIM) IEQIM=TABLE(I,J).EQ.TABLE(I-1,J)
120       CONTINUE
          TABLE(I,C)=LEFT
          PROB(I,C)=PROB(I,CM)-FACTLM(LEFT+1)
          IF (I.LT.R) BOUND(I+1,C)=BOUND(I,C)-LEFT
          GOTO 123
121       JP=J+1
          DO 122 JJ=JP,C
            TABLE(I,JJ)=0
            PROB(I,JJ)=PROB(I,JJ-1)
            BOUND(I+1,JJ)=BOUND(I,JJ)
122       CONTINUE
123       IF(I.EQ.1) GOTO 150
          MULT(I)=MULT(I-1)
          REPS(I)=ONE
          IF (.NOT.IEQIM) GOTO 150
          REPS(I)=REPS(I-1)+ONE
          MULT(I)=MULT(I)/REPS(I)
C
150     CONTINUE
C
C
C         CALL EVAL FOR TABLE 1
C
        CALL EVAL(TABLE,R,C,N,M,PROB(R,C),MULT(R))
C
C         COMMENCE ENUMERATION OF REMAINING TABLES
C         START OF MAIN LOOP
C
200     I=R
210     I=I-1
C
C         IF I = 0 NO MORE TABLES ARE POSSIBLE
C
        IF (I.EQ.0) RETURN
C
        J=CM
        LEFT=TABLE(I,C)
        ROWBND=BOUND(I,C)
C
C         TRY TO DECREASE ELEMENT (I,J)
C
220     IF (TABLE(I,J).GT.0 .AND. LEFT.LT.ROWBND) GOTO 230
C
C         ELEMENT (I,J) CANNOT BE DECREASED - TRY (I,J-1)
C
        IF(J.EQ.1) GOTO 210
        LEFT=LEFT+TABLE(I,J)
        ROWBND=ROWBND+BOUND(I,J)
        J=J-1
        GOTO 220
C
C         DECREASE ELEMENT (I,J)
```

```
C
230     IJ=TABLE(I,J)
      PROB(I,J)=PROB(I,J)+FLOGM(IJ+1)
      TABLE(I,J)=TABLE(I,J)-1
      BOUND(I+1,J)=BOUND(I+1,J)+1
C
C       IF ROW I WAS THE SAME AS ROW I-1 IT IS NO LONGER
C
      IF (REPS(I).EQ.ONE) GOTO 270
      REPS(I)=ONE
      MULT(I)=MULT(I-1)
C
C       COMPLETE ROW I WITH THE LARGEST POSSIBLE VALUES
C
270   II=I
      IIP=II+1
      IIM=II-1
      JNEXT=J+1
      LEFT=LEFT+1
      GOTO 380
C
C       FILL UP REMAINING ROWS
C
300   II=II+1
C
C       THE LAST ROW IS TREATED SEPARATELY
C
      IF (II.EQ.R) GOTO 400
      IIP=II+1
      IIM=II-1
      IF (REPT(II)) GOTO 310
C
C       ROW TOTAL N(II) IS NOT A REPEAT - MAKE ROW II AS LARGE AS POSSIBLE
C
      LEFT=N(II)
      JNEXT=1
      GOTO 380
C
C       REPEATED ROW TOTALS
C
C       (I) IF ROW II-1 SATISFIES THE BOUNDS ON ROW II REPEAT IT
C
310   DO 320 J=1,C
      IF (TABLE(IIM,J).GT.BOUND(II,J)) GOTO 330
        IJ=TABLE(IIM,J)
      TABLE(II,J)=IJ
      BOUND(IIP,J)=BOUND(II,J)-TABLE(II,J)
      IF (J.EQ.1) PROB(II,J) = PROB(IIM,C) - FACTLM(IJ+1)
      IF (J.NE.1) PROB(II,J) = PROB(II,J-1) - FACTLM(IJ+1)
320   CONTINUE
C
C       ROW II IS A REPEAT OF ROW II-1
C
      REPS(II)=REPS(IIM)+ONE
      MULT(II)=MULT(IIM)/REPS(II)
      GOTO 300
C
C       ELEMENT J OF ROW II-1 WAS TOO BIG
C
C       CONSTRUCT THE SEQUENCE Z(J) OF LOWER BOUNDS
C
```

```
330   IF (J.GT.1) GOTO 331
C
C       IF J=1 THE BOUNDS ARE SATISFIED AUTOMATICALLY
C
        IJ=BOUND(II,1)
      TABLE(II,1)=IJ
      PROB(II,1)=PROB(IIM,C)-FACTLM(IJ+1)
      JNEXT=2
      LEFT=N(II)-TABLE(II,1)
      BOUND(IIP,1)=0
      GOTO 380
331   Z(J)=N(II)
      JM=J-1
      IF (J.EQ.C) GOTO 336
      JP=J+1
      DO 335 JJ=JP,C
335   Z(J)=Z(J)-BOUND(II,JJ)
336   DO 340 JJM=1,JM
        JJ=J-JJM
        Z(JJ)=Z(JJ+1)-BOUND(II,JJ+1)
340   CONTINUE
C
C       (II) IF THE CUMULATIVE TOTALS OF ROW II-1 ALL EXCEED THE BOUNDS Z(J)
C                       MAKE ELEMENT (II,J) EQUAL TO ITS BOUND
C
      ROWSUM=0
      JKEEP=0
      DO 350 JJ=1,JM
        ROWSUM=ROWSUM+TABLE(IIM,JJ)
        IF ( ROWSUM.LT.Z(JJ) ) GOTO 360
        IF ( ROWSUM.GT.Z(JJ) .AND. TABLE(IIM,JJ).GT.0 ) JKEEP=JJ
350   CONTINUE
      TABLE(II,J)=BOUND(II,J)
      BOUND(IIP,J)=0
        IJ=TABLE(II,J)
      PROB(II,J) = PROB(II,JM) - FACTLM(IJ+1)
      REPS(II)=ONE
      MULT(II)=MULT(IIM)
C
C       COMPLETE ROW II WITH THE LARGEST POSSIBLE ELEMENTS
C
      JNEXT=JP
      LEFT=N(II)
      DO 355 JJ=1,J
355   LEFT=LEFT-TABLE(II,JJ)
      GOTO 380
C
C       (III) THE CUMULATIVE SUMS VIOLATE THE BOUNDS
C       IF NO ELEMENT OF ROW II-1 CAN BE CHANGED TO SATISFY THE BOUNDS
C                 NO SUITABLE ROW II IS POSSIBLE
C       IN THAT CASE GO BACK AND TRY DECREASING ROW II-1
C
360   IF (JKEEP.NE.0) GOTO 370
      I=II
      GOTO 210
C
C       ELEMENT (II,JKEEP) CAN BE DECREASED
C
370   BOUND(IIP,JKEEP)=BOUND(IIP,JKEEP) + 1
        IJ=TABLE(II,JKEEP)
      PROB(II,JKEEP)=PROB(II,JKEEP)+FLOGM(IJ+1)
```

```
      TABLE(II,JKEEP)=TABLE(II,JKEEP) - 1
C
C       COMPLETE THE ROW
C
      JNEXT=JKEEP+1
      LEFT=N(II)
      DO 375 JJ=1,JKEEP
375   LEFT=LEFT-TABLE(II,JJ)
C
C       ROW II IS COMPLETE UP TO ELEMENT JNEXT-1
C       MAKE THE REMAINING ELEMENTS AS LARGE AS POSSIBLE
C       (THIS SECTION OF CODE IS USED FOR EVERY ROW, REPEATED OR NOT)
C
380   IF (JNEXT.EQ.C) GOTO 390
C
      DO 385 J=JNEXT,CM
        TABLE(II,J)=MIN0(LEFT,BOUND(II,J))
        LEFT=LEFT-TABLE(II,J)
        BOUND(IIP,J)=BOUND(II,J)-TABLE(II,J)
        IJ=TABLE(II,J)
        IF (J.EQ.1) PROB(II,J) = PROB(IIM,C) - FACTLM(IJ+1)
        IF (J.NE.1) PROB(II,J)=PROB(II,J-1)-FACTLM(IJ+1)
        IF (LEFT.EQ.0) GOTO 391
385   CONTINUE
390   TABLE(II,C)=LEFT
      PROB(II,C)=PROB(II,CM)-FACTLM(LEFT+1)
      BOUND(IIP,C)=BOUND(II,C)-LEFT
      GOTO 393
391   JP=J+1
      DO 392 JJ=JP,C
        TABLE(II,JJ)=0
        PROB(II,JJ) = PROB(II,JJ-1)
        BOUND(IIP,JJ)=BOUND(II,JJ)
392   CONTINUE
393   REPS(II)=ONE
      IF (II.GT.1) MULT(II)=MULT(IIM)
      GOTO 300
C
C       THE FINAL ROW
C
400   IF (REPT(R)) GOTO 420
C
C       NOT A REPEAT - SET ROW R EQUAL TO ITS BOUNDS
C
        IJ=BOUND(R,1)
      TABLE(R,1) = IJ
      PROB(R,1) = PROB(RM,C) - FACTLM(IJ+1)
      DO 410 J=2,C
        IJ=BOUND(R,J)
        TABLE(R,J)=IJ
        PROB(R,J) = PROB(R,J-1) - FACTLM(IJ+1)
410   CONTINUE
      MULT(R)=MULT(RM)
      GOTO 500
C
C       ROW TOTAL R IS A REPEAT - ENSURE THAT IT IS LESS THAN ROW R-1
C
420   DO 430 J=1,C
      IF (BOUND(R,J).GT.TABLE(RM,J)) GOTO 440
        IJ=BOUND(R,J)
      TABLE(R,J)=IJ
```

```
      IF (J.EQ.1) PROB(R,J) = PROB(RM,C) - FACTLM(IJ+1)
      IF (J.NE.1) PROB(R,J) = PROB(R,J-1) - FACTLM(IJ+1)
      IF (TABLE(R,J).NE.TABLE(RM,J)) GOTO 450
430   CONTINUE
C
C       ROW R IS A REPEAT OF ROW R-1
C
      REPS(R)=REPS(RM)+ONE
      MULT(R)=MULT(RM)/REPS(R)
      GOTO 500
C
C       IF ROW R WOULD BE BIGGER THAN ROW R-1 GO BACK AND TRY
C          DECREASING ROW R-2
C
440   I=RM
      GOTO 210
C
C       ROW R IS ALREADY LESS THEN ROW R-1 SO NO MORE CHECKS ARE NEEDED
C
450   JP=J+1
      DO 460 JJ=JP,C
        IJ=BOUND(R,JJ)
        TABLE(R,JJ)=IJ
        PROB(R,JJ) = PROB(R,JJ-1) - FACTLM(IJ+1)
460   CONTINUE
      MULT(R)=MULT(RM)
C
C       THE TABLE IS COMPLETE - CALL SUBROUTINE EVAL
C
500     CALL EVAL(TABLE,R,C,N,M,PROB(R,C),MULT(R))
C
C
C       END OF MAIN LOOP
C
      GOTO 200
C
C
      END
C
      SUBROUTINE EVAL(TABLE,R,C,N,M,PROB,MULT)
      INTEGER TABLE(10,10),R,C,N(10),M(10),MULT
C
      RETURN
      END
```

```
      SUBROUTINE SMOOTH(NROW, NCOL, NDIM, X, W, A, B, NCYCLE, ICYCLE, G,
     *   EPS, IFAULT)
C
C     ALGORITHM AS 206  APPL. STATIST. (1984) VOL.33, NO.3
C
C     Subroutine to order a two-dimensional array using an algorithm of
C     Dykstra & Robertson (1982).   The ordering is done so that the
C     regression function is increasing in each independent variable.
C
C     Incorporates corrections from Applied Statistics vol.35(3),
C     vol.36(1) and vol.40(1).
C
      REAL X(NROW,NCOL), W(NROW,NCOL), A(NROW,NCOL,4), B(NDIM,5),
     *     G(NROW,NCOL), EPS, ZERO, DELTA, DELC, DELR, ORD, WW, FRACT
      DATA ZERO/0.0/, DELTA/0.00001/, FRACT/0.5/
C
      IFAULT = 0
C
C     Check that there are at least 2 rows and columns
C
      IF (NROW .LT. 2 .OR. NCOL .LT. 2) GO TO 120
C
C     Check that weights are positive or zero
C
      WSUM = ZERO
      WXSUM = ZERO
      WMIN = 1.0E+08
      DO 3 I = 1, NROW
        DO 3 J = 1, NCOL
          WW = W(I,J)
          IF (WW .LT. ZERO) GO TO 110
          IF (WW .LT. DELTA) GO TO 3
          WSUM = WSUM + WW
          WXSUM = WXSUM + WW * X(I,J)
          IF (WW .LT. WMIN) WMIN = WW
    3 CONTINUE
      IF (WSUM .LT. DELTA) GO TO 130
      WMEAN = WXSUM / WSUM
C
      DO 5 I = 1, NROW
        DO 5 J = 1, NCOL
          WW = W(I,J)
          A(I,J,3) = WW
          A(I,J,4) = X(I,J)
          IF (WW .GE. DELTA) GO TO 5
          A(I,J,3) = FRACT * WMIN
          A(I,J,4) = WMEAN
          ICT = ICT + 1
          IFAULT = 4
    5 CONTINUE
C
C     Initialize R and C to zero, and set up workspace
C
      IFLAG = 0
      DELR = ZERO
      DELC = ZERO
      ITIC = 0
    8 ITIC = ITIC + 1
      DO 10 I = 1, NROW
        DO 10 J = 1, NCOL
          G(I,J) = A(I,J,4)
```

```
            A(I,J,2) = ZERO
            A(I,J,1) = ZERO
   10 CONTINUE
C
C     Initialize counter for number of cycles
C
      ICOUNT = 0
      IF (IFLAG .EQ. 1) GO TO 55
      IF (ITIC .EQ. 3 .AND. DELC .GT. DELR) GO TO 55
C
C     Smooth over rows
C
   25 JCOUNT = 0
      DO 50 I = 1, NROW
        DO 30 J = 1, NCOL
          B(J,1) = G(I,J) - A(I,J,1)
          B(J,2) = A(I,J,3)
   30   CONTINUE
C
        CALL PAV(NCOL, B, 1, B(1,2), B(1,3))
C
        KCOUNT = 0
        DO 40 J = 1, NCOL
          ORD = B(J,3)
          A(I,J,1) = ORD - B(J,1)
          IF (ABS(ORD - G(I,J)) .LT. EPS) KCOUNT = KCOUNT + 1
          G(I,J) = ORD
   40   CONTINUE
C
C     Determine if there is no change in the Ith row from the previous
C     iteration
C
        IF (KCOUNT .EQ. NCOL) JCOUNT = JCOUNT + 1
   50 CONTINUE
C
      ICOUNT = ICOUNT + 1
      IF (ICOUNT .EQ. 2 .AND. IFLAG .EQ. 1)
     *    CALL DIST(A(1,1,1), NROW, NCOL, DELR, IFLAG)
      IF (ICOUNT .EQ. 2 .AND. IFLAG .EQ. 2 .AND. ITIC .EQ. 2) GO TO 8
      IF (ICOUNT .EQ. 1) GO TO 55
      IF (NCYCLE .EQ. ICOUNT) GO TO 100
C
C     Determine if there has been no change in all rows from the
C     previous iteration
C
      IF (JCOUNT .EQ. NROW) GO TO 90
C
C     Smooth over columns
C
   55 LCOUNT = 0
      DO 80 J = 1, NCOL
        DO 60 I = 1, NROW
   60   A(I,J,2) = G(I,J) - A(I,J,2)
C
        CALL PAV(NROW, A(1,J,2), 1, A(1,J,3), B(1,3))
C
        MCOUNT = 0
        DO 70 I = 1, NROW
          ORD = B(I,3)
          A(I,J,2) = ORD - A(I,J,2)
          IF (ABS(ORD - G(I,J)) .LT. EPS) MCOUNT = MCOUNT + 1
```

```
            G(I,J) = ORD
   70   CONTINUE
C
C     Determine if there is no change in the Jth column from the
C     previous iteration
C
        IF (MCOUNT .EQ. NROW) LCOUNT = LCOUNT + 1
   80 CONTINUE
C
      ICOUNT = ICOUNT + 1
      IF (ICOUNT .EQ. 2 .AND. IFLAG .EQ. 0)
     *    CALL DIST(A(1,1,2), NROW, NCOL, DELC, IFLAG)
      IF (ICOUNT .EQ. 2 .AND. IFLAG .EQ. 1) GO TO 8
      IF (ICOUNT .EQ. 1) GO TO 25
C
C     Determine if there is has been no change in any column from the
C     previous iteration
C
      IF (LCOUNT .EQ. NCOL) GO TO 90
C
C     Check if number of cycles has been reached
C
      IF (NCYCLE .EQ. ICOUNT) GO TO 100
      GO TO 25
   90 ICYCLE = ICOUNT
      RETURN
C
  100 ICYCLE = ICOUNT
      IFAULT = IFAULT + 1
      RETURN
  110 IFAULT = 2
      RETURN
  120 IFAULT = 3
      RETURN
  130 IFAULT = 6
      RETURN
      END


      SUBROUTINE PAV(K, X, IORDER, W, FINALX)
C
C     ALGORITHM AS 206.1  APPL. STATIST. (1984) VOL.33, NO.3
C
C     Apply pool adjacent violators theorem
C
      INTEGER NW(20)
      REAL X(20), W(20), FINALX(20), FX(20), PW(20), W1(20), WT(20), EPS
C
      DATA EPS/1.0E-06/
C
C     Set up workspace
C
      NWC = K
      DO 10 I = 1, K
        NW(I) = 1
        FX(I) = X(I)
        IF (IORDER .EQ. 0) FX(I) = -FX(I)
        WT(I) = W(I)
        PW(I) = WT(I) * FX(I)
        W1(I) = W(I)
   10 CONTINUE
```

```
      IBEL = K - 1
   20 I = 0
   30 I = I + 1
   35 IF (I .GT. IBEL) GO TO 50
      I1 = I + 1
C
C     Determine if pooling is required
C
      IF (FX(I) - FX(I1) .LE. EPS) GO TO 30
C
C     Pool the adjacent values
C
      PW(I) = PW(I) + PW(I1)
      W1(I) = W1(I) + W1(I1)
      FX(I) = PW(I) / W1(I)
      NW(I) = NW(I) + NW(I1)
      NWC = NWC - 1
      IF (I1 .GT. IBEL) GO TO 45
      DO 40 J = I1, IBEL
        J1 = J + 1
        PW(J) = PW(J1)
        W1(J) = W1(J1)
        FX(J) = FX(J1)
        NW(J) = NW(J1)
   40 CONTINUE
   45 IBEL = IBEL - 1
      GO TO 35
   50 ICOUNT = 0
      IF (IBEL .LE. 0) GO TO 70
C
C     Determine if all values are ordered
C
      DO 60 L = 1, IBEL
   60 IF (FX(L) - FX(L+1) .LE. EPS) ICOUNT = ICOUNT + 1
      IF (ICOUNT .NE. IBEL) GO TO 20
C
C     Recover final ordered values
C
   70 J = 1
      JL = 1
      JU = NW(1)
   80 DO 90 L = JL,JU
   90 FINALX(L) = FX(J)
      J = J + 1
      IF (J .GT. NWC) GO TO 100
      JL = JU + 1
      JU = JU + NW(J)
      GO TO 80
  100 IF (IORDER .EQ. 1) RETURN
      DO 110 I = 1, K
  110 FINALX(I) = -FINALX(I)
      RETURN
      END


      SUBROUTINE DIST(A1, NROW, NCOL, DEL, IFLAG)
C
C     ALGORITHM AS 206.2  APPL. STATIST. (1984) VOL.33, NO.3
C
      REAL A1(NROW,NCOL), ZERO, DEL
C
```

```
      DATA ZERO/0.0/
C
      DEL = ZERO
      DO 20 I = 1, NROW
        DO 20 J = 1, NCOL
          DEL = DEL + A1(I,J) * A1(I,J)
   20 CONTINUE
      IFLAG = IFLAG + 1
      RETURN
      END
```

```
      subroutine gllm(ni, nid, nj, nk, nkp, ji, y, c, conv, w, v, e, f,
     *  cspr, cslr, ifault)
c
c       algorithm as 207 appl. statist. (1984) vol.33, no.3
c
c       fitting a generalized log-linear model
c       to fully or partially classified frequencies.
c
      integer ji(ni)
      double precision y(nj), c(nid, nkp), w(4, ni), v(2, nkp), e(ni),
     *  f(nj), eps, zero, one, cmin, conv, cslr, cspr, csum, ctmax, ff,
     *  yy
      logical lconv
      data eps, zero, one/0.00001d0, 0.0d0, 1.0d0/
c
      ifault = 1
c
c       check the ji array
c
      do 100 i = 1, ni
      if (ji(i) .lt. 1 .or. ji(i) .gt. nj) return
  100 continue
      ifault = 0
c
c       initialize
c
      do 110 i = 1, ni
  110 e(i) = one
      do 120 j = 1, nj
  120 w(3, j) = zero
c
c       standardize the c matrix
c
      cmin = zero
      do 130 i = 1, ni
      do 130 k = 1, nk
      cmin = min(cmin, c(i, k))
  130 continue
      if (cmin .eq. zero) goto 150
      do 140 i = 1, ni
      do 140 k = 1, nk
        c(i, k) = c(i, k) - cmin
  140 continue
  150 ctmax = zero
      do 170 i = 1, ni
        csum = zero
        do 160 k = 1, nk
  160   csum = csum + c(i, k)
        ctmax = max(ctmax, csum)
        w(4, i) = csum
  170 continue
      if (ctmax .gt. eps) goto 180
      ifault = 2
      return
  180 if (abs(ctmax - one) .le. eps) goto 200
      do 190 i = 1, ni
        do 190 k = 1, nk
          c(i, k) = c(i, k) / ctmax
  190 continue
      goto 150
  200 do 210 i = 1, ni
```

```
            if (abs(w(4, i) - one) .gt. eps) goto 220
      210 continue
          nkk = nk
          goto 300
      220 nkk = nkp
          do 230 i = 1, ni
      230 c(i, nkk) = one - w(4, i)
    c
    c       enter the em algorithm
    c
      300  do 310 j = 1,nj
      310  f(j) = zero
           do 320 i = 1,ni
             j = ji(i)
             f(j) = f(j) + e(i)
      320  continue
    c
    c       check for convergence
    c
          lconv = .true.
          do 330 j = 1, nj
            if (abs(f(j) - w(3, j)) .gt. conv) lconv = .false.
            w(3, j) = f(j)
      330 continue
          if (lconv) goto 500
    c
          do 340 i = 1, ni
            j = ji(i)
            w(1, i) = y(j)
            if (f(j) .gt. eps) w(1, i) = e(i) * y(j) / f(j)
      340 continue
          do 360 k = 1, nkk
            v(1, k) = zero
            do 350 i = 1, ni
      350   v(1, k) = v(1, k) + c(i, k) * w(1, i)
      360 continue
    c
    c       enter the ipf algorithm
    c
      400 do 410 i = 1, ni
      410 w(2, i) = e(i)
          do 440 k = 1, nkk
            v(2, k) = zero
            do 420 i = 1, ni
      420 v(2, k) = v(2, k) + c(i,k) * e(i)
          do 430 i = 1, ni
      430 if (c(i, k) .gt. eps .and. v(2, k) .gt. eps)
         *  e(i) = e(i) * (v(1, k) / v(2, k)) ** c(i, k)
      440 continue
          do 450 i = 1, ni
            if (abs(e(i) - w(2, i)) .gt. conv) goto 400
      450 continue
          goto 300
    c
    c       calculate the goodness-of-fit statistics
    c
      500 cspr = zero
          cslr = zero
          do 530 j = 1, nj
            yy = y(j)
            ff = f(j)
```

```
        if (ff .le. eps) goto 530
        cspr = cspr + (yy - ff) ** 2 / ff
        if (yy .le. eps) goto 530
        cslr = cslr + yy * log(yy / ff)
  530 continue
      cslr = 2.0d0 * cslr
      return
      end
```

```
      SUBROUTINE LGSTN(ZM, ZP, TOL, IFAULT)
C
C        ALGORITHM AS 208  APPL. STATIST. (1985) VOL.34, NO.1
C
C        FITS LOGISTIC NORMAL FOR 3 DIMENSIONAL CASE, USING
C        MOMENTS OF FIRST AND SECOND ORDER
C
      REAL DJ(5, 5), EPS, ONE, R(5), SS, TOL, TWO, WW, WW0, WW1, WW2,
     *  WW3, ZERO, ZM(5), ZM1, ZM2, ZP(5)
      INTEGER IP(5)
      DATA ZERO, ONE, TWO, EPS, ITMAX / 0.0, 1.0, 2.0, 0.01, 20/
C
C        FIND STARTING VALUES
C
      IFAULT = 0
      N = 5
      R(1) = ZM(3) - ZM(1) ** 2
      R(2) = ZM(5) - ZM(2) ** 2
      R(3) = ZM(4) - ZM(1) * ZM(2)
      ZM1 = ONE - ZM(1)
      ZM2 = ONE - ZM(2)
      WW = ONE - ZM(1) - ZM(2)
      WW0 = ZM1 * ZM2 + ZM(1) * ZM(2)
      WW1 = R(1) * ZM2 ** 2 + R(2) * ZM(1) ** 2 + TWO * R(3) * ZM2 *
     *  ZM(1)
      WW2 = R(1) * ZM(2) * ZM2 + R(2) * ZM1 * ZM(1) + R(3) * WW0
      WW3 = R(1) * ZM(2) ** 2 + R(2) * ZM1 ** 2 + TWO * R(3) * ZM1 *
     *  ZM(2)
C
C        CHECK THAT MOMENTS ARE IN RANGE
C
      IF (AMIN1(R(1), R(2), ZM(1), ZM(2), WW, WW1, WW3) .LE. ZERO)
     *  IFAULT = 1
      IF (IFAULT .NE. 0) RETURN
      WW1 = SQRT(ONE / WW1)
      WW3 = SQRT(ONE / WW3)
      ZP(1) = ALOG(ZM(1) / WW)
      ZP(2) = ALOG(ZM(2) / WW)
      ZP(3) = ZM(1) * WW * WW1
      ZP(4) = ZM(2) * WW * WW3
C
C        FIND INITIAL VALUE FOR ZP(5) THAT IS INSIDE ALLOWED RANGE
      ZP(5) = AMIN1(AMAX1(WW1 * WW2 * WW3, -ONE + EPS), ONE - EPS)
C
C        INITIAL VALUES IN ZP SO START ITERATION CYCLE
C
      IT = 0
100      IT = IT + 1
C
C        FIND MOMENTS AND DERIVATIVES GIVEN LATEST PARAMETER VALUES
C
      CALL LSQ2(ZM, ZP, R, DJ, SS, IER)
      IF (IER .NE. 0) IFAULT = 4
      IF (IFAULT .NE. 0) RETURN
C
C        DECOMPOSE 2ND DERIVATIVE MATRIX
C
      CALL DECOMP(N, N, DJ, IP)
      IF (IP(N) .EQ. 0) IFAULT = 3
      IF (IFAULT .NE. 0) RETURN
C
```

```
C        SOLVE LINEAR EQUATION
C
      CALL SOLVE(N, N, DJ, R, IP)
C
C        FIND UPDATED PARAMETER ESTIMATES
C
      ZP(1) = ZP(1) - R(1)
      ZP(2) = ZP(2) - R(2)
      ZP(3) = ZP(3) * EXP(-R(3))
      ZP(4) = ZP(4) * EXP(-R(4))
      WW = EXP(-R(5)) * (ONE + ZP(5)) / (ONE - ZP(5))
      ZP(5) = (WW - ONE) / (WW + ONE)
      IF (SS .LE. TOL) RETURN
      IF (IT .EQ. ITMAX) IFAULT = 2
      IF (IFAULT .NE. 0) RETURN
      GOTO 100
      END
C
      SUBROUTINE LSQ2(ZM, ZP, R, DJ, SS, IER)
C
C        ALGORITHM AS 208.1  APPL. STATIST. (1985) VOL.34, NO.1
C
C        LSQ2 FINDS SS AND DERIVATIVES FOR FITTING LOGISTIC NORMAL
C        FOR S2 BY MOMENTS
C
      REAL D(5, 5, 6), DC, DJ(5, 5), DR, DT, D34, EPS, EPS1, HALF, ONE,
     *  R(5), SS, ZERO, ZI, ZJ, ZM(5), ZMOM(5, 5), ZP(5)
      INTEGER IMAP(6), KMAP(6)
      DATA KMAP(1), KMAP(2), KMAP(3), KMAP(4), KMAP(5), KMAP(6)
     *  /0, 1, 2, 1, 4, 2/
      DATA IMAP(1), IMAP(2), IMAP(3), IMAP(4), IMAP(5), IMAP(6)
     *  /0, 4, 3, 4, 3, 3/
      DATA ZERO, HALF, ONE, EPS, EPS1 /0.0, 0.5, 1.0, 7.7, 18.0/
C
      IER = 0
C
C      CHECK THAT PARAMETERS ARE IN RANGE
C
      IF (AMIN1(ZP(3), ZP(4)) .LE. ZERO) IER = 1
      IF (IER .NE. 0) RETURN
      IF (ABS(ZP(5)) .GT. ONE) IER = 2
      IF (AMAX1(EPS / ZP(3) + ABS(ZP(1)), EPS / ZP(4) + ABS(ZP(2)))
     *  .GT. EPS1) IER = 3
      IF (IER .NE. 0) RETURN
C
C        GET MOMENTS
C
      CALL MOM2(ZP, ZMOM)
C
C        INITIALISE D WITH MOMENTS OR ZEROS
C
      DO 100 K = 2, 4, 2
      DO 100 I = 1, 5
      J = 6 - I
100      D(I, J, K) = ZERO
      DO 120 I = 1, 5
      J0 = 6 - I
      DO 120 J = 1, J0
120      D(I, J, 1) = ZMOM(I, J)
C
C        DO DIFFERENCING ON D TO GET DERIVATIVES
```

```
C        KMPA AND IMAP TELL WHICH DIFFERENCES TO USE
C
      DO 200 K = 2, 6
      K0 = KMAP(K)
      I0 = IMAP(K)
      DO 200 I = 1, I0
      ZI = I - 1
      J0 = I0 - I + 1
      DO 200 J = 1, J0
      ZJ = J - 1
      IF (K .LE. 3) D(I, J, K) = ZI * D(I, J, K0) - (ZI + ZJ) *
     *  D(I + 1, J, K0)
      IF (K .GT. 3) D(I, J, K) = ZJ * D(I, J, K0) - (ZI + ZJ) *
     *  D(I, J + 1, K0)
200      CONTINUE
      L = 0
      SS = ZERO
      D34 = ONE / (ZP(3) * ZP(4))
      DC = HALF * (ONE - ZP(5) ** 2) * D34
      DR = ZP(3) / ZP(4)
      DO 400 I = 2, 3
      DO 400 J = 1, I
      K = I - J + 1
      L = L + 1
C
C        FIND SS, RESIDUALS AND DERIVATIVES W.R.T. PARAMETERS
C        D(K,J,1) CONTAINS FITTED MOMENT CORRESPONDING TO ZM(L)
C
      R(L) = D(K, J, 1) - ZM(L)
      SS = SS + R(L) ** 2
      DT = ZP(5) * D(K, J, 6)
      DJ(L, 1) = D(K, J, 2)
      DJ(L, 2) = D(K, J, 4)
      DJ(L, 3) = -D34 * (D(K, J, 3) / DR + DT)
      DJ(L, 4) = -D34 * (D(K, J, 5) * DR + DT)
      DJ(L, 5) = DC * D(K, J, 6)
400      CONTINUE
      RETURN
      END
C
      SUBROUTINE MOM2(ZP, A)
C
C        ALGORITHM AS 208.2  APPL. STATIST. (1985) VOL.34, NO.1
C
C        MOM2 CALCULATES E((X**(I-1))(Y**(J-1)))  I,J=1 TO 5
C          =A(I,J)  FOR LOGISTIC NORMAL
C           I+J LESS THAN 6
C           USING (20,20) POINT HERMITIAN INTEGRATION
C
      REAL A(5, 5), AB(10), CON, ONE, W, WT(10), W0, X, Y, Z, ZERO,
     *  ZP(5), Z1, Z10, Z2, Z20, Z3
C
C        WT AND AB CONTAIN THE WEIGHTS AND ABCISSAS FOR THE
C        HERMITIAN INTEGRATION
C
      DATA WT(1), WT(2), WT(3), WT(4), WT(5), WT(6), WT(7), WT(8),
     *  WT(9), WT(10)
     *  /0.2607930634495549E18, 0.161739333984E18,
     *  0.6150637206397691E17,0.1399783744710 1E17,
     *  0.1830103131080493E16, 0.1288262799619294E15,
     *  0.4402121090230853E13, 0.6127490259982946E11,
```

```
     *  0.2482062362315179E9, 0.1257800672437927E6/
      DATA AB(1), AB(2), AB(3), AB(4), AB(5), AB(6), AB(7), AB(8),
     *  AB(9), AB(10)
     *  /0.34696415708135593, 1.04294534880275103, 1.74524732081412671,
     *  2.45866361117236775, 3.18901481655338941, 3.94396735065731627,
     *  4.73458133404605532, 5.57873880589320117, 6.51059015701365447,
     *  7.61904854167975829/
      DATA ZERO, ONE, CON /0.0, 1.0, 1.0E-36/
C
      Z = SQRT(ONE - ZP(5) ** 2)
C
C        INITIALIZE A() TO ZERO
C
      DO 100 J = 1, 5
      J0 = 6 - J
      DO 100 J1 = 1, J0
100      A(J, J1) = ZERO
      DO 500 I = 1, 10
      X = AB(I)
      W0 = WT(I)
      DO 500 J = 1, 10
      Y = AB(J)
      W = W0 * WT(J)
C
C        GIVEN (X,Y)= GRID POINT AND W= WEIGHT DO FOR (+-X,+-Y)
C
      DO 500 I1 = 1, 2
      DO 400 J1 = 1, 2
      Z10 = W
      Z1 = EXP(X / ZP(3) + ZP(1))
      Z2 = EXP((Y * Z + ZP(5) * X) / ZP(4) + ZP(2))
      Z3 = ONE / (ONE + Z1 + Z2)
      Z1 = Z1 * Z3
      Z2 = Z2 * Z3
      DO 150 I0 = 1, 5
      Z20 = Z10
      J10 = 6 - I0
      DO 140 J0 = 1, J10
      A(I0, J0) = A(I0, J0) + Z20
      Z20 = Z20 * Z2
140      CONTINUE
      Z10 = Z10 * Z1
150      CONTINUE
      Y = -Y
400      CONTINUE
      X = -X
500      CONTINUE
C
C        RESCALE MOMENTS TO AVOID UNDERFLOW
C
      DO 600 J = 1, 5
      J0 = 6 - J
      DO 600 J1 = 1, J0
600      A(J, J1) = A(J, J1) * CON
      RETURN
      END
```

```
      SUBROUTINE PKURT(B2, N, P, ZP, IFAULT)
C
C        ALGORITHM AS209  APPL. STATIST. (1985) VOL. 34, NO. 1
C
C        EVALUATES P VALUE FOR COEFFICIENT OF KURTOSIS B2.
C        N IS SAMPLE SIZE, N GE 5.
C        ZP IS NORMAL EQUIVALENT DEVIATE FOR P.
C        BASED ON SIMULATIONS OF DISTRIBUTION OF B2 APPROXIMATED
C        BY MODIFIED SU. USES FIRST ORDER APPROXIMATION TO MEAN AND SD
C       OF TRANSFORMED B2 FOR N GT 5000.
C        APPROXIMATION ACCURATE TO ABOUT +/- 0.05 FOR ZP IN RANGE -3.1
C        LT ZP LT 3.1.
C
C        IFAULT = 0   OK
C        1   B2 LT 0
C        2   N LT 15
C        3   P(B2) LT 0.0001 (WHEN NLT15) : P RETURNED AS 0.0001
C        4   P(B2) GT 0.9999 (WHEN NLT15) : P RETURNED AS 0.9999
C        5,6 INTERNAL ERROR IN COMPUTING B-SPLINE INTERPOLANT
C        7   B2 BELOW EXTRAPOLATED SIMULATION LIMIT: P AND ZP RETURNED
C        AS 0
C
      INTEGER N1(22), N2(19)
      REAL LAMDA, CSL1(22), CI1(22), CSL2(19), CI2(19), KNOT(24)
      REAL PERC(20, 10), PWRK(20), COEF(24), PERZ(20)
C
C        WRK IS WORKING ARRAY OF DIM LWRK = (NO. OF POINTS)*6 + 16 =
C        136 FOR 20.
C
      REAL WRK(136), LOWP, LOWZ
      LOGICAL LOWER
      DATA MSMALL /22/, MBIG /19/, M /20/, LWRK /136/, I5000 /5000/,
     *  I100 /100/
      DATA LOWP /0.0001/, LOWZ /-3.71902/
      DATA ZERO /0.0E0/, ONE /1.0E0/, TWO /2.0E0/, THREE /3.0E0/, FIVE /
     *  5.0E0/
      DATA TWENT4 /24.0E0/, HUNDRD /100.0E0/, LOWER /.FALSE./
C
      DATA N1(1), N1(2), N1(3), N1(4), N1(5) /15, 15, 15, 15, 17/
      DATA N1(6), N1(7), N1(8), N1(9), N1(10) /18, 19, 20, 22, 25/
      DATA N1(11), N1(12), N1(13), N1(14), N1(15) /30, 35, 40, 45, 50/
      DATA N1(16), N1(17), N1(18), N1(19), N1(20) /60, 70, 80, 100, 100/
      DATA N1(21), N1(22) /100, 100/
C
      DATA N2(1), N2(2), N2(3), N2(4), N2(5) /100, 100, 100, 100, 140/
      DATA N2(6), N2(7), N2(8), N2(9), N2(10) /160, 180, 200, 250, 300/
      DATA N2(11), N2(12), N2(13), N2(14), N2(15) /400, 500, 600, 800,
     *  1000/
      DATA N2(16), N2(17), N2(18), N2(19) /5000, 5000, 5000, 5000/
C
      DATA CSL1(1), CSL1(2), CSL1(3) /0.130608, 0.130309, 0.130799/
      DATA CSL1(4), CSL1(5), CSL1(6) /0.131271, 0.130269, 0.131037/
      DATA CSL1(7), CSL1(8), CSL1(9) /0.133146, 0.131201, 0.131552/
      DATA CSL1(10), CSL1(11), CSL1(12) /0.134288, 0.136172, 0.139247/
      DATA CSL1(13), CSL1(14), CSL1(15) /0.145013, 0.153807, 0.167036/
      DATA CSL1(16), CSL1(17), CSL1(18) /0.191936, 0.217312, 0.246664/
      DATA CSL1(19), CSL1(20), CSL1(21), CSL1(22) /0.0, 0.0, 0.0, 0.0/
C
      DATA CI1(1), CI1(2), CI1(3) /0.991947, 0.984242, 0.981813/
      DATA CI1(4), CI1(5), CI1(6) /0.967821, 0.962803, 0.951360/
      DATA CI1(7), CI1(8), CI1(9) /0.934610, 0.909853, 0.873722/
```

```
      DATA CI1(10), CI1(11), CI1(12) /0.823252, 0.772149, 0.716999/
      DATA CI1(13), CI1(14), CI1(15) /0.639262, 0.531453, 0.375579/
      DATA CI1(16), CI1(17), CI1(18) /0.141451, -0.096974, -0.316306/
      DATA CI1(19), CI1(20), CI1(21), CI1(22) /0.0, 0.0, 0.0, 0.0/
C
      DATA CSL2(1), CSL2(2), CSL2(3) /0.246227, 0.242238, 0.264287/
      DATA CSL2(4), CSL2(5), CSL2(6) /0.290128, 0.307728, 0.316642/
      DATA CSL2(7), CSL2(8), CSL2(9) /0.322317, 0.304390, 0.284983/
      DATA CSL2(10), CSL2(11), CSL2(12) /0.267009, 0.240296, 0.221648/
      DATA CSL2(13), CSL2(14), CSL2(15) /0.162402, 0.111749, 0.092712/
      DATA CSL2(16), CSL2(17), CSL2(18), CSL2(19) /0.0, 0.0, 0.0, 0.0/
C
      DATA CI2(1), CI2(2), CI2(3) /-0.320361, -0.338391, -0.439550/
      DATA CI2(4), CI2(5), CI2(6) /-0.592816, -0.678397, -0.754728/
      DATA CI2(7), CI2(8), CI2(9) /-0.794209, -0.800182, -0.788895/
      DATA CI2(10), CI2(11), CI2(12) /-0.765971, -0.760544, -0.746093/
      DATA CI2(13), CI2(14), CI2(15) /-0.733140, -0.724729, -0.719960/
      DATA CI2(16), CI2(17), CI2(18), CI2(19) /0.0, 0.0, 0.0, 0.0/
C
      DATA PERC(1, 1), PERC(2, 1), PERC(3, 1), PERC(4, 1), PERC(5, 1)
     *  /1.170, 1.173, 1.177, 1.196, 1.212/
      DATA PERC(6, 1), PERC(7, 1), PERC(8, 1), PERC(9, 1), PERC(10, 1)
     *  /1.244, 1.278, 1.348, 1.497, 1.808/
      DATA PERC(11, 1), PERC(12, 1), PERC(13, 1), PERC(14, 1), PERC(15,
     *  1) /2.133, 2.465, 2.694, 2.875, 3.003/
      DATA PERC(16, 1), PERC(17, 1), PERC(18, 1), PERC(19, 1), PERC(20,
     *  1) /3.110, 3.162, 3.219, 3.230, 3.244/
C
      DATA PERC(1, 2), PERC(2, 2), PERC(3, 2), PERC(4, 2), PERC(5, 2)
     *  /1.016, 1.042, 1.059, 1.136, 1.189/
      DATA PERC(6, 2), PERC(7, 2), PERC(8, 2), PERC(9, 2), PERC(10, 2)
     *  /1.287, 1.382, 1.488, 1.598, 1.867/
      DATA PERC(11, 2), PERC(12, 2), PERC(13, 2), PERC(14, 2), PERC(15,
     *  2) /2.211, 2.667, 2.993, 3.281, 3.516/
      DATA PERC(16, 2), PERC(17, 2), PERC(18, 2), PERC(19, 2), PERC(20,
     *  2) /3.746, 3.875, 4.047, 4.084, 4.138/
C
      DATA PERC(1, 3), PERC(2, 3), PERC(3, 3), PERC(4, 3), PERC(5, 3)
     *  /1.116, 1.139, 1.155, 1.213, 1.252/
      DATA PERC(6, 3), PERC(7, 3), PERC(8, 3), PERC(9, 3), PERC(10, 3)
     *  /1.329, 1.416, 1.533, 1.699, 1.949/
      DATA PERC(11, 3), PERC(12, 3), PERC(13, 3), PERC(14, 3), PERC(15,
     *  3) /2.277, 2.779, 3.188, 3.544, 3.872/
      DATA PERC(16, 3), PERC(17, 3), PERC(18, 3), PERC(19, 3), PERC(20,
     *  3) /4.227, 4.443, 4.775, 4.868, 5.010/
C
      DATA PERC(1, 4), PERC(2, 4), PERC(3, 4), PERC(4, 4), PERC(5, 4)
     *  /1.077, 1.131, 1.162, 1.265, 1.315/
      DATA PERC(6, 4), PERC(7, 4), PERC(8, 4), PERC(9, 4), PERC(10, 4)
     *  /1.391, 1.469, 1.581, 1.753, 2.032/
      DATA PERC(11, 4), PERC(12, 4), PERC(13, 4), PERC(14, 4), PERC(15,
     *  4) /2.340, 2.853, 3.316, 3.722, 4.110/
      DATA PERC(16, 4), PERC(17, 4), PERC(18, 4), PERC(19, 4), PERC(20,
     *  4) /4.560, 4.851, 5.351, 5.491, 5.770/
C
      DATA PERC(1, 5), PERC(2, 5), PERC(3, 5), PERC(4, 5), PERC(5, 5)
     *  /1.132, 1.180, 1.208, 1.297, 1.350/
      DATA PERC(6, 5), PERC(7, 5), PERC(8, 5), PERC(9, 5), PERC(10, 5)
     *  /1.439, 1.524, 1.634, 1.801, 2.089/
      DATA PERC(11, 5), PERC(12, 5), PERC(13, 5), PERC(14, 5), PERC(15,
     *  5) /2.408, 2.910, 3.398, 3.844, 4.263/
```

```
      DATA PERC(16, 5), PERC(17, 5), PERC(18, 5), PERC(19, 5), PERC(20,
     *  5) /4.807, 5.156, 5.801, 6.013, 6.371/
C
      DATA PERC(1, 6), PERC(2, 6), PERC(3, 6), PERC(4, 6), PERC(5, 6)
     *  /1.145, 1.217, 1.247, 1.338, 1.390/
      DATA PERC(6, 6), PERC(7, 6), PERC(8, 6), PERC(9, 6), PERC(10, 6)
     *  /1.476, 1.565, 1.679, 1.844, 2.133/
      DATA PERC(11, 6), PERC(12, 6), PERC(13, 6), PERC(14, 6), PERC(15,
     *  6) /2.458, 2.963, 3.467, 3.943, 4.419/
      DATA PERC(16, 6), PERC(17, 6), PERC(18, 6), PERC(19, 6), PERC(20,
     *  6) /4.981, 5.389, 6.183, 6.448, 6.925/
C
      DATA PERC(1, 7), PERC(2, 7), PERC(3, 7), PERC(4, 7), PERC(5, 7)
     *  /1.189, 1.239, 1.273, 1.372, 1.428/
      DATA PERC(6, 7), PERC(7, 7), PERC(8, 7), PERC(9, 7), PERC(10, 7)
     *  /1.513, 1.600, 1.718, 1.888, 2.178/
      DATA PERC(11, 7), PERC(12, 7), PERC(13, 7), PERC(14, 7), PERC(15,
     *  7) /2.505, 3.009, 3.506, 4.002, 4.492/
      DATA PERC(16, 7), PERC(17, 7), PERC(18, 7), PERC(19, 7), PERC(20,
     *  7) /5.110, 5.547, 6.411, 6.760, 7.304/
C
      DATA PERC(1, 8), PERC(2, 8), PERC(3, 8), PERC(4, 8), PERC(5, 8)
     *  /1.198, 1.272, 1.303, 1.401, 1.457/
      DATA PERC(6, 8), PERC(7, 8), PERC(8, 8), PERC(9, 8), PERC(10, 8)
     *  /1.545, 1.635, 1.753, 1.925, 2.212/
      DATA PERC(11, 8), PERC(12, 8), PERC(13, 8), PERC(14, 8), PERC(15,
     *  8) /2.539, 3.045, 3.544, 4.043, 4.542/
      DATA PERC(16, 8), PERC(17, 8), PERC(18, 8), PERC(19, 8), PERC(20,
     *  8) /5.178, 5.663, 6.645, 7.050, 7.750/
C
      DATA PERC(1, 9), PERC(2, 9), PERC(3, 9), PERC(4, 9), PERC(5, 9)
     *  /1.230, 1.301, 1.337, 1.431, 1.486/
      DATA PERC(6, 9), PERC(7, 9), PERC(8, 9), PERC(9, 9), PERC(10, 9)
     *  /1.577, 1.665, 1.784, 1.955, 2.246/
      DATA PERC(11, 9), PERC(12, 9), PERC(13, 9), PERC(14, 9), PERC(15,
     *  9) /2.570, 3.075, 3.575, 4.085, 4.602/
      DATA PERC(16, 9), PERC(17, 9), PERC(18, 9), PERC(19, 9), PERC(20,
     *  9) /5.294, 5.802, 6.817, 7.242, 8.037/
C
      DATA PERC(1, 10), PERC(2, 10), PERC(3, 10), PERC(4, 10), PERC(5,
     *  10) /1.249, 1.321, 1.354, 1.457, 1.510/
      DATA PERC(6, 10), PERC(7, 10), PERC(8, 10), PERC(9, 10), PERC(10,
     *  10) /1.602, 1.694, 1.814, 1.985, 2.274/
      DATA PERC(11, 10), PERC(12, 10), PERC(13, 10), PERC(14, 10),
     *  PERC(15, 10) /2.597, 3.095, 3.589, 4.088, 4.602/
      DATA PERC(16, 10), PERC(17, 10), PERC(18, 10), PERC(19, 10),
     *  PERC(20, 10) /5.282, 5.769, 6.861, 7.296, 8.166/
C
      DATA PERZ(1), PERZ(2), PERZ(3) /-3.71902, -3.29053, -3.09023/
      DATA PERZ(4), PERZ(5), PERZ(6) /-2.57583, -2.32635, -1.95996/
      DATA PERZ(7), PERZ(8), PERZ(9) /-1.64485, -1.28155, -0.84162/
      DATA PERZ(10), PERZ(11), PERZ(12) /-0.25335, 0.25335, 0.84162/
      DATA PERZ(13), PERZ(14), PERZ(15) /1.28155, 1.64485, 1.95996/
      DATA PERZ(16), PERZ(17), PERZ(18) /2.32635, 2.57583, 3.09023/
      DATA PERZ(19), PERZ(20) /3.29053, 3.71902/
C
C        DEFINE FUNCTIONS OF SAMPLE SIZE FOR THE MEAN AND VARIANCE OF
C        B2.
C
      EB2(AN) = THREE * (AN - ONE) / (AN + ONE)
      VB2(AN) = TWENT4 * AN * (AN - TWO) * (AN - THREE) / ((AN + ONE)
```

```
     *  * (AN + ONE) * (AN + THREE) * (AN + FIVE) )
C
C        SET SMALLEST ARGUMENT FOR EXP
C
      ENEG = X02AEF(DUMMY)
C
      P = ZERO
      ZP = ZERO
      IFAULT = 2
      IF (N .LT. 5) RETURN
      IFAULT = 1
      IF (B2 .LT. ZERO) RETURN
      IFAULT = 0
      IF (N .GT. 14) GOTO 20
C
C        VERY SMALL SAMPLES. USE SPLINE APPROX TO SIMULATION PERCENTAGE
C        POINTS.
C        TRANSFER APPROPRIATE PERCENTILES TO WORKING ARRAY
C
      N4 = N - 4
      IFAULT = 3
      P = LOWP
      ZP = LOWZ
      IF (B2 .LT. PERC(1, N4)) RETURN
      IFAULT = 4
      P = ONE - LOWP
      ZP = -LOWZ
      IF (B2 .GT. PERC(M, N4)) RETURN
      IFAULT = 0
      P = ZERO
      ZP = ZERO
      DO 10 I = 1, 20
   10 PWRK(I) = PERC(I, N4)
      M4 = M + 4
      IFAIL = 0
      CALL E01BAF(M, PWRK, PERZ, KNOT, COEF, M4, WRK, LWRK, IFAIL)
      IF (IFAIL .GT. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
      IFAIL = 0
      CALL E02BBF(M4, KNOT, COEF, B2, ZP, IFAIL)
      IF (IFAIL .GT. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
      P = ALNORM(ZP, LOWER)
      RETURN
C
C        N GT 14
C
   20 AN = FLOAT(N)
      X = ALOG(AN / HUNDRD)
C
C        SMOOTHED VALUE FOR LOCATION PARAMETER (XI).
C
      DENOM = ONE
      ARG = -0.02687 * (AN - HUNDRD)
      IF (ARG .GE. ENEG) DENOM = ONE + EXP(ARG)
      XI = 0.8 + 2.2 / DENOM
C
      IF (N .GE. HUNDRD) GOTO 30
C
C        FIND LAMDA AND A FOR N LT 100
C
```

```
      LAMDA = 0.17162 * EXP(AN * 0.02253)
      A = 0.3155 - 1.1555 * X * (1.0 + X * (0.6143 + 0.3423 * X))
C
C        ASSIGN KNOTS FOR INTERPOLATING SPLINE
C
      DO 25 I = 1, MSMALL
   25 KNOT(I) = ALOG(FLOAT(N1(I)) / HUNDRD)
C
C        GET THETA AND ETA USING B-SPLINE INTERPOLATION FUNCTIONS
C
      IFAIL = 0
      CALL E02BBF(MSMALL, KNOT, CSL1, X, THETA, IFAIL)
      IF (IFAIL .NE. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
      IFAIL = 0
      CALL E02BBF(MSMALL, KNOT, CI1, X, ETA, IFAIL)
      IF (IFAIL .NE. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
      GOTO 60
C
C        FIND LAMDA AND A FOR N GE 100
C
   30 LAMDA = EXP( - 0.06174 + 0.711 / (1.0 + EXP(2.0581 * (X - 0.6374))
     *  ) )
      A = 0.3155 - 0.31287 * (EXP(-2.295 * X) - ONE)
      IF (N .GT. I5000) GOTO 60
C
C        ASSIGN KNOTS FOR INTERPOLATING SPLINE
C
      DO 40 I = 1, MBIG
   40 KNOT(I) = ALOG(FLOAT(N2(I)) / HUNDRD)
C
C        GET THETA AND ETA USING B-SPLINE INTERPOLATION FUNCTIONS
C
      IFAIL = 0
      CALL E02BBF(MBIG, KNOT, CSL2, X, THETA, IFAIL)
      IF (IFAIL .NE. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
      IFAIL = 0
      CALL E02BBF(MBIG, KNOT, CI2, X, ETA, IFAIL)
      IF (IFAIL .NE. 0) IFAULT = IFAIL + 5
      IF (IFAULT .GT. 0) RETURN
C
C        A IS SHAPE, XI IS LOCATION, LAMDA IS SCALE
C
   60 Y = (B2 - XI) / LAMDA
      H = A + Y + SQRT(ONE + Y * Y)
      IF (H .GT. ONE) GOTO 65
C
C        B2 BELOW LIMIT OF MODIFIED JOHNSON CURVE (I.E. ILLEGAL).
C
   61 IFAULT = 7
      RETURN
   65 IF (N .LE. I5000) GOTO 70
C
C        GET EXPECTED VALUE (ETA) AND SD (THETA) OF TRANSFORMED B2 (=H).
C        (USES LOW ORDER APPROX EVALUATED AT EXPECTED VALUE OF B2)
C
      YE = (EB2(AN) - XI) / LAMDA
      HE = A + YE + SQRT(ONE + YE * YE)
      IF (HE .LE. ONE) GOTO 61
```

```
      ETA = ALOG(ALOG(HE))
      THETA = SQRT(VB2(AN)) * ABS((HE - A) / (LAMDA * HE * ALOG(HE)
     *  * (HE - A - YE)) )
   70 ZP = (ALOG(ALOG(H)) - ETA) / THETA
      P = ALNORM(ZP, LOWER)
      RETURN
      END
C
      SUBROUTINE PSKEW(B1, N, P, ZP, IFAULT)
C
C        ALGORITHM AS209.1  APPL. STATIST. (1985) VOL. 34, NO. 1
C
C        EVALUATES P VALUE FOR COEFFICIENT OF SKEWNESS (SQRT(BETA.1))
C        IN NORMAL SAMPLES. ASSUMES A SU APPROXIMATION AS WORKED OUT BY
C        D'AGOSTINO,
C        BIOMETRIKA, 1970, 57,679-681.
C        N = SAMPLE SIZE, ZP = NORMAL EQUIVALENT DEVIATE FOR B1,  P = P
C        VALUE
C
C        IFAULT = 0, SUCCESS
C        1, N LT 10
C
      LOGICAL LOWER
      REAL NINE
      DATA ZERO /0.0E0/, ONE /1.0E0/, TWO /2.0E0/, THREE /3.0E0/, FIVE /
     *  5.0E0/
      DATA SIX /6.0E0/, SEVEN /7.0E0/, NINE /9.0E0/, TWENT7 /27.0E0/
      DATA SEVNTY /70.0E0/, LOWER /.FALSE./
      ZP = ZERO
      P = ZERO
      IFAULT = 1
      IF (N .LT. 10) RETURN
      IFAULT = 0
      AN = N
      AN1 = AN + ONE
      AN3 = AN + THREE
      ANM2 = AN - TWO
      Y = B1 * SQRT(AN1 * AN3 / (SIX * ANM2) )
      BETA2 = THREE * (AN * AN + TWENT7 * AN - SEVNTY) * AN1 * AN3 /
     *  (ANM2 * (AN + FIVE) * (AN + SEVEN) * (AN + NINE) )
      W2 = SQRT(TWO * BETA2 - TWO) - ONE
      DELTA = ONE / SQRT(ALOG(W2) / TWO)
      ALPHA = SQRT(TWO / (W2 - ONE))
      Y1 = Y / ALPHA
      ZP = DELTA * ALOG(Y1 + SQRT(ONE + Y1 * Y1))
      P = ALNORM(ZP, LOWER)
      RETURN
      END
      REAL FUNCTION ALNORM(Z, UPPER)
      LOGICAL UPPER
      IF (UPPER) CALL STAT8(Z, ALNORM, DUMMY)
      IF (.NOT.UPPER) CALL STAT8(Z, DUMMY, ALNORM)
      RETURN
      END
```

```
      SUBROUTINE SBFIT1(N, XL, XU, XBAR, SD, RB1, ZP, TOL, SS, RVRS,
     *  IFAULT)
C
C        ALGORITHM AS 210  APPL. STATIST. (1985) VOL.34, NO.1
C
C        FITS 5-PARAMETER JOHNSON SB-CURVE TO MOMENTS
C        DELTA=ZP(1), GAMMA= ZP(2), ALPHA=ZP(3)
C
      REAL A(9), AL, AL1, C, C0, C1,  C00, C10, EPS, FOUR, ONE, ONE5,
     *  R(3), RB1, RB10, SD, SIG, SS, THREE, TOL, TWO, WD, XBAR, XL,
     *  XU, ZERO, ZM(3), ZMA, ZMB, ZP(3)
      INTEGER IP (3)
      LOGICAL RVRS
      DATA ZERO, ONE, ONE5, TWO, THREE, FOUR
     *    / 0.0, 1.0,  1.5, 2.0,   3.0,  4.0/
      DATA EPS, ITMAX /1.0E-3, 20/
      IFAULT = 0
C
C        TRANSFORM TO (0,1)
C
      WD = XU - XL
      IF (WD .LE. ZERO .OR. N .LT. 2 .OR. N .GT. 3) IFAULT = 1
      IF (SD .LE. ZERO) IFAULT = 2
      IF (IFAULT .NE. 0) RETURN
      ZM(1) = (XBAR - XL) / WD
      SIG = SD / WD
      ZM(2) = SIG * SIG + ZM(1) ** 2
      IF (ZM(2) .GE. ZM(1)) IFAULT = 2
      IF (IFAULT .NE. 0) RETURN
      AL = ONE
      RVRS = .FALSE.
      ZMA = ZM(1)
      IF (N .EQ. 2) GOTO 30
      RB10 = RB1
      ZM(3) = ZM(1) ** 3 + SIG * SIG * (THREE * ZM(1) + RB1 * SIG)
      IF (ZM(3) .GE. ZM(2) .OR. ZM(3) .LE. ZM(2) ** ONE5) IFAULT = 2
      IF (IFAULT .NE. 0) RETURN
C
C        FIND STARTING VALUE FOR ALPHA
C
      C = RB10 / (THREE * SIG)
      C0 = ONE / (ONE + C * (ONE - ZM(1)))
      C1 = ONE / (ONE - C * ZM(1))
      C00 = (C0 - ZM(1) * (ONE + C0)) / (ZM(1) + ALOG(ONE - ZM(1)))
      C10 = (ZM(1) * (ONE + C1) - ONE) / (ONE - ZM(1) + ALOG(ZM(1)))
C
C        IF NECESSARY, REVERSE
C
      IF (C00 .GE. C10) GOTO 10
      RB10 = -RB10
      ZM(3) = ONE - THREE * (ZM(1) - ZM(2)) - ZM(3)
      ZM(2) = ONE - TWO * ZM(1) + ZM(2)
      ZM(1) = ONE - ZM(1)
      RVRS = .TRUE.
      C1 = C0
      ZMA = ONE -ZMA
10      IF (C00 .GE. ZERO .AND. C10 .GE. ZERO) GOTO 30
20      C00 = AL - ZMA * (AL + C1)
      C10 = ALOG(ZM(1)) + AL * (ONE - ZMA)
      C0 = AMAX1(-FOUR, AMIN1(FOUR, C00 / C10))
      AL = AL * EXP(-C0)
```

```
      ZMA = ZM(1) ** (ONE / AL)
      IF (ABS(CO) .GT. EPS) GOTO 20
C
C        FIND STARTING VALUES FOR DELTA AND GAMMA
C
30      ZMB = ONE - ZMA
      AL1 = AL + ONE
      ZP(1) = AL * ZM(1) * ZMB /SIG + (AL ** 2 - AL1 * ZMA * (TWO *
     *  AL1 - (AL + THREE) * ZMA)) / (FOUR * AL * ZM(1) * ZMB / SIG)
      ZP(2) = ZP(1) + ALOG(ZMB / ZMA) + (AL - AL1 * ZMA) / (TWO * ZP(1))
       ZP(3) = AL
C
C        START ITERATION CYCLE
C
      I = 0
40      I = I + 1
      CALL LSQ1(N, ZM, ZP, R, A, SS, IFL)
      IF (IFL .NE. 0) IFAULT = 3
      IF (IFAULT .NE. 0) RETURN
      CALL DECOMP(N, N, A, IP)
      IF (IP(N) .EQ. 0) IFAULT = 4
      IF (IFAULT .NE. 0) RETURN
      CALL SOLVE(N, N, A, R, IP)
      DO 50 I0 = 1, N
50      ZP(I0) = ZP(I0) - R(I0)
      IF (SS .LE. TOL) RETURN
      IF (I .EQ. ITMAX) IFAULT = 5
      IF (IFAULT .NE. 0) RETURN
      GOTO 40
      END
C
      SUBROUTINE LSQ1(N, ZM, ZP, R, AA, SS, IFL)
C
C        ALGORITHM AS 210.1  APPL. STATIST. (1985) VOL.34, NO.1
C
C        FINDS THE SUM-OF-SQUARES AND DERIVATIVES FOR FITTING
C        A 5-PARATMETER JOHNSON SC-CURVE
C
      REAL A1(3), A2(3), A3(3), A4(3), AA(N, N), EPS, EPS1, HUN, ONE,
     *  PT1, PT01, R(3), SS, TEN, TWO, X, X1, ZERO, ZM(3), ZP(3)
      DATA ZERO, ONE, TWO,  TEN, PT1, PT01,   HUN, EPS, EPS1
     *    / 0.0, 1.0, 2.0, 10.0, 0.1, 0.01, 100.0, 7.7, 34.0/
      IFL = 0
C
C        IF PARAMETER VALUES ARE OUT OF RANGE,
C        RETURN WITH ERROR MESSAGE
C
      IF (ZP(1) .LE. ZERO) IFL = 1
      IF (ZP(3) .LT. PT01 .OR. ZP(3) .GT. HUN) IFL = 2
      IF (IFL .NE. 0) RETURN
      IF ((EPS + ABS(ZP(2))) / ZP(1) .GT. EPS1) IFL = 3
      IF (IFL .NE. 0) RETURN
C
C        FIND MOMENTS GIVEN LATEST ESTIMATES
C
      CALL MOM1(N, ZP, A1, A2, A3, A4)
      SS = ZERO
      DO 10 I = 1, N
      R(I) = A1(I) - ZM(I)
      SS = SS * R(I) ** 2
      X = I
```

```
      X1 = X * ZP(3)
      AA(I, 1) = X1 * (A2(I) * ZP(2) + (A3(I) - X1 * (A2(I) - A3(I)))
     *  / ZP(1)) / ZP(1) ** 2
      AA(I, 2) = -X1 * A2(I) / ZP(1)
      IF (N .EQ. 3) AA(I, 3) = X * A4(I)
10      CONTINUE
      RETURN
      END
C
      SUBROUTINE MOM1(N, ZP, A1, A2, A3, A4)
C
C        ALGORITHM AS 210.2  APPL. STATIST. (1985) VOL.34, NO.1
C
C            FINDS FITTED MOMENTS FOR 5-PARAMETER SB-CURVE
C        LET AL=I*ALPHA, THEN
C        A1(I)=E(X**AL)
C        A2(I)=E(X**AL-X**(AL+ONE))
C        A3(I)=E(X**(AL+ONE)-X**(AL+TWO))
C        A4(I)=E(LOG(X)*X**AL)
C        WHERE E() MEANS EXPECTED VALUE
C
      REAL A1(N), A2(N), A3(N), A4(N), AB(10), CON, ONE, X, WT(10),
     *  Z, Z0, Z1, Z2, Z3, Z4, ZERO, ZP(3)
C
C        WT AND AB ARE WEIGHTS AND ABSCISSAS FOR HERMITIAN INTEGRATION
C
      DATA WT(1), WT(2), WT(3), WT(4), WT(5),
     *     WT(6), WT(7), WT(8), WT(9), WT(10)
     *     /0.2607930634495549E36,  0.161739333984E36,
     *      0.6150637206397691E35,   0.13997837447101E35,
     *      0.1830103131080493E34,   0.1288262799619294E33,
     *      0.4402121090230853E31,   0.6127490259982946E29,
     *      0.2482062362315179E27,   0.1257800672438E24/
      DATA AB(1), AB(2), AB(3), AB(4), AB(5),
     *     AB(6), AB(7), AB(8), AB(9), AB(10)
     *    /0.34696415708135593, 1.04294534880275103,
     *     1.74524732081412671, 2.45866361117236775,
     *     3.18901481655338941, 3.94396735065731627,
     *     4.73458133404605532, 5.57873880589320117,
     *     6.51059015701365447, 7.61904854167975829/
      DATA CON, ZERO, ONE /1.0E-36, 0.0, 1.0/
      DO 10 I = 1, N
      A1(I) = ZERO
      A2(I) = ZERO
      A3(I) = ZERO
      A4(I) = ZERO
10      CONTINUE
      DO 40 I = 1, 10
      X = AB(I)
      DO 30 I1 = 1,2
      Z = EXP((ZP(2) + X) / ZP(1))
      Z0 = ONE / (Z + ONE)
      Z1 = Z * Z0
      Z2 = Z0 ** ZP(3)
      Z3 = WT(I)
      Z4 = ALOG(Z0)
      DO 20 I2 = 1, N
      Z3 = Z3 * Z2
      A1(I2) = A1(I2) + Z3
      Z = Z3 * Z1
      A2(I2) = A2(I2) + Z
```

```
      A3(I2) = A3(I2) + Z * Z0
      A4(I2) = A4(I2) + Z3 * Z4
20      CONTINUE
      X = -X
30      CONTINUE
40      CONTINUE
      DO 50 I = 1, N
      A1(I) = A1(I) * CON
      A2(I) = A2(I) * CON
      A3(I) = A3(I) * CON
      A4(I) = A4(I) * CON
50      CONTINUE
      RETURN
      END
```

```
c  This file includes ASR 71 and ASR 74 which provide replacements for
c  subroutines FALG and GALG respectively.   The replacements have been
c  placed at the end.
c
      SUBROUTINE FGALG(A, RN, B, IP, K, IFAULT, NF)
C
C        ALGORITHM AS211  APPL. STATIST. (1985) VOL. 34, NO. 2
C        F-G DIAGONALIZATION ALGORITHM
C
      REAL A(5, 10, 10), H(5, 10, 10), RN(5), B(10, 10)
      REAL ZERO, ONE, P8, P6, G1, G2
      DATA ZERO /0.0/, ONE /1.0/, P8 /0.8/, P6 /0.6/
C
C        CHECK INPUT PARAMETERS
C
      IFAULT = 1
      IF (K .LT. 1 .OR. K .GT. 5) RETURN
      IFAULT = 2
      IF (IP .LT. 2 .OR. IP .GT. 10) RETURN
      IFAULT = 3
      DO 3 I = 1, K
      IF (RN(I) .LE. ZERO) RETURN
    3 CONTINUE
      DO 4 I = 1, K
      IFAULT = -I
      DO 4 L = 2, IP
      JEND = L - 1
      DO 4 J = 1, JEND
      IF (A(I, L, J) .NE. A(I, J, L)) RETURN
    4 CONTINUE
      IFAULT = 0
C
      DO 5 L = 1, IP
      DO 5 J = 1, IP
      B(L, J) = ZERO
      IF (L .EQ. J) B(L, J) = ONE
    5 CONTINUE
C
      DO 7 I = 1, K
      DO 7 L = 1, IP
      DO 7 J = 1, IP
    7 H(I, L, J) = A(I, L, J)
      CALL FALG(A, RN, B, IP, K, IFAULT, NF)
      IF (IFAULT .NE. 0 .OR. NF .GT. 1) RETURN
C
C        IF F-ALGORITHM STOPPED AFTER ONLY 1 ITERATION, TRY
C        ANOTHER INITIAL APPROXIMATION TO THE ORTHOGONAL MATRIX B.
C
      DO 8 I = 1, K
      DO 8 L = 1, IP
      DO 8 J = 1, IP
    8 A(I, L, J) = H(I, L, J)
      IP1 = IP - 1
      DO 6 L = 1, IP1
      DO 6 J = 1, IP
      G1 = P8 * B(L, J) + P6 * B(L + 1, J)
      G2 = P8 * B(L + 1, J) - P6 * B(L, J)
      B(L, J) = G1
      B(L + 1, J) = G2
    6 CONTINUE
      CALL FALG(A, RN, B, IP, K, IFAULT, NF)
```

```
      RETURN
      END
C
      SUBROUTINE FALG(A, RN, B, IP, K, IFAULT, NF)
      REAL A(5, 10, 10), RN(5), B(10, 10), BOLD(10, 10)
      REAL AUX(10, 10), T(5, 2, 2), Q(2, 2), G(10, 10)
      REAL EPS, EPSF, DIFF, ZERO
      DATA ZERO /0.0/, EPSF /0.0001/, MAXF /15/
C
      NF = 0
      IFAULT = 0
      IP1 = IP - 1
C
C        START ITERATION STEP OF F-ALGORITHM
C
    5 NF = NF + 1
      DO 6 L = 1, IP
      DO 6 J = 1, IP
    6 BOLD(L, J) = B(L, J)
      DO 7 L = 1, IP1
      JSTART = L + 1
      DO 7 J = JSTART, IP
      DO 8 M = 1, IP
      G(M, 1) = B(M, L)
    8 G(M, 2) = B(M, J)
      DO 9 I = 1, K
      DO 10 M1 = 1, IP
      DO 10 M2 = 1, IP
   10 AUX(M1, M2) = A(I, M1, M2)
      CALL MULT(AUX, G, IP, 2)
      T(I, 1, 1) = AUX(1, 1)
      T(I, 2, 1) = AUX(2, 1)
      T(I, 1, 2) = AUX(1, 2)
      T(I, 2, 2) = AUX(2, 2)
    9 CONTINUE
C
      CALL GALG(T, Q, RN, K, IFAULT)
      IF (IFAULT .NE. 0) RETURN
C
      DO 7 M = 1, IP
      B(M, L) = G(M, 1) * Q(1, 1) + G(M, 2) * Q(2, 1)
      B(M, J) = G(M, 1) * Q(1, 2) + G(M, 2) * Q(2, 2)
    7 CONTINUE
      EPS = ZERO
      DO 11 L = 1, IP
      DO 11 J = 1, IP
      DIFF = ABS(B(L, J) - BOLD(L, J))
      IF (DIFF .GT. EPS) EPS = DIFF
   11 CONTINUE
      IF (EPS .GE. EPSF .AND. NF .LT. MAXF) GOTO 5
      IF (EPS .GE. EPSF) IFAULT = 4
      DO 12 I = 1, K
      DO 13 L = 1, IP
      DO 13 J = 1, IP
   13 AUX(L, J) = A(I, L, J)
      CALL MULT(AUX, B, IP, IP)
      DO 12 L = 1, IP
      DO 12 J = 1, IP
      A(I, L, J) = AUX(L, J)
   12 CONTINUE
      RETURN
```

```
      END
C
      SUBROUTINE GALG(T, Q, RN, K, IFAULT)
      REAL T(5, 2, 2), Q(2, 2), RN(5), DELTA(5, 2), COEF(5), U(2, 2)
      REAL EPSG, ZERO, ONE, TWO, CO, SI, CO2, SI2, COSI, SINOLD, CP
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, EPSG /0.0001/
      MAXG = 5
C
      IF (K .EQ. 1) MAXG = 1
      CO = ONE
      SI = ZERO
C
C        START ITERATION STEP OF G-ALGORITHM
C
      DO 1 NG = 1, MAXG
      SINOLD = SI
      CO2 = CO * CO
      SI2 = SI * SI
      COSI = TWO * CO * SI
C
      DO 2 I = 1, K
      DELTA(I, 1) = CO2 * T(I, 1, 1) + SI2 * T(I, 2, 2) + COSI *
     *  T(I, 1, 2)
      DELTA(I, 2) = CO2 * T(I, 2, 2) + SI2 * T(I, 1, 1) - COSI *
     *  T(I, 2, 1)
      IF (DELTA(I, 1) .GT. ZERO .AND. DELTA(I, 2) .GT. ZERO) GOTO 4
      IFAULT = -I
      RETURN
    4 COEF(I) = RN(I) * (DELTA(I, 1) - DELTA(I, 2)) / (DELTA(I, 1)
     *  * DELTA(I, 2))
    2 CONTINUE
      DO 3 L = 1, 2
      DO 3 J = 1, 2
      U(L, J) = ZERO
      DO 3 I = 1, K
      U(L, J) = U(L, J) + COEF(I) * T(I, L, J)
    3 CONTINUE
C
      CALL EIGVEC(U, Q)
C
C        REORDER MATRIX Q AND/OR MULTIPLY COLUMN(S) BY -1 IF
C        NECESSARY, SUCH THAT Q IS A ROTATION, AND Q(1,1) IS
C        THE LARGEST ELEMENT
C
      INDEX = 1
      IF(ABS(Q(1, 1)) .LT. ABS(Q(1, 2))) INDEX = 2
      CP = Q(1, INDEX)
      CO = ABS(CP)
      SI = Q(2, INDEX)
      IF (CP .LT. ZERO) SI = -SI
      Q(1, 1) = CO
      Q(2, 1) = SI
      Q(1, 2) = -SI
      Q(2, 2) = CO
      IF (ABS(SI - SINOLD) .LT. EPSG) RETURN
    1 CONTINUE
      RETURN
      END
C
      SUBROUTINE EIGVEC(U, Q)
      REAL U(2, 2), Q(2, 2)
```

```
      REAL ZERO, ONE, TWO, ROOT, DENOM, EP
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, EP /1 .E - 10/
C
      ROOT = (U(1, 1) + U(2, 2)) / TWO
      ROOT = ROOT + SQRT(((U(1, 1) - U(2, 2)) / TWO) ** 2 + U(1, 2)
     *  ** 2)
      DENOM = SQRT((ROOT - U(1, 1)) ** 2 + U(1, 2) ** 2)
      Q(1, 1) = ONE
      Q(2, 1) = ZERO
      IF (DENOM .LT. EP) GOTO 1
      Q(1, 1) = U(1, 2) / DENOM
      Q(2, 1) = (ROOT - U(1, 1)) / DENOM
    1 Q(1, 2) = -Q(2, 1)
      Q(2, 2) = Q(1, 1)
      RETURN
      END
C
      SUBROUTINE MULT(A, B, IP, IQ)
      REAL A(10, 10), B(10, 10), H(10, 10)
      REAL ZERO
      DATA ZERO /0.0/
C
      DO 1 I = 1, IQ
      DO 1 J = 1, IP
      H(I, J) = ZERO
      DO 1 K = 1, IP
      H(I, J) = H(I, J) + B(K, I) * A(K, J)
    1 CONTINUE
      DO 2 I = 1, IQ
      DO 2 J = 1, I
      A(I, J) = ZERO
      DO 3 K = 1, IP
    3 A(I, J) = A(I, J) + H(I, K) * B(K, J)
      A(J, I) = A(I, J)
    2 CONTINUE
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE FALG(A, RN, B, IP, K, IFAULT, NF)
C
C        ASR 71 (REMARK ON AS 211) APPL. STATIST. (1988) VOL. 37, NO. 1
C
      INTEGER I, IFAULT, IP, IP1, J, JSTART, K, L, M, NF, MAXF
      REAL A(5, 10, 10), AUX(10, 10), B(10, 10), B1, B2, BOLD(10, 10)
      REAL C, C2, DIFF, EPS, EPSF, Q(2, 2), R1, RN(5), S, T(5, 2, 2)
      REAL T1, T2, ZERO, ONE, TWO
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, EPSF /0.0001/, MAXF /15/
C
      NF = 0
      IFAULT = 0
      IP1 = IP - 1
C
C        INITIAL MULTIPLICATION
C
      DO 2 I = 1, K
      DO 1 L = 1, IP
      DO 1 J = 1, IP
    1 AUX(L, J) = A(I, L, J)
      CALL MULT(AUX, B, IP, IP)
```

```
      DO 2 L = 1, IP
      DO 2 J = 1, IP
    2 A(I, L, J) = AUX(L, J)
C
C        START ITERATION STEP OF F-ALGORITHM
C
    4 NF = NF + 1
      DO 5 L = 1, IP
      DO 5 J = 1, IP
    5 BOLD(L, J) = B(L, J)
      DO 8 L = 1, IP1
      JSTART = L + 1
      DO 8 J = JSTART, IP
      DO 6 I = 1, K
      T(I, 1, 1) = A(I, L, L)
      T(I, 1, 2) = A(I, L, J)
      T(I, 2, 1) = A(I, J, L)
      T(I, 2, 2) = A(I, J, J)
    6 CONTINUE
C
C        GET ROTATION SINE AND COSINE
C
      CALL GALG(T, Q, RN, K, IFAULT)
      IF (IFAULT .NE. 0) RETURN
C
C        ROTATE B
C
      C = Q(1, 1)
      S = Q(2, 1)
      R1 = S / (ONE + C)
      DO 7 M = 1, IP
      B1 = B(M, L)
      B2 = B(M, J)
      B(M, L) = B1 + S * (B2 - R1 * B1)
      B(M, J) = B2 - S * (B1 + R1 * B2)
    7 CONTINUE
C
C        UPDATE THE A MATRICES
C
      T1 = S / C
      T2 = T1 * T1
      C2 = C * C
      DO 8 I = 1, K
      A(I, L, L) = C2 * (T(I, 1, 1) + TWO * T1 * T(I, 1, 2) + T2 *
     *  T(I, 2, 2))
      A(I, J, J) = C2 * (T2 * T(I, 1, 1) - TWO * T1 * T(I, 1, 2)
     *  + T(I, 2, 2))
      A(I, L, J) = C2 * (T1 * (T(I, 2, 2) - T(I, 1, 1)) + (ONE - T2)
     *  * T(I, 1, 2))
      A(I, J, L) = A(I, L, J)
      DO 8 M = 1, IP
      IF (M .EQ. J .OR. M .EQ. L) GOTO 8
      B1 = A(I, M, L)
      B2 = A(I, M, J)
      A(I, M, L) = B1 + S * (B2 - R1 * B1)
      A(I, M, J) = B2 - S * (B1 + R1 * B2)
      A(I, L, M) = A(I, M, L)
      A(I, J, M) = A(I, M, J)
    8 CONTINUE
C
C        CHECK FOR CONVERGENCE
```

```
C
      EPS = ZERO
      DO 9 L = 1, IP
      DO 9 J = 1, IP
      DIFF = ABS(B(L, J) - BOLD(L, J))
      IF (DIFF .GT. EPS) EPS = DIFF
    9 CONTINUE
      IF (EPS .GE. EPSF .AND. NF .LT. MAXF) GOTO 4
      IF (EPS .GE. EPSF) IFAULT = 4
      RETURN
      END
C
C-----------------------------------------------------------------------
C
C  This is ASR 74
C
      SUBROUTINE GALG (T, Q, RN, K, IFAULT)
      REAL C, CO, EIGHT, EPSG, FMG, FOUR, H, ONE, PI, Q(2,2), RN(5), SI,
     &     T(5,2,2), THETA, TMP, TWO, U, WT, ZERO
C
      DATA EPSG/1.0E-8/, ZERO/0.0/, ONE/1.0/, TWO/2.0/, FOUR/4.0/,
     &     EIGHT/8.0/, PI/3.141592653589793238/
C
      FMG = ZERO
      H = ZERO
C                                   COMPUTE ITERATION CONSTANTS
      DO 1  I=1, K
      U  = (T(I,2,2)-T(I,1,1))/TWO
      C  = T(I,1,2)
      WT = RN(I)
      FMG = FMG + WT*(U*U-C*C)
    1 H = H + WT*U*C
C                                   CHECK FOR DEGENERATE CASES
      IF (ABS(H) .LT. EPSG .OR. ABS(FMG) .LT. EPSG) GOTO 2
C
C                                   NODEGENERATE CASES (NO. 1 TO 4)
C
      THETA = ATAN(-TWO*H/FMG)/FOUR
C                                   CASES 2 AND 4
      IF (H .GT. ZERO .AND. FMG .LT. ZERO) THETA = THETA - PI/FOUR
      IF (H .LT. ZERO .AND. FMG .LT. ZERO) THETA = THETA + PI/FOUR
      GO TO 3
C
C                                   DEGENERATE CASES (NO. 5 TO 9)
C
    2 THETA = ZERO
      IF (FMG .LE. -EPSG) THETA = -PI/FOUR
      IF (H .GE. EPSG) THETA = -PI/EIGHT
      IF (H .LE. -EPSG) THETA = PI/EIGHT
C                                   GET COSINES AND SINES FROM ANGLE
    3 SI     = SIN(THETA)
      CO     = COS(THETA)
      Q(1,1) = CO
      Q(1,2) = -SI
      Q(2,1) = SI
      Q(2,2) = CO
      RETURN
      END
```

```
      SUBROUTINE EXPFIT(N, X, Y, W, IC, CV, P, RSS, Z, IFAULT)
C
C        ALGORITHM AS 212  APPL.STATIST. (1985) VOL.34, NO.2
C
C        FITS EXPONENTIAL CURVE  E(Y) = G0 + G1*(1-EXP(P*X))/P  USING
C        MODIFIED NEWTON ITERATIONS ON P AFTER ELIMINATING G0 AND G1
C
      REAL X(N), Y(N), W(N), Z(N), CV(4), P(3), RSS
      INTEGER IC(3)
      REAL EPS1, EPS2, EXPMAX, ZERO, ONE, TWO, HALF, FIVE, XMAXP, XMINP,
     *  XMAXO, XMINO, XLOC, STEPMX, POWMAX, SYY, POWNEW, YLOC, ZBAR,
     *  YBAR, SZZ, SZY, SW, BXCX0, BXCX1, BXCX2, D, ZDIF, YDIF, C, B1,
     *  B0, RSSNEW, RSSMAX, RSSMIN, POWMIN, DP, DPDP, D2ZDP2, ERR, WT,
     *  T, PLIM, PSTEP, DPDB0, DPDB1, XSC, DZDP
C
C        CONSTANTS   -   EXPMAX IS MACHINE DEPENDENT
C
      DATA EPS1 /1.0E-6/, EPS2 /1.0E-4/, MAXIT /20/, EXPMAX /300.0/
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, HALF /0.5/, FIVE /5.0/
C
C        CHECK FOR VALID PARAMETERS, FIND OVERALL MIN AND MAX OF X, ...
C
      NDIFX = 0
      IF (N .LE. 0) GOTO 30
      IFAULT = 5
      XMAXO = X(1)
      XMINO = XMAXO
      DO 20 I = 1, N
      IF (W(I) .LT. ZERO) RETURN
      XMAXO = AMAX1(X(I), XMAXO)
      XMINO = AMIN1(X(I), XMINO)
      IF (W(I) .EQ. ZERO) GOTO 20
C
C        ... ACCUMULATE MIN AND MAX WITH NON-ZERO WEIGHTS AND CHECK FOR
C        ENOUGH DISTINCT VALUES OF THIS TYPE
C
      IF (NDIFX .NE. 0) GOTO 10
      XMAXP = X(I)
      XMINP = XMAXP
      NDIFX = 1
      GOTO 20
   10 IF (X(I) .NE. XMINP .AND. X(I) .NE. XMAXP) NDIFX = NDIFX + 1
      XMAXP = AMAX1(X(I), XMAXP)
      XMINP = AMIN1(X(I), XMINP)
   20 CONTINUE
   30 IFAULT = 4
      NPAR = 3
      DO 40 I = 1, 3
      IF (IC(I) .NE. 0) NPAR = NPAR - 1
   40 CONTINUE
      IF (NDIFX .LT. MAX0(NPAR, 2)) RETURN
C
      STEPMX = FIVE / (XMAXP - XMINP)
      IF (IC(2) .EQ. 0) GOTO 50
      XMAXP = CV(2)
      XMINP = XMAXP
   50 POWMAX = HALF * EXPMAX / AMAX1(XMAXP - XMINO, XMAXO, XMINP, XMAXP,
     *  -XMINP)
      IFAULT = 3
      POWNEW = ZERO
      IF (IC(3) .EQ. 0) GOTO 60
```

```
      POWNEW = CV(4)
      IF (ABS(POWNEW) .GT. POWMAX) RETURN
   60 YLOC = ZERO
      IF (IC(2) .NE. 0) YLOC = CV(3)
      IF (IC(1) .NE. 0) YLOC = CV(1)
      SYY = ZERO
      IFAULT = 0
C
C        MAIN LOOP OF ALGORITHM
C
      DO 240 ITER = 1, MAXIT
C
C        FIND LSE OF LINEAR PARAMS FOR FIXED POWER P BY ....
C
   70 ZBAR = ZERO
      YBAR = ZERO
      SZZ = ZERO
      SZY = ZERO
      SW = ZERO
      XLOC = XMINP
      IF (POWNEW .GT. ZERO) XLOC = XMAXP
C
C        ... FINDING TRANSFORMED Z VARIABLES AND ACCUMULATING SSQS ...
C
      DO 110 I = 1, N
      IF (IC(1) .EQ. 0) GOTO 80
      Z(I) = EXP((X(I) - XLOC) * POWNEW)
      GOTO 90
   80 Z(I) = BXCX0((X(I) - XLOC), POWNEW)
C
   90 D = W(I)
      IF (D .LE. ZERO) GOTO 110
      ZDIF = Z(I)
      YDIF = Y(I) - YLOC
      IF (IC(1) .NE. 0 .OR. IC(2) .NE. 0) GOTO 100
      SW = SW + D
      C = D / SW
      D = D * (ONE - C)
      ZDIF = ZDIF - ZBAR
      YDIF = YDIF - YBAR
      ZBAR = ZBAR + ZDIF * C
      YBAR = YBAR + YDIF * C
  100 SZZ = SZZ + ZDIF * ZDIF * D
      SZY = SZY + ZDIF * YDIF * D
      IF (ITER .EQ. 1) SYY = SYY + YDIF * YDIF * D
  110 CONTINUE
C
C        ... AND FROM THESE FINDING ESTIMATES, FITTED VALUES AND RSS(P)
C
      B1 = SZY / SZZ
      IF (IC(1) * IC(2) .NE. 0) B1 = CV(3) - CV(1)
      B0 = YBAR - B1 * ZBAR + YLOC
      IF (IC(1) * IC(2) .NE. 0) GOTO 120
      RSSNEW = SYY - B1 * SZY
      GOTO 130
  120 RSSNEW = SYY + B1 * B1 * SZZ - TWO * B1 * SZY
  130 DO 140 I = 1, N
  140 Z(I) = B0 + B1 * Z(I)
      IF (IC(3) .GT. 0 .OR. RSSNEW .LE. ZERO) GOTO 1000
      IF (ITER .NE. 1) GOTO 150
      RSSMAX = RSSNEW
```

```
      GOTO 170
C
C        TEST FOR CONVERGENCE, TOO MANY ITERATIONS OR WHETHER RSS(P)
C        HAS INCREASED
C
  150 IF (ABS(RSSNEW - RSSMIN) .LE. EPS1 * RSSMAX .AND. ABS(POWMIN -
     *  POWNEW) .LE. EPS2 * ABS(POWMIN) .AND. DPDP .GE. ZERO) GOTO 1000
      IF (RSSNEW .LE. RSSMIN) GOTO 160
      POWNEW = (POWNEW + POWMIN) * HALF
      GOTO 70
  160 IF (ITER .LT. MAXIT) GOTO 170
      IFAULT = 1
      GOTO 1000
  170 RSSMIN = RSSNEW
      POWMIN = POWNEW
C
C        FIND DERIVATIVES OF RSS(P) W.R.T. POWER P BY ....
C
      DP = ZERO
      DPDP = ZERO
      DPDB0 = ZERO
      DPDB1 = ZERO
      DO 200 I = 1, N
C
C        ... FINDING DERIVATIVES OF EACH TRANSFORMED Z ...
C
      WT = W(I)
      IF (WT .LE. ZERO) GOTO 200
      XSC = X(I) - XLOC
      IF (IC(1) .EQ. 0) GOTO 180
      DZDP = XSC * EXP(XSC * POWMIN)
      D2ZDP2 = DZDP * XSC
      GOTO 190
  180 DZDP = BXCX1(XSC, POWMIN)
      D2ZDP2 = BXCX2(XSC, POWMIN)
C
C        ... ACCUMULATING PARTIAL DERIVATIVES OF RSS  W.R.T. ALL
C        PARAMETERS ...
C
  190 ERR = Y(I) - Z(I)
      DP = DP + ERR * DZDP * WT
      DPDP = DPDP + (B1 * DZDP * DZDP - ERR * D2ZDP2) * WT
      DPDB1 = DPDB1 + (Y(I) - B0 - TWO * ERR) * DZDP * WT
      DPDB0 = DPDB0 + DZDP * WT
  200 CONTINUE
      DP = -TWO * B1 * DP
      DPDP = TWO * B1 * DPDP
      DPDB1 = TWO * DPDB1
      DPDB0 = TWO * B1 * DPDB0
C
C        ... AND FROM THESE OBTAINING THE FULL DERIVATIVES W.R.T. P
C
      IF (IC(1) * IC(2) .NE. 0) GOTO 210
      T = DPDB1 - ZBAR * DPDB0
      DPDP = DPDP - HALF * T * T / SZZ
      IF (IC(1) .EQ. 0 .AND. IC(2) .EQ. 0) DPDP = DPDP - HALF * DPDB0 *
     *  DPDB0 / SW
C
C        FIND MODIFIED NEWTON STEP FOR P, ENSURING THAT NEITHER THE
C        STEP NOR RESULTING POWER IS TOO LARGE
C
```

```
  210 PLIM = STEPMX * FLOAT(ITER)
      IF (DPDP .NE. ZERO) GOTO 220
      PSTEP = SIGN(PLIM, DP)
      GOTO 230
  220 PSTEP = DP / ABS(DPDP)
      IF (PSTEP .LT. -PLIM) PSTEP = -PLIM
      IF (PSTEP .GT. PLIM) PSTEP = PLIM
  230 POWNEW = POWMIN - PSTEP
      IF (ITER .EQ. MAXIT - 1 .AND. ABS(PSTEP) * FLOAT(ITER) .GT. HALF *
     *  ABS(POWNEW) .OR. ABS(POWNEW) .GT. POWMAX) POWNEW = SIGN(POWMAX,
     *  -PSTEP)
      IF (POWMIN * POWNEW .GE. POWMAX * POWMAX) GOTO 1000
  240 CONTINUE
C
C        FINALLY RETURN PARAMETER ESTIMATES
C
 1000 RSS = AMAX1(RSSNEW, ZERO)
      IF (ABS(POWNEW) .GE. POWMAX) IFAULT = 2
      P(3) = POWNEW
      IF (IC(1) .EQ. 0) GOTO 1010
      T = B1 * EXP( - POWNEW * XLOC)
      P(1) = B0 + T
      P(2) = T * POWNEW
      RETURN
 1010 P(2) = B1 * EXP( - XLOC * POWNEW)
      P(1) = B0 + B1 * BXCX0( - XLOC, POWNEW)
      RETURN
      END
C
      REAL FUNCTION BXCX0(X, P)
C
C        ALGORITHM AS 212.1 APPL.STATIST. (1985) VOL.34, NO.2
C
C        EVALUATES  (EXP(X*P) - 1)/P
C
      REAL X, P
      REAL EPS, ONE, XP, TERM, SERIES, AI
C
C        CONSTANTS  -  EPS IS MACHINE DEPENDENT
C
      DATA EPS /1.0E-10/
      DATA ONE /1.0/
C
      XP = X * P
      IF (ABS(XP) .LT. ONE) GOTO 10
      BXCX0 = (EXP(XP) - ONE) / P
      RETURN
   10 TERM = ONE
      SERIES = ONE
      AI = ONE
   20 AI = AI + ONE
      TERM = TERM * XP / AI
      SERIES = SERIES + TERM
      IF (ABS(TERM) .GT. EPS) GOTO 20
      BXCX0 = SERIES * X
      RETURN
      END
C
      REAL FUNCTION BXCX1(X, P)
C
C        ALGORITHM AS 212.2 APPL. STATIST. (1985) VOL.34, NO.2
```

```
C
C       EVALUATES  DERIVATIVE OF (EXP(X*P) - 1)/P
C       WITH RESPECT TO P
C
      REAL X, P
      REAL EPS, HALF, ONE, TWO, XP, TERM, SERIES, AI
C
C       CONSTANTS  -  EPS IS MACHINE DEPENDENT
C
      DATA EPS /1.0E-10/
      DATA HALF /0.5/, ONE /1.0/, TWO /2.0/
C
      XP = X * P
      IF (ABS(XP) .LT. ONE) GOTO 10
      BXCX1 = ((XP - ONE) * EXP(XP) + ONE) / (P * P)
      RETURN
   10 TERM = HALF
      SERIES = HALF
      AI = ONE
   20 TERM = TERM * (AI + ONE) / AI / (AI + TWO) * XP
      SERIES = SERIES + TERM
      AI = AI + ONE
      IF (ABS(TERM) .GT. EPS) GOTO 20
      BXCX1 = SERIES * (X * X)
      RETURN
      END
C
      REAL FUNCTION BXCX2(X, P)
C
C       ALGORITHM AS 212.3  APPL. STATIST. (1985) VOL.34, NO.2
C
C       EVALUATES  SECOND DERIVATIVE OF (EXP(X*P) - 1)/P
C       WITH RESPECT TO P
C
      REAL X, P
      REAL EPS, ONE, TWO, THREE, XP, TERM, SERIES, AI
C
C       CONSTANTS  -  EPS IS MACHINE DEPENDENT
C
      DATA EPS /1.0E-10/
      DATA ONE /1.0/, TWO /2.0/, THREE /3.0/
C
      XP = X * P
      IF (ABS(XP) .LT. ONE) GOTO 10
      BXCX2 = (((XP - TWO) * XP + TWO) * EXP(XP) - TWO) / (P * P * P)
      RETURN
   10 TERM = ONE / THREE
      SERIES = TERM
      AI = ONE
   20 TERM = TERM * (AI + TWO) / AI / (AI + THREE) * XP
      SERIES = SERIES + TERM
      AI = AI + ONE
      IF (ABS(TERM) .GT. EPS) GOTO 20
      BXCX2 = SERIES * (X * X * X)
      RETURN
      END
```

```
      SUBROUTINE RPCORM(NX, N, D, C, U, A, M, ISEED, IFAULT)
C
C        ALGORITHM AS213 APPL. STATIST. (1985) VOL. 34, NO. 2.
C        THIS ALGORITHM RANDOMLY SELECTS A CORRELATION MATRIX FROM THE
C        CLASS OF ALL CORRELATION MATRICES WITH SPECIFIED EIGENVALUES
C
      REAL D(NX), C(NX, NX), U(NX), A(NX, NX)
      INTEGER M(NX, 3)
      DATA EPS /0.00001/
      DATA ZERO, ONE, TWO, FOUR, ZP25, ZP5 /0.0, 1.0, 2.0, 4.0, 0.25,
     *  0.5/
C
      IFAULT = 0
      IF(N .GT. 0) GOTO 40
   10 IFAULT = 1
      RETURN
   20 IFAULT = 2
      RETURN
   30 IFAULT = 3
      RETURN
   40 IF(N .GT. NX) GOTO 10
      TT = ZERO
      DO 60 I = 1, N
      IF(D(I)  .LT. ZERO) GOTO 10
      TT = TT + D(I)
      DO 50 J = 1, N
   50 A(I, J) = ZERO
      A(I, I) = ONE
   60 CONTINUE
      IF (ABS(TT - FLOAT(N)) .GT.  EPS * FLOAT(N)) GOTO 10
C
C        GENERATE AN ORTHOGONAL MATRIX A FROM THE INVARIANT MEASURE ON
C        THE GROUP OF ORTHOGONAL MATRICES.  CONSTRUCT A ONE COLUMN AT
C        A TIME.
C
      DO 300 L = 1, N
      NMLP1 = N - L + 1
      CALL GGNOR(ISEED, NMLP1, U(L))
      IF(L .EQ. N) GOTO 260
      SS = ZERO
      DO 120 I = L, N
  120 SS = SS + U(I) * U(I)
      UL = SQRT(SS)
      IF(UL .EQ. ZERO) GOTO 20
C
C        CALCULATE LENGTH OF U AND STORE IT IN UL.  RANDOMLY CHOSE
C        SIGN OF UL TO PREVENT LOSS OF SIGNIFICANCE IN VL.
C
      UL = SIGN(UL, -U(L))
C
C        PROJECT U ONTO ORTHOGONAL COMPLEMENT OF FIRST L-1 COLUMNS OF
C        A, NORMALIZE AND STORE THE PROJECTION IN L-TH COLUMN OF A.
C        CALCULATE PROJECTION MATRIX FOR ORTHOGONAL COMPLEMENT OF
C        FIRST L COLUMNS OF A AND STORE IT IN LAST N-L COLUMNS OF A.
C
      LP1 = L + 1
      VL = UL - U(L)
      DO 250 I = 1, N
      TT = ZERO
      DO 230 J = L, N
  230 TT = TT + A(I, J) * U(J)
```

```
      HV = TT / UL
      HW = (HV - A(I, L)) / VL
      A(I, L) = HV
      DO 240 J = LP1, N
  240 A(I, J) = A(I, J) - HW * U(J)
  250 CONTINUE
C
C        ASSIGN RANDOM ORIENTATION TO L-TH COLUMN OF A.
C
  260 IF(U(L) .GT. ZERO) GOTO 300
      DO 270 I = 1, N
  270 A(I, L) = -A(I, L)
  300 CONTINUE
C
C        COMPUTE RANDOM COVARIANCE MATRIX C=ADA'
C
      DO 310 J = 1, N
      DO 310 I = J, N
      C(I, J) = ZERO
      DO 305 K = 1, N
  305 C(I, J) = C(I, J) + A(I, K) * D(K) * A(J, K)
  310 CONTINUE
      DO 315 J = 2, N
      JM1 = J - 1
      DO 315 I = 1, JM1
      C(I, J) = C(J, I)
  315 CONTINUE
C
C        RANDOMLY SELECT TWO DIAGONAL ELEMENTS OF C, ONE GREATER THAN
C        ONE AND THE OTHER LESS THAN ONE, THEN CONSTRUCT THE APPRO-
C        PRIATE 2X2 ROTATIONAL MATRIX WHICH SETS ONE DIAGNOL ELEMENT
C        EQUAL TO ONE WHILE MAINTAINING THE STRUCTURE OF EIGENVALUES
C
      NK = 0
      DO 320 I = 1, N
      IF(ABS(C(I, I) - ONE)  .GT.  EPS) GOTO 320
      NK = NK + 1
      M(NK, 1) = I
  320 CONTINUE
      NM = N - NK - 1
      IF(NM .EQ. -1) RETURN
      IF(NM .EQ. 0) GOTO 30
      SUM = ZERO
      DO 400 NN = 1, NM
      IG = 0
      IL = 0
      DO 350 K = 1, N
C
C        M(.,1), M(.,2) AND M(.,3) STORE INDICES OF DIAGONAL ELEMENTS
C        OF C WHICH ARE .EQ. ONE, .GT. ONE, AND .LT. ONE RESPECTIVELY
C
      DO 330 KK = 1, NK
      IF (K .EQ. M(KK, 1)) GOTO 350
  330 CONTINUE
      IF (C(K, K) .GT. (ONE + EPS)) GOTO 340
      IL = IL + 1
      M(IL, 3) = K
      GOTO 350
  340 IG = IG + 1
      M(IG, 2) = K
  350 CONTINUE
```

```
      IF (IL * IG .EQ. 0) GOTO 30
      II = INT(GGUBF(ISEED) * IG) + 1
      JJ = INT(GGUBF(ISEED) * IL) + 1
      I1 = M(II, 2)
      J1 = M(JJ, 3)
      I = MIN0(I1, J1)
      J = MAX0(I1, J1)
      IF (ABS(C(I, I) - ONE) .LE. EPS) GOTO 390
C
C        SOLVE THE QUADRATIC EQUATION AND DETERMINE THE ROTATION ANGLE
C
      CII = C(I, I)
      CIJ = C(I, J)
      CJJ = C(J, J)
      CIIMJJ = CII - CJJ
      CIIPJJ = CII + CJJ - TWO
      CIJIJ = CIJ * CIJ
      CA = CIJIJ + ZP25 * CIIMJJ * CIIMJJ
      CB = ZP5 * CIIPJJ * CIIMJJ
      CC = ZP25 * CIIPJJ * CIIPJJ - CIJIJ
      DT = SQRT(CB * CB - FOUR * CA * CC)
C
C        RANDOMLY SELECT ONE OF THE TWO POSSIBLE SOLUTIONS
C
      IF (GGUBF(ISEED) .GE. ZP5) DT = -DT
      THETA = ARCOS(ZP5 * (-CB + DT) / CA) / TWO
      CS = COS(THETA)
      SN = SIN(THETA)
C
C        CHOOSE THE CORRECT DIRECTION OF ROTATION
C
      IF (ABS(CS * CS * CII + SN * SN * CJJ + TWO * CS * SN * CIJ - ONE)
     *  .GT. EPS) SN = -SN
C
C        COMPUTE C=PCP' USING THE FACT THAT P IS A ROTATIONAL
C        MATRIX WITH THE ROTATION BEING OPERATED BY 2 X 2 MATRIX
C        (CS SN) / (-SN CS).  ONLY ELEMENTS OF I AND J-TH ROWS
C        (HENCE COLUMNS) OF C ARE CHANGED BY THE ROTATION.
C
      DO 360 II = 1, N
      DO 360 JJ = 1, N
  360 A(II, JJ) = C(II, JJ)
      A(I, I) = CS * CS * CII + SN * SN * CJJ + TWO * CS * SN * CIJ
      IF(ABS(A(I, I) - ONE)  .GT. EPS) GOTO 30
      M(NN, 1) = I
      NK = NK + 1
      A(J, J) = CS * CS * CJJ + SN * SN * CII - TWO * CS * SN * CIJ
      A(J, I) = (CS * CS - SN * SN) * CIJ + CS * SN * (CJJ - CII)
      A(I, J) = A(J, I)
      DO 370 L = 1, N
      IF(L .EQ. I .OR. L .EQ. J) GOTO 370
      A(I, L) = CS * C(I, L) + SN * C(J, L)
      A(J, L) = -SN * C(I, L) + CS * C(J, L)
      A(L, I) = A(I, L)
      A(L, J) = A(J, L)
  370 CONTINUE
      DO 380 II = 1, N
      DO 380 JJ = 1, N
  380 C(II, JJ) = A(II, JJ)
  390 SUM = SUM + C(I, I)
  400 CONTINUE
```

```
C
C        SET THE REMAINING DIAGONAL ELEMENT TO BE (N-SUM) AND
C        SEE IF IT IS WITHIN THE SPECIFIED PRECISION LIMIT
C
      C(N, N) = FLOAT(N) - SUM
      IF(ABS(C(N, N) - ONE)  .GT. EPS) GOTO 30
      RETURN
      END
```

```
        subroutine monte(est, conf, nsims, simval, ordval, lsb, sbound,
     # lbb, bbound, alow, ahi, blow, bhi, clow, chi, dlow, dhi,
     # ifault)
        implicit double precision (a-h,o-z)
c
c         Algorithm AS214 Appl. Statistics. (1985) vol. 34, no. 3
c
c         Sets up monte carlo confidence intervals
c         uses function ppnd - algorithm AS 111 (see also AS241)
c
      logical lsb, lbb
      dimension simval(nsims), ordval(nsims)
      data least /5/
      data zero, half, one, two, hun, big /0.0d0,
     #  0.5d0, 1.0d0, 2.0d0, 1.0d2, 1.0d20 /
c
      ifault = 0
      sbnd = sbound
      bbnd = bbound
      if (conf .ge. hun) ifault = 6
      if (conf .le. zero) ifault = 7
      if (ifault .gt. 0) return
c
c        Symmetric mci
c
c        Find mean and variance
c
      v1 = zero
      v2 = zero
      do 10 j = 1, nsims
        fj = float(j)
        v2 = v2 + (fj - one) * (simval(j) - v1) ** 2 / fj
        v1 = (simval(j) + (fj - one) * v1) / fj
 10   continue
      stderr = zero
      if (v2 .gt. zero) stderr = sqrt(v2 / float(nsims - 1))
      alpha = half * (hun - conf) / hun
      z = -ppnd(alpha, ifault)
      if (ifault .eq. 0) goto 20
      ifault = 6
      return
 20   ahi = z * stderr
      alow = est -ahi
      ahi = est + ahi
c
c        Calculate bias adjustment, so that the number of values that
c        must be ordered is known
c
      call biasad(est, nsims, simval, z, limit1, limit2, ifault)
      limitl = alpha * float(nsims + 1) + half
      limitu = (one-alpha) * float(nsims + 1) + half
      if (limitl .lt. least) ifault = 5
      if (limitl .eq. limitu) ifault = 7
      if (ifault .gt. 0) return
      l1 = max0(limit1, 2 * limitl)
      l2 = min0(limit2, 2 * limitu - nsims - 1)
      if (lsb) sbnd = -big
      if (lbb) bbn = big
c
c        Select and order the l1 smallest and the nsims+1-l2 largest
c        values
```

```
c
      call order(simval,ordval, nsims, l1, l2)
      if (sbnd .le. ordval(1)) goto 40
      if (lsb) goto 30
      ifault = 1
      return
 30   sbnd = ordval(1) * two
      if (sbnd .gt. zero) sbnd = -sbnd
 40   if (bbnd .ge. ordval(nsims)) goto 60
      if (lbb) goto 50
      ifault = 2
      return
 50   bbnd = ordval(nsims) * two
      if (bbnd .lt. zero) bbnd = -bbnd
c
c        Equal tails mci ( percentile method )
c
 60   blow = ordval(limitl)
      bhi = ordval(limitu)
c
c        Bias-corrected percentile method
c
      clow = sbnd
      if (limit1 .gt. 0) clow = ordval(limit1)
      chi = bbnd
      if (limit2 .le. nsims) chi = ordval(limit2)
c
c        Minimum lenght mci
c
      call lmin(ordval, nsims, sbnd, bbnd, limitl, limitu, dlow, dhi)
      return
      end
c
c
c
c
c
      subroutine biasad(est, nsims, simval, z, limit1, limit2, ifault)
      implicit double precision (a-h,o-z)
c
c        Finds adjustment required to implement bias corrected
c        percentile method
c
c        Uses functions alnorm and ppnd -
c        algorithms  AS 66 and AS 111
c
      dimension simval(nsims)
      data half, two/0.5d0, 2.0d0/
c
      j = 0
      k = 0
c
c        m = (number of values less than est)+(half the number equal to est),
c        rounded up if not integral
c
      do 30  m = 1, nsims
        if (simval(m) - est) 10, 20, 30
 10     j = j + 1
 20     k = k + 1
 30   continue
      m = (j + k + 1) / 2
```

```
      if (m .eq. 0) ifault = 3
      if (m .eq. nsims) ifault = 4
      if (ifault .gt. 0) return
      zed = two * ppnd(float(m) / float(nsims), ifail)
c
c ifail cannot exceed 0 since m .ge. 1 and m .le. (nsims-1)
c
      fn1 = float(nsims + 1)
      limit1 = fn1 * alnorm(zed - z, .false.) + half
      limit2 = fn1 * alnorm(zed + z, .false.) + half
      return
      end
c
c
c
c
c
c
      subroutine order(simval, ordval, nsims, l1, l2)
      implicit double precision (a-h, o-z)
c
c        Orders the smallest and largest values from simval and
c        puts them in ordval
c
      dimension simval(nsims), ordval(nsims)
c
      ordval(1) = simval(1)
      ordval(nsims) = ordval(1)
      lm1 = 1
      lm2 = nsims
      do 100 j = 2, nsims
        if (simval(j) .lt. ordval(lm1)) goto 10
        if (lm1 .eq. l1) goto 50
        lm1 = lm1 +1
        ordval(lm1) = simval(j)
        goto 50
 10     if (lm1 .lt. l1) lm1 =lm1 +1
        if (lm1 .eq. 2) goto 30
        ll =lm1 -2
        do 20 k = 1, ll
          kk = lm1 - k
          ordval(kk + 1) = ordval(kk)
          if (ordval(kk-1) .le. simval(j)) goto 40
 20     continue
 30     ordval(2) = ordval(1)
        kk=1
 40     ordval(kk) = simval(j)
 50     if (simval(j) .gt. ordval(lm2)) goto 60
        if (lm2 .eq. l2) goto 100
        lm2 = lm2 - 1
        ordval(lm2) = simval(j)
        goto 100
 60     if (lm2 .gt. l2) lm2 = lm2 - 1
        if (lm2 .eq. nsims - 1) goto 80
        ll = nsims - 2
        do 70 k = lm2, ll
          kk = k + 1
          ordval(k) = ordval(kk)
          if (ordval(kk + 1) .ge. simval(j)) goto 90
 70     continue
 80     kk = nsims
```

```
          ordval(kk -1) = ordval(kk)
 90       ordval(kk) = simval(j)
 100    continue
        return
        end
c
c
c
        subroutine lmin(ordval, nsims, sbnd, bbnd, limitl, limitu,
       #dlow,dhi)
        implicit double precision (a-h, o-z)
c
c          Finds the interval with minimum length
c
        dimension ordval(nsims)
        length = limitu - limitl
        numb = nsims - length
        diff = ordval(nsims) - ordval(1)
        l = 0
        do 10 k = 1, numb
          kk = k + length
          if (diff .le. ordval(kk) - ordval(k)) goto 10
          l = k
          diff = ordval(kk) - ordval(k)
 10     continue
        dlow = ordval(l)
        l = l + length
        dhi = ordval(l)
        l = numb + 1
        if (dhi - dlow .le. bbnd - ordval(l)) goto 20
        dlow = ordval(l)
        dhi = bbnd
  20    l = length
        if (dhi - dlow .le. ordval(l) -sbnd) return
        dlow = sbnd
        dhi = ordval (l)
        return
        end
```

```
      SUBROUTINE MLEGEV(X, N, PARA, VCOV, MONIT, IFAULT)

C        ALGORITHM AS215   APPL. STATIST. (1985) VOL. 34, NO. 3
C        Modifications in AS R76 (1989) have been incorporated.
C
C        Additional modifications by J. R. M. Hosking, August 1994:
C        Modify steepest-ascent step so that it is invariant to
C        rescaling the data.  Improves chance of convergence from
C        poor initial values.
C
C        MAXIMUM-LIKELIHOOD ESTIMATION OF GENERALIZED EXTREME-VALUE
C        DISTRIBUTION
C
      DOUBLE PRECISION PARA(3), VCOV(6), X(N)
      DOUBLE PRECISION A, ACCA, ACCG, ACCU, AI, AIGI, AN, D, DA, DAA,
     *  DAG, DELA, DELG, DELU, DG, DGG, DU, DUA, DUG, DUU, E, F, FOLD,
     *  G, GAI, GG, GI, GIPQ, GNORM, H, HALF, HE, HH, ONE, P, PA, PQ,
     *  PQG, PU, Q, QA, QU, R, RA, RATIO, RG, RU, SE, SH, SHE, SHH,
     *  SHHE, SMALL, SRF, STEPA, STEPG, STEPU, SY, SYE, SYHE, SYYE,
     *  TEMP1, TEMP2, U, VLNEG, XMAX, XMIN, Y, YE, Z, ZERO
      CHARACTER*8 ACTI1, ACTI2, ACTI3, ACTI4, ACTI5, ACTI6, ACTI7,
     *  ACTI8, ACTI9
      DATA ACTI1/'  NEWTON'/, ACTI2/'  ST.ASC'/, ACTI3/'  RESETK'/,
     *  ACTI4/'  SR.INF'/, ACTI5/'  SR.LIK'/, ACTI6/'  MAX.SR'/,
     *  ACTI7/'  MAX.EV'/, ACTI8/'  MAX.IT'/, ACTI9/'  CONVGD'/
      DATA ZERO /0.0D0/, HALF /0.5D0/, ONE/1.0D0/
C
C        ADDU,ACCA,ACCG ARE ACCURACY CRITERIA FOR TESTING CONVERGENCE
C        STEPU,STEPA,STEPG ARE MAXIMUM STEPLENGTHS FOR ITERATIONS
C        ACCU,ACCA,STEPU,STEPA ARE SCALED BY CURRENT VALUE OF A WHEN
C        USED IN PROGRAM
C
      DATA ACCU, ACCA, ACCG /3 * 1D-5/, STEPU, STEPA, STEPG /
     *  0.5D0, 0.25D0, 0.2D0/
C
C        MAXIT IS MAX. NO. OF ITERATIONS
C        MAXEV IS MAX. NO. OF EVALUATIONS OF LIKELIHOOD FUNCTION
C        SRF IS STEPLENGTH REDUCTION FACTOR
C        MAXSR IS MAX. NO. OF STEPLENGTH REDUCTIONS PERMITTED PER
C        ITERATION
C
      DATA MAXIT /30/, MAXEV /50/, SRF /0.25D0/, MAXSR /30/
C
C        SMALL IS A SMALL NUMBER, USED TO ADJUST THE SHAPE PARAMETER TO
C        AVOID AN EXACT ZERO VALUE OR BORDERLINE INFEASIBILITY
C        ALNEG IS A LARGE NEGATIVE NUMBER, USED TO INITIALIZE
C        LOG-LIKELIHOOD
C
      DATA SMALL /1.D-3/, VLNEG /-1.D37/
C
C        FIND MIN AND MAX DATA VALUE
C
      DO 10 I = 1, 6
   10 VCOV(I) = ZERO
      IFAULT = 1
      IF (N .LE. 2) GOTO 170
      XMIN = X(1)
      XMAX = X(1)
      DO 20 I = 2, N
        IF (X(I) .LT. XMIN) XMIN = X(I)
        IF (X(I) .GT. XMAX) XMAX = X(I)
```

```
   20 CONTINUE
C
C       INITIALIZATION
C       U IS LOCATION PARAMETER
C       A IS SCALE PARAMETER
C       G IS SHAPE PARAMETER
C
        IF (MONIT .GT. 0) WRITE (6, 6000)
        IFAULT = 0
        NITER = 0
        NEVAL =0
        FOLD = VLNEG
        U = PARA(1)
        A = PARA(2)
        G = PARA(3)
        IF (ABS(G) .LT. SMALL) G = SMALL
        IF (A .LE. ZERO) A = ONE
        AN = DBLE(FLOAT(N))
C
C          CHECK WHETHER ALL DATA POINTS LIE WITHIN THE RANGE OF THE GEV
C          DISTRIBUTION WITH THE INITIAL PARAMETERS - IF NOT, ADJUST THE
C          SHAPE PARAMETER SO AS TO BRING ALL POINTS WITHIN RANGE
C
        IF (G .GT. ZERO) GOTO 30
        IF (XMIN .GE. U) GOTO 40
        Z = A / (XMIN - U)
        IF (G .GT. Z) GOTO 40
        IF (MONIT .GT. 0) WRITE (6, 6010) NITER, NEVAL, U, A, G, ACTI3
        G = Z + SMALL
        IF (G .GE. ZERO) G = HALF * Z
        GOTO 40
   30   IF (XMAX .LE. U) GOTO 40
        Z = A / (XMAX - U)
        IF (G .LT. Z) GOTO 40
        IF(MONIT .GT. 0) WRITE (6, 6010) NITER, NEVAL, U, A, G, ACTI3
        G = Z - SMALL
        IF (G .LE. ZERO) G = HALF * Z
C
C          START OF MAIN LOOP
C
   40   DO 140 NITER = 1, MAXIT
C
        NSR = 0
   50   IF (NEVAL .GE. MAXEV) GOTO 150
        NEVAL = NEVAL + 1
        AI = ONE / A
        GI = ONE /G
        GAI = G * AI
        AIGI = AI * GI
        GG = ONE - G
C
C       ACCUMULATE SUMS OF QUANTITIES OCCURRING IN LIKELIHOOD
C       DERIVATIVES
C
C       IN PRESCOTT AND WALDEN'S NOTATION:
C       Z IS 1 - K * (X(I)-U) / A
C       Y IS THE REDUCED VARIATE - (1/K) * LOG(Z)
C       E IS EXP(-Y)
C       H IS EXP(K*Y)
C
        SY = ZERO
```

```
      SE = ZERO
      SYE = ZERO
      SYYE = ZERO
      SH = ZERO
      SHE = ZERO
      SYHE = ZERO
      SHHE = ZERO
      SHH = ZERO
      DO 60 I = 1, N
        Z = ONE -GAI * (X(I) - U)
        Y = -GI * LOG(Z)
        E = EXP(-Y)
        H = ONE / Z
        YE = Y * E
        HE = H * E
        HH = H * H
        SY = SY + Y
        SE = SE + E
        SYE = SYE + YE
        SYYE = SYYE + Y * YE
        SH = SH + H
        SHE = SHE + HE
        SYHE = SYHE + Y * HE
        SHHE = SHHE + HH * E
        SHH = SHH + HH
   60 CONTINUE
C
C       F IS CURRENT VALUE OF LIKELIHOOD FUNCTIONN
C
      F = -AN * LOG(A) - GG * SY - SE
      IF (F .GT. FOLD) GOTO 90
C
C       LIKELIHOOD HAS NOT INCREASED - REDUCE STEPLENGTH AND TRY AGAIN
C
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI5, F
      IF (NSR .EQ. MAXSR) GOTO 80
   70 NSR = NSR + 1
      U = U - DELU
      A = A - DELA
      G = G - DELG
      DELU = SRF * DELU
      DELA = SRF * DELA
      DELG = SRF * DELG
      U = U + DELU
      A = A + DELA
      G = G + DELG
      IF (A .GT. G * (XMIN - U) .AND. A .GT. G * (XMAX - U) .AND. G .NE.
     *  ZERO) GO TO 50
      IF (MONIT .GT. 0) WRITE (6, 6010) NITER, NEVAL, U, A, G, ACTI4
      IF (NSR .LT. MAXSR) GOTO 70
C
C        MAX. NO. OF STEPLENGTH REDUCTIONS REACHED
C        IF CURRENT ITERATION IS NEWTON-RAPHSON, TRY STEEPEST ASCENT
C        INSTEAD.  IF CURRENT ITERATION IS STEEPEST ASCENT, GIVE UP.
C
   80 U = U - DELU
      A = A - DELA
      G = G - DELG
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI6
      IF (ITYPE .EQ. 1) GOTO 100
      IFAULT = 4
```

```
      GOTO 160
C
C        P,Q,R, ARE AS DEFINED IN FLOOD STUDIES REPORT
C
   90 FOLD = F
      P = AN - SE
      Q = SHE - GG * SH
      R = AN - SY + SYE
      PQ = P + Q
      GIPQ = GI * PQ
C
C        FIRST DERIVATIVES OF LOG-LIKELIHOOD
C
      DU = -AI * Q
      DA = -AIGI * PQ
      DG = -GI * (R - GIPQ)
      IF (MONIT .GT. 0) GNORM = SQRT(DU * DU + DA * DA + DG * DG)
C
C        DERIVATIVES OF P,Q,R
C
      PU = -AI * SHE
      PA = GI * PU + AIGI * SE
      QU = GG * AI * (SHHE + G * SHH)
      RU = AI * (SH - SHE + SYHE)
      RA = GI * RU - AIGI * (AN - SE + SYE)
      RG = GI * (SY - SYE + SYYE - A * RA)
      QA = AI * Q + GI * (PU + QU)
      PQG = GIPQ + A * (RA - GI * (PA + QA))
C
C         MINUS SECOND DERIVATIVE OF LOG-LIKELIHOOD (HESSIAN MATRIX)
C
      DUU = AI * QU
      DUA = AIGI * (PU + QU)
      DAA = -AIGI * (AI * PQ - PA - QA)
      DUG = GI * (RU - GI * (PU + QU))
      DAG = -AIGI * (GIPQ - PQG)
      DGG = GI * (RG - GI * (PQG + R - GIPQ - GIPQ))
C
C         INVERT HESSIAN MATRIX
C
      DO 95 KK = 1, 3
        IF (DUU .LE. ZERO) GO TO 100
        D = ONE / DUU
        TEMP1 = -DUA * D
        IF (KK .GT. 2) TEMP1 = -TEMP1
        TEMP2 = -DUG * D
        IF (KK .GT. 1) TEMP2 = -TEMP2
        DUU = DAA + TEMP1 * DUA
        DUA = DAG + TEMP1 * DUG
        DAA = DGG + TEMP2 * DUG
        DUG = TEMP1
        DAG = TEMP2
        DGG = D
   95 CONTINUE
C
C        CALCULATE STEPLENGTHS
C
      ITYPE = 1
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI1, F,
     *  GNORM
      DELU = DUU * DU + DUA * DA + DUG * DG
```

```
      DELA = DUA * DU + DAA * DA + DAG * DG
      DELG = DUG * DU + DAG * DA + DGG * DG
      RATIO = MAX(ABS(DELU) / (STEPU * A), ABS(DELA) / (STEPA * A),
     *  ABS(DELG) / STEPG)
      IF (RATIO .LT. ONE) GOTO 110
      RATIO = ONE / RATIO
      DELU = DELU * RATIO
      DELA = DELA * RATIO
      DELG = DELG * RATIO
      GOTO 110
C
C        HESSIAN IS NOT POSITIVE DEFINITE - MAKE A LARGE STEP IN THE
C        DIRECTION OF STEEPEST ASCENT
C
  100 ITYPE = 2
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI2, F,
     *  GNORM
C-------------------------------------  NEXT 11 LINES ADDED, AUG.94
      D = ABS(VLNEG)
      TEMP1 = D
      IF (DU .NE. ZERO) TEMP1 = STEPU / (ABS(DU) * A)
      TEMP2 = D
      IF (DA .NE. ZERO) TEMP2 = STEPA / (ABS(DA) * A)
      Z = D
      IF (DG .NE. ZERO) Z = STEPG / ABS(DG)
      RATIO = MIN(TEMP1, TEMP2, Z)
      DELU = RATIO * DU * A * A
      DELA = RATIO * DA * A * A
      DELG = RATIO * DG
C-------------------------------------  NEXT 11 LINES REMOVED, AUG.94
C      D = ABS(VLNEG)
C      TEMP1 = D
C      IF (DU .NE. ZERO) TEMP1 = STEPU * A / ABS(DU)
C      TEMP2 = D
C      IF (DA .NE. ZERO) TEMP2 = STEPA * A / ABS(DA)
C      Z = D
C      IF (DG .NE. ZERO) Z = STEPG / ABS(DG)
C      RATIO = MIN(TEMP1, TEMP2, Z)
C      DELU = RATIO * DU
C      DELA = RATIO * DA
C      DELG = RATIO * DG
C-------------------------------------  END OF DELETED CODE
C
C        ADJUST PARAMETERS
C
  110 U = U + DELU
      A = A + DELA
      G = G + DELG
C
C        TEST FOR FEASIBILITY
C
      IF (A .GT. G * (XMIN - U) .AND. A .GT. G * (XMAX - U) .AND. G .NE.
     *  ZERO) GOTO 130
      DO 120 NSR = 1, MAXSR
        IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI4
        U = U - DELU
        A = A - DELA
        G = G - DELG
        DELU = SRF * DELU
        DELA = SRF * DELA
        DELG = SRF * DELG
```

```
          U = U + DELU
          A = A + DELA
          G = G + DELG
          IF (A .GT. G * (XMIN - U) .AND. A .GT. G * (XMAX - Y) .AND.
     *     G .NE. ZERO) GOTO 140
  120 CONTINUE
      GOTO 80
C
C        TEST FOR CONVERGENCE
C
  130 IF (ABS(DELU) .GT. ACCU * A) GOTO 140
      IF (ABS(DELA) .GT. ACCA * A) GOTO 140
      IF (ABS(DELG) .GT. ACCG) GOTO 140
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, NEVAL, U, A, G, ACTI9
      VCOV(1) = DUU
      VCOV(2) = DUA
      VCOV(3) = DAA
      VCOV(4) = DUG
      VCOV(5) = DAG
      VCOV(6) = DGG
      GOTO 160
C
C        END OF MAIN LOOP
C
  140 CONTINUE
C
C        ITERATIONS NOT CONVERGED - SET FAULT FLAG
C
      IFAULT = 2
      IF (MONIT .GT. 0) WRITE(6, 6010) MAXIT, NEVAL, U, A, G, ACTI8
      GOTO 160
  150 IFAULT = 3
      IF (MONIT .GT. 0) WRITE(6, 6010) NITER, MAXEV, U, A, G, ACTI7
C
C        ITERATION FINISHED -COPY RESULTS INTO ARRAY PARA
C
  160 IF (MONIT .GT. 0) WRITE(6, 6020)
      PARA(1) = U
      PARA(2) = A
      PARA(3) = G
      RETURN
C
  170 DO 180 I = 1, 3
  180 PARA(I) = ZERO
      RETURN
C
 6000 FORMAT(/' MAXIMUM-LIKELIHOOD ESTIMATION OF GENERALIZED EXTREME',
     *  1X, 'VALUE DISTRIBUTION'// ' ITER EVAL', 8X, 'XI', 5X, 'ALPHA',
     *  9X, 'K  ACTION', 6X, 'LOG-L', 7X, 'GNORM')
 6010 FORMAT(1X, I4, I5, 3F10.4, 1X, A, F11.3, 1PD12.2)
 6020 FORMAT(1H0)
      END
```

```
      SUBROUTINE COMBIN(S, IS, R, IR, N, ODMAX, DIMAX, WORKSP, IFAULT)
C
C        ALGORITHM AS 216 APPL. STATIST. (1985) VOL.34, NO.3
C
C        COMBINES THE CHOLESKI FACTORISATIONS OF OBSERVATIONS WITH
C        POSITIVE WEIGHTS (S) AND NEGATIVE WEIGHTS (R) AND MAKES
C        GILL-MURRAY MODIFICATION. ALSO FINDS MAXIMUM DIAGONAL AND
C        OFF-DIAGONAL ELEMENTS OF HESSIAN.
C
      REAL S(IS), R(IR), WORKSP(N), ODMAX, DIMAX
      REAL ZERO, EPS, ELEM, BETA2, SII, RII, CMAX, SIINEW, EI, A, W
      DATA ZERO /0.0/, EPS /1.0E-8/
C
C        CHECK FOR VALID PARAMETERS
C
      IFAULT = 1
      NNP1 = N * (N + 1) / 2
      IF (N .LT. 2 .OR. IS .LT. NNP1 .OR. IR .LT. NNP1) RETURN
      IFAULT = 0
      NPARAM = N - 1
C
C        FIND MAXIMUM DIAG AND OFF-DIAG ELEMENTS
C
      ODMAX = ZERO
      DIMAX = ZERO
      IJ = NNP1 - N
C
C        LOOP FOR I = N-1 DOWN TO 1
C
      I = NPARAM
C
C        LOOP FOR J = I DOWN TO 1
C
   10 J = I
      JJ = IJ
   20 IF (J .LE. 0) GOTO 50
      IF (I .EQ. J) ELEM = S(JJ) - R(JJ)
      IF (I .NE. J) ELEM = S(JJ) * S(IJ) - R(JJ) * R(IJ)
C
C        LOOP FOR K = 1 UP TO (J-1)
C
      IF (J .EQ. 1) GOTO 40
      KK = 0
      IK = I * (I - 1) / 2
      JMINUS = J - 1
      JK = J * JMINUS / 2
      DO 30 K = 1, JMINUS
      KK = KK + K
      IK = IK + 1
      JK = JK + 1
   30 ELEM = ELEM + S(IK) * S(JK) * S(KK) - R(IK) * R(JK) * R(KK)
C
   40 CONTINUE
      IF (I .EQ. J) DIMAX = AMAX1(DIMAX, ABS(ELEM))
      IF (I .NE. J) ODMAX = AMAX1(ODMAX, ABS(ELEM))
      IJ = IJ - 1
      JJ = JJ - J
      J = J - 1
      GOTO 20
C
   50 I = I - 1
```

```
      IF (I .GT. 0) GOTO 10
C
C        COMBINE POSITIVE DEFINITE AND NEGATIVE DEFINITE COMPONENTS
C
      BETA2 = AMAX1(DIMAX, ODMAX / FLOAT(NPARAM), EPS)
      II = 0
      DO 130 I = 1, N
C
C        ALTER ITH ROW OF CHOLESKI S  ....
C
      II = II + I
      SII = S(II)
      RII = R(II)
      IF (I .LT. N) GOTO 60
      S(II) = SII - RII
      RETURN
   60 IF (RII .LE. ZERO) GOTO 130
      DO 70 J = 1, I
   70 WORKSP(J) = ZERO
      IJ = II
      CMAX = ZERO
      IPLUS = I + 1
      DO 80 J = IPLUS, N
      IJ = IJ + J - 1
      WORKSP(J) = S(IJ)
      S(IJ) = S(IJ) * SII - R(IJ) * RII
      IF (J .LT. N .AND. ABS(S(IJ)) .GT. CMAX) CMAX = ABS(S(IJ))
   80 CONTINUE
C
      SIINEW = AMAX1(CMAX * CMAX / BETA2, ABS(SII - RII), EPS)
      S(II) = SIINEW
C
C        .. IF GILL-MURRAY MODIFICATION IS NEEDED UPDATE REST OF S ..
C
      EI = SIINEW - SII * RII
      A = SII / (SII + EI)
      IJ = II
      DO 90 J = IPLUS, N
      IJ = IJ + J - 1
      R(IJ) = R(IJ) - A * WORKSP(J)
   90 S(IJ) = S(IJ) / SIINEW
      IF (EI .LE. ZERO) GOTO 100
      IFAULT = -1
      W = EI * A
      CALL UPDATE(S, WORKSP, W, N, IS)
C
C        .... THEN UPDATE REST OF R
C
  100 W = RII * (SII + EI) / SIINEW
      DO 110 J = 1, I
  110 WORKSP(J) = ZERO
      IJ = II
      DO 120 J = IPLUS, N
      IJ = IJ + J - 1
  120 WORKSP(J) = R(IJ)
      CALL UPDATE(R, WORKSP, W, N, IR)
  130 CONTINUE
C
      END

      SUBROUTINE NUISNC(R, IR, NPARAM, DBDC, IDBDC, DC, DCDC, DIMAX,
```

```
     *  ODMAX, WORKSP, IFAULT)
C
C        ALGORITHM AS 216.1 APPL. STATIST. (1985) VOL.34, NO.3
C
C
C        INCORPORATES DERIVATIVES W.R.T A NUISANCE PARAMETER INTO THE
C        HESSIAN AND GRADIENT WHOSE CHOLESKI FACTORISATION IS IN R AND
C        MAKES GILL-MURRAY MODIFICATION TO GET POSITIVE-DEFINITE HESSIAN
C
      REAL R(IR), DBDC(IDBDC), WORKSP(NPARAM), DC, DCDC, DIMAX, ODMAX
      REAL ZERO, EPS, ONE, BETA2, W, A, TEMP, B
      DATA ZERO /0.0/, EPS /1.0E-7/, ONE /1.0/
C
C        CHECK FOR VALID PARAMETERS
C
      NPPLUS = NPARAM + 1
      NPLESS = NPARAM - 1
      L0 = NPARAM * NPPLUS / 2
      IFAULT = 1
      IF (NPARAM .LE. 1 .OR. L0 + NPPLUS .GT. IR .OR. IDBDC .LT. NPLESS)
     *  RETURN
      IFAULT = 0
C
C        COPY DERIVATIVES INTO R AFTER ELEMENTS OF KDK' AND UPDATE
C        DIMAX AND ODMAX
C
      LI = L0
      DO 10 I = 1, NPLESS
      LI = LI + 1
      ODMAX = AMAX1(ODMAX, ABS(DBDC(I)))
   10 R(LI) = -DBDC(I)
      R(LI + 1) = DC
      DIMAX = AMAX1(DIMAX, ABS(DCDC))
      BETA2 = AMAX1(DIMAX, ODMAX / NPARAM, EPS)
C
C        REPLACE EACH DERIVATIVE BY THE APPROPRIATE ELEMENT FROM THE
C        REPRESENTATION OF MFM'
C
      LI = L0
      IJ = 0
      DO 50 I = 1, NPARAM
      LI = LI + 1
      IF (I .EQ. 1) GOTO 30
      IMINUS = I - 1
      LJ = L0
      DO 20 J = 1, IMINUS
      LJ = LJ + 1
      IJ = IJ + 1
   20 R(LI) = R(LI) - R(IJ) * R(LJ)
   30 IJ = IJ + 1
C
C        ADD TO DIAGONAL OF HESSIAN TO MAKE IT POSITIVE DEFINITE, IF
C        NECESSARY
C
      IF (I .EQ. NPARAM) GOTO 50
      W = AMAX1(R(LI) * R(LI) / BETA2, ABS(R(IJ)), EPS) - R(IJ)
      IF (W .LE. ZERO) GOTO 50
      IFAULT = -1
      DO 40 J = 1, NPARAM
   40 WORKSP(J) = ZERO
      WORKSP(I) = ONE
```

```
         CALL UPDATE(R, WORKSP, W, NPARAM, L0)
      50 CONTINUE
C
C        INTERCHANGE LAST TWO ROWS AND COLS OF MFM'
C
         A = -DCDC
         LI = L0
         KI = L0 - NPARAM
         II = 0
         DO 60 I = 1, NPLESS
         LI = LI + 1
         KI = KI + 1
         II = II + I
         TEMP = R(LI)
         R(LI) = R(KI)
         R(KI) = TEMP / R(II)
      60 A = A - R(KI) * TEMP
C
         KI = KI + 1
         B = R(KI)
         IF (A .LT. EPS) IFAULT = -1
         R(KI) = AMAX1(ABS(A), EPS)
         LI = LI + 1
         TEMP = R(LI)
         R(LI) = TEMP / R(KI)
         R(LI + 1) = B - TEMP * R(LI)
C
         RETURN
         END

         SUBROUTINE STEP(R, IR, PARAMS, OLDLL, NPARAM, RIDGE, WORKSP, IW,
        *  IFAULT)
C
C        ALGORITHM AS 216.2 APPL. STATIST. (1985) VOL.34, NO.3
C
C
C        TAKES A POSITIVE DEFINITE MODIFIED HESSIAN AND USES MARQUARDT'S
C        METHOD, INCREASING DIAGONAL UNTIL A LARGER LOG-LIKELIHOOD IS
C        FOUND
C
         REAL R(IR), PARAMS(NPARAM), WORKSP(IW), OLDLL, RIDGE
         REAL EPS, ZERO, ONE, C, ADDR, DIDIX, TEMP, W, ANEWLL
C
         DATA EPS /1.0E-7/, ZERO /0.0/, ONE /1.0/, C /10.0/
         DATA MAXIT /10/
C
C        CHECK FOR VALID PARAMETERS
C
         IFAULT = 1
         NPPLUS = NPARAM + 1
         NRUSED = NPPLUS * (NPPLUS + 1) / 2
         IF (NPARAM .LE. 0 .OR. IR .LT. NRUSED .OR. IW .LT. NPPLUS) RETURN
         IFAULT = 0
C
         ITER = 1
         IF (RIDGE .LE. ZERO) GOTO 50
         IF (RIDGE .GE. EPS * C) RIDGE = RIDGE / C
         ADDR = RIDGE
C
C        FIND DIAGONAL ELEMENTS OF HESSIAN ....
C
```

```
   10 IJ = 0
      DO 40 I = 1, NPARAM
      DIDIX = ZERO
      JJ = 0
      DO 20 J = 1, I
      JJ = JJ + J
      IJ = IJ + 1
      TEMP = ONE
      IF (I .NE. J) TEMP = R(IJ)
   20 DIDIX = DIDIX + TEMP * TEMP * R(JJ)
      IF (R(JJ) .LT. ZERO) GOTO 90
C
C        .... AND ADD TO THEM
C
      W = ADDR * DIDIX
      DO 30 J = 1, NPPLUS
   30 WORKSP(J) = ZERO
      WORKSP(I) = ONE
      CALL UPDATE(R, WORKSP, W, NPPLUS, IR)
   40 CONTINUE
C
C        IF LIKELIHOOD IS NOT INCREASED, ADD BIGGER RIDGE TO HESSIAN
C
   50 CALL BSUB(R, IR, NPPLUS, WORKSP, IW, IFAIL)
      DO 60 I = 1, NPARAM
   60 WORKSP(I) = WORKSP(I) + PARAMS(I)
      ANEWLL = ALLIKE(WORKSP, NPARAM)
      IF (ANEWLL .GE. OLDLL) GOTO 70
      ADDR = RIDGE * (C + ONE) / (ONE + RIDGE)
      IF (RIDGE .GE. ZERO) RIDGE = RIDGE * C
      ITER = ITER + 1
      IFAULT = -1
      IF (ITER .LE. MAXIT .AND. RIDGE .GT. ZERO) GOTO 10
C
C        TOO MANY ITERATIONS
C
      IFAULT = 2
      RETURN
C
C        RETURN IMPROVED PARAMETERS
C
   70 OLDLL = ANEWLL
      DO 80 I = 1, NPARAM
   80 PARAMS(I) = WORKSP(I)
      RETURN
C
C        HESSIAN NOT POSITIVE SEMI-DEFINITE ON INPUT
C
   90 IFAULT = 3
      RETURN
      END
```

```
      SUBROUTINE DIPTST(X, N, DIP, XL, XU, IFAULT, GCM, LCM, MN, MJ)
C
C     ALGORITHM AS 217 APPL. STATIST. (1985) VOL.34, NO.3
C
C     Does the dip calculation for an ordered vector X using the
C     greatest convex minorant and the least concave majorant, skipping
C     through the data using the change points of these distributions.
C     It returns the dip statistic 'DIP' and the modal interval
C     (XL, XU).
C
      REAL X(N)
      INTEGER MN(N), MJ(N), LCM(N), GCM(N), HIGH
      REAL ZERO, HALF, ONE
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/
C
      IFAULT = 1
      IF (N .LE. 0) RETURN
      IFAULT = 0
C
C     Check if N = 1
C
      IF (N .EQ. 1) GO TO 4
C
C     Check that X is sorted
C
      IFAULT = 2
      DO 3 K = 2, N
        IF (X(K) .LT. X(K-1)) RETURN
    3 CONTINUE
      IFAULT = 0
C
C     Check for all values of X identical,
C     and for 1 < N < 4.
C
      IF (X(N) .GT. X(1) .AND. N .GE. 4) GO TO 5
    4 XL = X(1)
      XU = X(N)
      DIP = ZERO
      RETURN
C
C     LOW contains the index of the current estimate of the lower end
C     of the modal interval, HIGH contains the index for the upper end.
C
    5 FN = FLOAT(N)
      LOW = 1
      HIGH = N
      DIP = ONE / FN
      XL = X(LOW)
      XU = X(HIGH)
C
C     Establish the indices over which combination is necessary for the
C     convex minorant fit.
C
      MN(1) = 1
      DO 28 J = 2, N
        MN(J) = J - 1
   25   MNJ = MN(J)
        MNMNJ = MN(MNJ)
        A = FLOAT(MNJ - MNMNJ)
        B = FLOAT(J - MNJ)
        IF (MNJ .EQ. 1 .OR. (X(J) - X(MNJ))*A .LT. (X(MNJ) - X(MNMNJ))
```

```
     +          *B) GO TO 28
           MN(J) = MNMNJ
           GO TO 25
   28 CONTINUE
C
C     Establish the indices over which combination is necessary for the
C     concave majorant fit.
C
      MJ(N) = N
      NA = N - 1
      DO 34 JK = 1, NA
        K = N - JK
        MJ(K) = K + 1
   32   MJK = MJ(K)
        MJMJK = MJ(MJK)
        A = FLOAT(MJK - MJMJK)
        B = FLOAT(K - MJK)
        IF (MJK .EQ. N .OR. (X(K) - X(MJK))*A .LT. (X(MJK) - X(MJMJK))
     +          *B) GO TO 34
        MJ(K) = MJMJK
        GO TO 32
   34 CONTINUE
C
C     Start the cycling.
C     Collect the change points for the GCM from HIGH to LOW.
C
   40 IC = 1
      GCM(1) = HIGH
   42 IGCM1 = GCM(IC)
      IC = IC + 1
      GCM(IC) = MN(IGCM1)
      IF (GCM(IC) .GT. LOW) GO TO 42
      ICX = IC
C
C     Collect the change points for the LCM from LOW to HIGH.
C
      IC = 1
      LCM(1) = LOW
   44 LCM1 = LCM(IC)
      IC = IC + 1
      LCM(IC) = MJ(LCM1)
      IF (LCM(IC) .LT. HIGH) GO TO 44
      ICV = IC
C
C     ICX, IX, IG are counters for the convex minorant,
C     ICV, IV, IH are counters for the concave majorant.
C
      IG = ICX
      IH = ICV
C
C     Find the largest distance greater than 'DIP' between the GCM and
C     the LCM from LOW to HIGH.
C
      IX = ICX - 1
      IV = 2
      D = ZERO
      IF (ICX .NE. 2 .OR. ICV .NE. 2) GO TO 50
      D = ONE / FN
      GO TO 65
   50 IGCMX = GCM(IX)
      LCMIV = LCM(IV)
```

```
      IF (IGCMX .GT. LCMIV) GO TO 55
C
C     If the next point of either the GCM or LCM is from the LCM,
C     calculate the distance here.
C
      LCMIV1 = LCM(IV - 1)
      A = FLOAT(LCMIV - LCMIV1)
      B = FLOAT(IGCMX - LCMIV1 - 1)
      DX = (X(IGCMX) - X(LCMIV1)*A) / (FN*(X(LCMIV) - X(LCMIV1)))
     +          - B / FN
      IX = IX - 1
      IF (DX .LT. D) GO TO 60
      D = DX
      IG = IX + 1
      IH = IV
      GO TO 60
C
C     If the next point of either the GCM or LCM is from the GCM,
C     calculate the distance here.
C
   55 LCMIV = LCM(IV)
      IGCM = GCM(IX)
      IGCM1 = GCM(IX + 1)
      A = FLOAT(LCMIV - IGCM1 + 1)
      B = FLOAT(IGCM - IGCM1)
      DX = A / FN - ((X(LCMIV) - X(IGCM1))*B) / (FN * (X(IGCM)
     +          - X(IGCM1)))
      IV = IV + 1
      IF (DX .LT. D) GO TO 60
      D = DX
      IG = IX + 1
      IH = IV - 1
   60 IF (IX .LT. 1) IX = 1
      IF (IV .GT. ICV) IV = ICV
      IF (GCM(IX) .NE. LCM(IV)) GO TO 50
   65 IF (D .LT. DIP) GO TO 100
C
C     Calculate the DIPs for the current LOW and HIGH.
C
C     The DIP for the convex minorant.
C
      DL = ZERO
      IF (IG .EQ. ICX) GO TO 80
      ICXA = ICX - 1
      DO 76 J = IG, ICXA
        TEMP = ONE / FN
        JB = GCM(J + 1)
        JE = GCM(J)
        IF (JE - JB .LE. 1) GO TO 74
        IF (X(JE) .EQ. X(JB)) GO TO 74
        A = FLOAT(JE - JB)
        CONST = A / (FN * (X(JE) - X(JB)))
        DO 72 JR = JB, JE
          B = FLOAT(JR - JB + 1)
          T = B / FN - (X(JR) - X(JB))*CONST
          IF (T .GT. TEMP) TEMP = T
   72   CONTINUE
   74   IF (DL .LT. TEMP) DL = TEMP
   76 CONTINUE
C
C     The DIP for the concave majorant.
```

```
C
   80 DU = ZERO
      IF (IH .EQ. ICV) GO TO 90
      ICVA = ICV - 1
      DO 88 K = IH, ICVA
        TEMP = ONE / FN
        KB = LCM(K)
        KE = LCM(K + 1)
        IF (KE - KB .LE. 1) GO TO 86
        IF (X(KE) .EQ. X(KB)) GO TO 86
        A = FLOAT(KE - KB)
        CONST = A / (FN * (X(KE) - X(KB)))
        DO 84 KR = KB, KE
          B = FLOAT(KR - KB - 1)
          T = (X(KR) - X(KB))*CONST - B / FN
          IF (T .GT. TEMP) TEMP = T
   84   CONTINUE
   86   IF (DU .LT. TEMP) DU = TEMP
   88 CONTINUE
C
C     Determine the current maximum.
C
   90 DIPNEW = DL
      IF (DU .GT. DL) DIPNEW = DU
      IF (DIP .LT. DIPNEW) DIP = DIPNEW
      LOW = GCM(IG)
      HIGH = LCM(IH)
C
C     Recycle
C
      GO TO 40
C
  100 DIP = HALF * DIP
      XL = X(LOW)
      XU = X(HIGH)
C
      RETURN
      END
```

```
C This file contains:
C 1. Single-precision version of AS 218 as supplied by the RSS Algorithms
C    Editor
C 2. Double-precision version supplied by Luis Escobar (author)
C 3. A demo program (double precision) supplied by the author
C
C
      SUBROUTINE SEVINF(ITYPE, ZL, ZR, F11, F12, F22, IFAULT)
C
C        ALGORITHM AS218  APPL. STATIST. (1986) VOL. 35, NO. 1
C
C        FISHER INFORMATION MATRIX ELEMENTS FOR TIME (TYPE I) OR
C        FAILURE (TYPE II) CENSORED SMALLEST EXTREME VALUE DATA
C
      REAL BG, DEN, ETA, F11, F12, F22, G, ONE, THET0L, THET0R, THET1L,
     *  THET1R, THET2L, THET2R, X, ZERO, ZL, ZMAX, ZMIN, ZR, ZU
C
      DATA ONE /1.0E0/, ZERO /0.0E0/, ZMAX /3.6E0/, ZMIN /-34.0E0/
C
C        FUNCTION STATEMENTS - PDF (G(X)) AND CDF (BG(X)) FOR A
C        STANDARDIZED SMALLEST EXTREME VALUE RANDOM VARIABLE
C
      G(X) = EXP(X - EXP(X))
      BG(X) = ONE - EXP(-EXP(X))
C
C        CHECK FOR ILLEGAL ITYPE
C
      IF (ITYPE .LT. 1 .OR. ITYPE .GT. 4) GOTO 6
C
C        IDENTIFY THE CENSORING TYPE
C        ITYPE = 1, 2, 3, 4 CORRESPOND TO COMPLETE, RIGHT,
C        LEFT, AND LEFT AND RIGHT CENSORED DATA, RESPECTIVELY
C
      GOTO (1, 2, 3, 4), ITYPE
C
C        COMPLETE DATA - FISHER MATRIX ELEMENTS
C
    1 CALL INTEGR(ZMAX, F11, F12, F22, IFAULT)
      RETURN
C
C        RIGHT CENSORED DATA - FISHER MATRIX ELEMENTS
C
    2 CALL INTEGR(ZR, F11, F12, F22, IFAULT)
      RETURN
C
C        LEFT CENSORED DATA
C        SET THE RIGHT CENSORING BOUND (ZU) EQUAL TO ZMAX
C
    3 ZU = ZMAX
      GOTO 5
C
C        LEFT AND RIGHT CENSORED DATA - CHECK THAT THE LEFT
C        CENSORING BOUND IS NOT GREATER THAN THE RIGHT BOUND
C
    4 IF (ZL .GT. ZR) GOTO 7
      ZU = ZR
C
C        LEFT OR LEFT AND RIGHT CENSORED DATA - FISHER MATRIX ELEMENTS
C        ETA IS THE SECOND TERM IN THE EQUATION FOR F11
C
    5 ETA = ZERO
```

```
      DEN = ZERO
      IF (ZL .GT. ZMIN .AND. ZL .LT. ZMAX) DEN = BG(ZL) * EXP(-EXP(ZL))
      IF (DEN .GT. ZERO) ETA = G(ZL) ** 2 / DEN
      CALL INTEGR(ZL, THET0L, THET1L, THET2L, IFAULT)
      IF (IFAULT .EQ. 1) RETURN
      CALL INTEGR(ZU, THET0R, THET1R, THET2R, IFAULT)
      IF (IFAULT .EQ. 1) RETURN
      F11 = THET0R - THET0L + ETA
      F12 = THET1R - THET1L + ZL * ETA
      F22 = THET2R - THET2L + ZL * ZL * ETA
      RETURN
C
C        SET IFAULT
C
    6 IFAULT = 2
      RETURN
    7 IFAULT = 3
      RETURN
      END
C
      SUBROUTINE INTEGR(Z, THETA0, THETA1, THETA2, IFAULT)
C
C        COMPUTES THE INTEGRALS FROM (-INFINITY) TO Z OF THE FOLLOWING
C        FUNCTIONS: G(X)=EXP(X-EXP(X)), G1(X)=(1+X)*G(X), AND
C        G2(X)=(1+X)*G1(X).  G(X) HAS A CLOSED FORM INTEGRAL.
C        G1 AND G2 ARE INTEGRATED BY USING POWER SERIES EXPANSIONS
C        WHEN Z.LE.1, AND BY POWER SERIES EXPANSIONS THROUGH Z=1
C        PLUS A GAUSSIAN QUADRATURE FROM 1 TO Z WHEN Z.GT.1
C
      REAL ASYMP1, ASYMP2, EZ, FACTOR, G1, G1XL, G1XR, G2XL, G2XR, HALF,
     *  ONE, S1, S2, THETA0, THETA1, THETA2, THET11, THET21, TOL, TWO,
     *  X, X1, X2, X3, X4, X5, Z, ZERO, ZMAX, ZMIN
C
      REAL P(8), W(8)
C
C        P AND W ARE CONSTANTS USED IN THE GAUSSIAN QUADRATURES
C
      DATA P(1) /0.4947004674958250E0/, P(2) /0.4722875115366163E0/,
     *  P(3) /0.4328156011939159E0/, P(4) /0.3777022041775015E0/, P(5)
     *  /0.3089381222013219E0/, P(6) /0.2290083888286137E0/, P(7)
     *  /0.1408017753896295E0/, P(8) /0.4750625491881872E-1/
      DATA W(1) /0.1357622970587705E-1/, W(2) /0.3112676196932395E-1/,
     *  W(3) /0.4757925584124639E-1/, W(4) /0.6231448562776694E-1/,
     *  W(5) /0.7479799440828837E-1/, W(6) /0.8457825969750127E-1/, W(7)
     *  /0.9130170752246179E-1/, W(8) /0.9472530522753425E-1/
C
C        ASYMP1, ASYMP2, THET11, AND THET21 ARE THE INTEGRALS
C        FOR G1 AND G2 WHEN Z=+INFINITY AND Z=1 ,RESPECTIVELY
C
      DATA ASYMP1 /0.4227843350984671E0/, ASYMP2 /0.1823680660852879E1/,
     *  HALF /0.5E0/, JMAX /25/, ONE /1.0E0/, TOL /1.0E-11/, THET11 /
     *  0.2720757938345342E0/, THET21 /0.1475933122158450E1/, TWO /
     *  2.0E0/, ZERO /0.0E0/, ZMAX /3.6E0/, ZMIN /-34.0E0/
C
C        FUNCTION REQUIRED IN THE GAUSSIAN QUADRATURE
C
      G1(X) = (ONE + X) * EXP(X - EXP(X))
C
C        CHECK FOR EXTREME VALUES
C
      IFAULT = 0
```

```
      IF (Z .GE. ZMAX) GOTO 5
      IF (Z .LE. ZMIN) GOTO 6
C
C        COMPUTATION OF THETA0 - INTEGRAL OF G(X)=EXP(X-EXP(X))
C
      EZ = EXP(Z)
      THETA0 = ONE - EXP(-EZ)
C
C        SELECT INTEGRATION BY POWER SERIES EXPANSIONS OR
C        BY POWER SERIES EXPANSIONS AND GAUSSIAN QUADRATURE
C
      IF (Z .GE. ONE) GOTO 3
C
C        Z IS LESS THAN 1 - INTEGRATION BY POWER SERIES EXPANSIONS
C
C        COMPUTATION OF THETA1 AND THETA2
C
      FACTOR = -ONE
      S1 = ZERO
      S2 = ZERO
      DO 1 J = 1, JMAX
C
C        TERMS TO EVALUATE THETA1(Z) - INTEGRAL OF G1(X)=(1+X)G(X)
C
      X5 = FLOAT(J)
      FACTOR = -FACTOR * EZ / X5
      X1 = Z - ONE / X5
      X3 = FACTOR * X1
      S1 = S1 + X3
C
C        TERMS TO EVALUATE THETA2(Z) - INTEGRAL OF G2(X)=(1+X)G1(X)
C
      X2 = X1 * X1 + ONE / (X5 * X5)
      X4 = FACTOR * X2
      S2 = S2 + X4
C
C        TESTS FOR CONVERGENCE OF THE SERIES -  THE SUMMATIONS STOP WHEN
C        THE ABSOLUTE VALUE OF THE LAST ADDED TERM IS SMALLER THAN TOL
C        FOR BOTH SERIES - A FAULT IS DECLARED IF CONVERGENCE IS NOT
C        REACHED IN A MAXIMUM OF JMAX TERMS
C
      X5 = AMAX1(ABS(X3), ABS(X4))
      IF (X5 .LT. TOL) GOTO 2
    1 CONTINUE
      IFAULT = 1
      RETURN
C
C        ADD TERMS TO OBTAIN INTEGRALS
C
    2 THETA1 = THETA0 + S1
      THETA2 = TWO * THETA1 - THETA0 + S2
      RETURN
C
C        Z IS BETWEEN 1 AND 3.6.
C        THET11 AND THET12 CONTAIN THE INTEGRALS OF G1(X) AND G2(X)
C        OVER (-INFINITY,1) AND THEY ARE DEFINED IN THE DATA STATEMENTS
C
C        GAUSSIAN QUADRATURE TO INTEGRATE G1(X) AND G2(X) OVER (1,Z)
C
    3 S1 = ZERO
      S2 = ZERO
```

```
      X1 = HALF * (Z + ONE)
      X2 = Z - ONE
      DO 4 IQUAD = 1, 8
      X3 = X1 - P(IQUAD) * X2
      X4 = X1 + P(IQUAD) * X2
      G1XL = G1(X3)
      G1XR = G1(X4)
      G2XL = (ONE + X3) * G1XL
      G2XR = (ONE + X4) * G1XR
      S1 = S1 + W(IQUAD) * (G1XL + G1XR)
      S2 = S2 + W(IQUAD) * (G2XL + G2XR)
    4 CONTINUE
      S1 = S1 * X2
      S2 = S2 * X2
C
C        ADD TERMS TO OBTAIN INTEGRALS
C
      THETA1 = S1 + THET11
      THETA2 = S2 + THET21
      RETURN
C
C        ASYMPTOTIC VALUES.  Z IS GREATER THAN OR EQUAL TO 3.6
C
    5 THETA0 = ONE
      THETA1 = ASYMP1
      THETA2 = ASYMP2
      RETURN
C
C        ASYMPTOTIC VALUES.  Z IS SMALLER THAN OR EQUAL TO -34
C
    6 THETA0 = ZERO
      THETA1 = ZERO
      THETA2 = ZERO
      RETURN
      END
C
C------------------------- Double precision version ------------------
C
      SUBROUTINE SEVINF(ITYPE,ZL,ZR,F11,F12,F22,IFAULT)
C
C        ALGORITHM AS218  APPL. STATIST. (1986) VOL. 35, NO. 1
C
C       FISHER INFORMATION MATRIX ELEMENTS FOR TIME (TYPE I) OR
C       FAILURE (TYPE II) CENSORED SMALLEST EXTREME VALUE DATA
C
      DOUBLE PRECISION BG, DEN, ETA, F11, F12, F22, G, ONE, THET0L,
     * THET0R, THET1L, THET1R, THET2L, THET2R, X, ZERO,
     * ZL, ZMAX, ZMIN, ZR, ZU
C
      DATA  ONE/1.0D0/, ZERO/0.0D0/, ZMAX/3.6D0/, ZMIN/-34.0D0/
C
C       FUNCTION STATEMENTS - PDF (G(X)) AND CDF (BG(X)) FOR A
C       STANDARDIZED SMALLEST EXTREME VALUE RANDOM VARIABLE
C
      G(X) = EXP(X-EXP(X))
      BG(X) = ONE-EXP(-EXP(X))
C
C       CHECK FOR ILLEGAL ITYPE
C
      IF (ITYPE.LT.1 .OR. ITYPE.GT.4) GOTO 6
C
```

```
C       IDENTIFY THE CENSORING TYPE
C       ITYPE = 1, 2, 3, 4 CORRESPOND TO COMPLETE, RIGHT,
C       LEFT, AND LEFT AND RIGHT CENSORED DATA, RESPECTIVELY
C
        GOTO (1,2,3,4), ITYPE
C
C       COMPLETE DATA - FISHER MATRIX ELEMENTS
C
    1 CALL INTEGR(ZMAX,F11,F12,F22,IFAULT)
      RETURN
C
C       RIGHT CENSORED DATA - FISHER MATRIX ELEMENTS
C
    2 CALL INTEGR(ZR,F11,F12,F22,IFAULT)
      RETURN
C
C       LEFT CENSORED DATA
C       SET THE RIGHT CENSORING BOUND (ZU) EQUAL TO ZMAX
C
    3 ZU = ZMAX
      GOTO 5
C
C       LEFT AND RIGHT CENSORED DATA - CHECK THAT THE LEFT
C       CENSORING BOUND IS NOT GREATER THAN THE RIGHT BOUND
C
    4 IF (ZL.GT.ZR) GOTO 7
      ZU = ZR
C
C       LEFT OR LEFT AND RIGHT CENSORED DATA - FISHER MATRIX ELEMENTS
C       ETA IS THE SECOND TERM IN THE EQUATION FOR F11
C
    5 ETA = ZERO
      DEN = ZERO
      IF (ZL.GT.ZMIN.AND.ZL.LT.ZMAX) DEN = BG(ZL)*DEXP(-DEXP(ZL))
      IF (DEN.GT.ZERO) ETA = G(ZL)**2/DEN
      CALL INTEGR(ZL,THET0L,THET1L,THET2L,IFAULT)
      IF (IFAULT.EQ.1) RETURN
      CALL INTEGR(ZU,THET0R,THET1R,THET2R,IFAULT)
      IF (IFAULT.EQ.1) RETURN
      F11 = THET0R-THET0L+ETA
      F12 = THET1R-THET1L+ZL*ETA
      F22 = THET2R-THET2L+ZL*ZL*ETA
      RETURN
C
C       SET IFAULT
C
    6 IFAULT = 2
      RETURN
    7 IFAULT = 3
      RETURN
      END

      SUBROUTINE INTEGR(Z,THETA0,THETA1,THETA2,IFAULT)
C
C       COMPUTES THE INTEGRALS FROM (-INFINITY) TO Z OF THE FOLLOWING
C       FUNCTIONS: G(X) = EXP(X-EXP(X)), G1(X) = (1+X)*G(X), AND
C       G2(X) = (1+X)*G1(X).  G(X) HAS A CLOSED FORM INTEGRAL.
C       G1 AND G2 ARE INTEGRATED BY USING POWER SERIES EXPANSIONS
C       WHEN Z.LE.1, AND BY POWER SERIES EXPANSIONS THROUGH Z = 1
C       PLUS A GAUSSIAN QUADRATURE FROM 1 TO Z WHEN Z.GT.1
C
```

```
      DOUBLE PRECISION ASYMP1, ASYMP2, EZ, FACTOR, G1, G1XL, G1XR,
     * G2XL, G2XR, HALF, ONE, S1, S2, THETA0, THETA1,
     * THETA2, THET11, THET21, TOL, TWO, X, X1, X2, X3,
     * X4, X5, Z, ZERO, ZMAX, ZMIN
C
      DOUBLE PRECISION P(8), W(8)
C
C       P AND W ARE CONSTANTS USED IN THE GAUSSIAN QUADRATURES
C
      DATA P(1)/0.4947004674958250D0 /, P(2)/0.4722875115366163D0 /,
     *     P(3)/0.4328156011939159D0 /, P(4)/0.3777022041775015D0 /,
     *     P(5)/0.3089381222013219D0 /, P(6)/0.2290083888286137D0 /,
     *     P(7)/0.1408017753896295D0 /, P(8)/0.4750625491881872D-1/
      DATA W(1)/0.1357622970587705D-1/, W(2)/0.3112676196932395D-1/,
     *     W(3)/0.4757925584124639D-1/, W(4)/0.6231448562776694D-1/,
     *     W(5)/0.7479799440828837D-1/, W(6)/0.8457825969750127D-1/,
     *     W(7)/0.9130170752246179D-1/, W(8)/0.9472530522753425D-1/
C
C       ASYMP1, ASYMP2, THET11, AND THET21 ARE THE INTEGRALS
C       FOR G1 AND G2 WHEN Z = +INFINITY AND Z = 1 ,RESPECTIVELY
C
      DATA ASYMP1/0.4227843350984671D0/, ASYMP2/0.1823680660852879D1/,
     *     HALF/0.5D0/, JMAX/25/, ONE/1.0D0/, TOL/1.0D-11/,
     *     THET11/0.2720757938345342D0/, THET21/0.1475933122158450D1/,
     *     TWO/2.0D0/, ZERO/0.0D0/, ZMAX/3.6D0/, ZMIN/-34.0D0/
C
C       FUNCTION REQUIRED IN THE GAUSSIAN QUADRATURE
C
      G1(X) = (ONE+X)*EXP(X-EXP(X))
C
C       CHECK FOR EXTREME VALUES
C
      IFAULT = 0
      IF(Z.GE.ZMAX) GOTO 5
      IF(Z.LE.ZMIN) GOTO 6
C
C       COMPUTATION OF THETA0 - INTEGRAL OF G(X) = EXP(X-EXP(X))
C
      EZ = DEXP(Z)
      THETA0 = ONE-DEXP(-EZ)
C
C       SELECT INTEGRATION BY POWER SERIES EXPANSIONS OR
C       BY POWER SERIES EXPANSIONS AND GAUSSIAN QUADRATURE
C
      IF(Z.GE.ONE) GOTO 3
C
C       Z IS LESS THAN 1 - INTEGRATION BY POWER SERIES EXPANSIONS
C
C       COMPUTATION OF THETA1 AND THETA2
C
      FACTOR = -ONE
      S1 = ZERO
      S2 = ZERO
      DO 1 J = 1,JMAX
C
C       TERMS TO EVALUATE THETA1(Z) - INTEGRAL OF G1(X) = (1+X)G(X)
C
      X5 = FLOAT(J)
      FACTOR = -FACTOR*EZ/X5
      X1 = Z-ONE/X5
      X3 = FACTOR*X1
```

```
      S1 = S1+X3
C
C       TERMS TO EVALUATE THETA2(Z) - INTEGRAL OF G2(X) = (1+X)G1(X)
C
      X2 = X1*X1+ONE/(X5*X5)
      X4 = FACTOR*X2
      S2 = S2+X4
C
C       TESTS FOR CONVERGENCE OF THE SERIES -  THE SUMMATIONS STOP WHEN
C       THE ABSOLUTE VALUE OF THE LAST ADDED TERM IS SMALLER THAN TOL
C       FOR BOTH SERIES - A FAULT IS DECLARED IF CONVERGENCE IS NOT
C       REACHED IN A MAXIMUM OF JMAX TERMS
C
      X5 = MAX1(ABS(X3),ABS(X4))
      IF (X5.LT.TOL) GOTO 2
    1 CONTINUE
      IFAULT = 1
      RETURN
C
C       ADD TERMS TO OBTAIN INTEGRALS
C
    2 THETA1 = THETA0+S1
      THETA2 = TWO*THETA1-THETA0+S2
      RETURN
C
C       Z IS BETWEEN 1 AND 3.6.
C       THET11 AND THET12 CONTAIN THE INTEGRALS OF G1(X) AND G2(X)
C       OVER (-INFINITY,1) AND THEY ARE DEFINED IN THE DATA STATEMENTS
C
C       GAUSSIAN QUADRATURE TO INTEGRATE G1(X) AND G2(X) OVER (1,Z)
C
    3 S1 = ZERO
      S2 = ZERO
      X1 = HALF*(Z+ONE)
      X2 = Z-ONE
      DO 4 IQUAD = 1,8
      X3 = X1-P(IQUAD)*X2
      X4 = X1+P(IQUAD)*X2
      G1XL = G1(X3)
      G1XR = G1(X4)
      G2XL = (ONE+X3)*G1XL
      G2XR = (ONE+X4)*G1XR
      S1 = S1+W(IQUAD)*(G1XL+G1XR)
      S2 = S2+W(IQUAD)*(G2XL+G2XR)
    4 CONTINUE
      S1 = S1*X2
      S2 = S2*X2
C
C       ADD TERMS TO OBTAIN INTEGRALS
C
      THETA1 = S1+THET11
      THETA2 = S2+THET21
      RETURN
C
C       ASYMPTOTIC VALUES.  Z IS GREATER THAN OR EQUAL TO 3.6
C
    5 THETA0 = ONE
      THETA1 = ASYMP1
      THETA2 = ASYMP2
      RETURN
C
```

```
C       ASYMPTOTIC VALUES.  Z IS SMALLER THAN OR EQUAL TO -34
C
    6 THETA0 = ZERO
      THETA1 = ZERO
      THETA2 = ZERO
      RETURN
      END
C
C------------------------ Demo program (double precision) -------------
C
C
C         DRIVER PROGRAM FOR SEVINF
C
      PROGRAM DEM218
      DOUBLE PRECISION
     *     COV, DET, F11, F12, F22, ONE, Q1, Q2,
     *     SEVEN, TENTH, VAR1, VAR2, ZERO, ZL, ZR
C
      DATA ONE/1.0D0/, SEVEN/7.0D0/, TENTH/0.1D0/, ZERO/0.0D0/
      DATA IUNIT0/10/, IUNIT1/15/
C
      OPEN(IUNIT0, FILE='HARTER.OUT', STATUS='NEW')
      OPEN(IUNIT1, FILE='MEEKER.OUT', STATUS='NEW')
C
1     FORMAT(1H1, 2X, 'Q1', 4X, 'Q2', 8X, 'VAR(U)', 8X, 'VAR(S)', 8X,
     * 'COV(U,S)', 2X, 'IFAULT')
2     FORMAT(1H , 2F6.2, 3(3X, F11.6), I3)
3     FORMAT(1H1, ' ZETA', 4X, 'VAR(U)', 9X, 'VAR(S)', 9X,
     * 'COV(U,S)', 2X, 'IFAULT')
4     FORMAT(1H , F5.2, 3(3X, F11.5), I3)
C
C     HERE WE REPRODUCE THE FIRST FIVE COLUMNS OF TABLE 1
C     IN HARTER AND MOORE JASA(1968), P. 889-901
C
      WRITE(IUNIT0, 1)
      DO 20 I = 1, 10
        Q1 = TENTH*FLOAT(I-1)
        IF (Q1.GT.ZERO) ZL = LOG(-LOG(ONE-Q1))
        JMAX = 11 - I
        DO 10 J = 1, JMAX
          Q2 = TENTH*FLOAT(J-1)
          ITYPE = 4
          IF (Q1.GT.ZERO.AND.Q2.EQ.ZERO) ITYPE = 3
          IF (Q1.EQ.ZERO.AND.Q2.GT.ZERO) ITYPE = 2
          IF (Q1.EQ.ZERO.AND.Q2.EQ.ZERO) ITYPE = 1
          IF (Q2.GT.ZERO) ZR = LOG(-LOG(Q2))
          CALL SEVINF(ITYPE, ZL, ZR, F11, F12, F22, IFAULT)
          DET = F11*F22 - F12*F12
          VAR1 = F22/DET
          VAR2 = F11/DET
          COV = -F12/DET
          WRITE(IUNIT0, 2) Q1, Q2, VAR1, VAR2, COV, IFAULT
10      CONTINUE
20    CONTINUE
C
C     HERE WE PARTIALLY REPRODUCE TABLE 1 IN MEEKER AND
C     NELSON, TECHNOMETRICS 1977, VOL. 19, NO. 4, 473-476
C
      WRITE(IUNIT1, 3)
      ITYPE = 2
      DO 30 J = 1, 91
```

```
        ZR = -SEVEN + TENTH*FLOAT(J-1)
        CALL SEVINF(ITYPE, ZL, ZR, F11, F12, F22, IFAULT)
        DET = F11*F22 - F12*F12
        VAR1 = F22/DET
        VAR2 = F11/DET
        COV = -F12/DET
        WRITE(IUNIT1, 4) ZR, VAR1, VAR2, COV, IFAULT
30    CONTINUE
      STOP
      END
```

```
      SUBROUTINE BTREE(FIND, LOC, NODE, LLINK, RLINK, BF, MAXT, NTREE,
     +          ROOT, LAST, IFAULT)
C
C       ALGORITHM AS219  APPL. STATIST. (1986) VOL. 35, NO. 2
C
C       BTREE returns the location (LOC) of an item (FIND) in a
C       height-balanced tree composed of NODE, LLINK, RLINK  and BF.
C       The tree is rooted at ROOT and contains NTREE nodes.   LAST
C       points to the next available free location in the tree.
C
C***    N.B. The user is expected to provided a function ICOMP to replace
C            the example provided here. ***
C
        INTEGER LLINK(MAXT), RLINK(MAXT), BF(MAXT)
        INTEGER NTREE, T, S, PNTR, R, A, MAXT, K, ROOT, ICOMP
        REAL    NODE(MAXT), FIND
C
        IFAULT = 0
        IF (NTREE .GE. 1) GO TO 20
C
C       First item in tree
C
        LAST = LAST + 1
        IF (LAST .GT. MAXT) THEN
          IFAULT = 1
          RETURN
        END IF
        NODE(LAST) = FIND
        LLINK(LAST) = 0
        RLINK(LAST) = 0
        BF(LAST) = 0
        ROOT = LAST
        NTREE = 1
        LOC = LAST
        RETURN
C
C       Search through tree for current item.
C       If found, return.    PNTR moves through the tree.   S points to
C       the place from which re-balancing may be needed, and T points
C       to the parent of S.
C
   20   T = 0
        S = ROOT
        PNTR = S
   30   K = ICOMP(FIND, NODE(PNTR))
        IF (K .NE. 0) GO TO 40
        LOC = PNTR
        RETURN
   40   IF (K .GT. 0) THEN
          LOC = RLINK(PNTR)
        ELSE IF (K .LT. 0) THEN
          LOC = LLINK(PNTR)
        END IF
        IF (LOC .NE. 0) GO TO 60
        NTREE = NTREE + 1
        LAST = LAST + 1
        IF (LAST .LE. MAXT) GO TO 50
        IFAULT = 1
        RETURN
C
   50   LOC = LAST
```

```
          IF (K .GT. 0) THEN
            RLINK(PNTR) = LOC
          ELSE IF (K .LT. 0) THEN
            LLINK(PNTR) = LOC
          END IF
          GO TO 100
C
   60     IF (BF(LOC) .EQ. 0) GO TO 70
          T = PNTR
          S = LOC
   70     PNTR = LOC
          GO TO 30
C
C         Add new node to tree and adjust balance factors
C
  100     NODE(LOC) = FIND
          LLINK(LOC) = 0
          RLINK(LOC) = 0
          BF(LOC) = 0
          A = ICOMP(FIND, NODE(S))
          IF (A .GT. 0) THEN
            PNTR = RLINK(S)
          ELSE IF (A .LT. 0) THEN
            PNTR = LLINK(S)
          END IF
          R = PNTR
C
  110     IF (PNTR .EQ. LOC) GO TO 130
          K = ICOMP(FIND, NODE(PNTR))
          IF (K .GT. 0) GO TO 120
          BF(PNTR) = -1
          PNTR = LLINK(PNTR)
          GO TO 110
  120     BF(PNTR) = 1
          PNTR = RLINK(PNTR)
          GO TO 110
C
  130     IF (BF(S) .NE. 0) GO TO 140
          BF(S) = A
          RETURN
  140     IF (BF(S) .NE. -A) GO TO 150
          BF(S) = 0
          RETURN
  150     IF (BF(R) .NE. A) GO TO 190
C
C         Single rotation
C
          PNTR = R
          IF (A .NE. -1) GO TO 170
          LLINK(S) = RLINK(R)
          RLINK(R) = S
          GO TO 180
  170     RLINK(S) = LLINK(R)
          LLINK(R) = S
  180     BF(S) = 0
          BF(R) = 0
          GO TO 240
C
C         Double rotation
C
  190     IF (A .NE. -1) GO TO 200
```

```
            PNTR = RLINK(R)
            RLINK(R) = LLINK(PNTR)
            LLINK(PNTR) = R
            LLINK(S) = RLINK(PNTR)
            RLINK(PNTR) = S
            GO TO 210
      200   PNTR = LLINK(R)
            LLINK(R) = RLINK(PNTR)
            RLINK(PNTR) = R
            RLINK(S) = LLINK(PNTR)
            LLINK(PNTR) = S
      210   BF(S) = 0
            BF(R) = 0
            IF (BF(PNTR) .NE. A) GO TO 220
            BF(S) = -A
            GO TO 230
      220   IF (BF(PNTR) .EQ. -A) BF(R) = A
      230   BF(PNTR) = 0
      240   IF (T .NE. 0) GO TO 250
            ROOT = PNTR
            RETURN
      250   IF (S .NE. RLINK(T)) THEN
              LLINK(T) = PNTR
            ELSE
              RLINK(T) = PNTR
            END IF
C
            RETURN
            END
C
C
C
            SUBROUTINE TREEP(PERM, LLINK, RLINK, MAXT, NTREE, ROOT, STACK,
     +             MAXSTA, IFAULT)
            INTEGER NTREE, PERM(NTREE), LLINK(MAXT), RLINK(MAXT)
            INTEGER PNTR, STACK(MAXSTA), SPTR, ROOT
C
C       Returns a permutation vector (PERM) to the tree contained in
C       NODE, LLINK, RLINK and rooted at ROOT
C
            IFAULT = 0
            IF (NTREE .LT. 1) RETURN
            SPTR = 0
            PNTR = ROOT
            I = 1
       10   IF (PNTR .EQ. 0) GO TO 20
C
C       Traverse left subtree
C
            SPTR = SPTR + 1
            IF (SPTR .GT. MAXSTA) GO TO 900
            STACK(SPTR) = PNTR
            PNTR = LLINK(PNTR)
            GO TO 10
       20   IF (SPTR .LT. 1) RETURN
C
C       Process NODE and traverse right subtree
C
            PNTR = STACK(SPTR)
            SPTR = SPTR - 1
            PERM(I) = PNTR
```

```
            I = I + 1
            PNTR = RLINK(PNTR)
            GO TO 10
C
  900       IFAULT = 1
            RETURN
            END
C
C
C
            INTEGER FUNCTION SEARCH(FIND, NODE, LLINK, RLINK, MAXT, NTREE,
     +                 ROOT)
            INTEGER LLINK(MAXT), RLINK(MAXT)
            INTEGER NTREE, PNTR, MAXT, ROOT, ICOMP
            REAL    NODE(MAXT), FIND
C
C           Returns the location in the tree of the item FIND.
C           Returns 0 if FIND is not in the tree.
C
            SEARCH = 0
            IF (NTREE .LT. 0) RETURN
            PNTR = ROOT
   10       K = ICOMP(FIND, NODE(PNTR))
            IF (K .EQ. 0) GO TO 20
            IF (K .LT. 0) THEN
              PNTR = LLINK(PNTR)
            ELSE
              PNTR = RLINK(PNTR)
            END IF
            IF (PNTR .NE. 0) GO TO 10
C
   20       SEARCH = PNTR
            RETURN
            END
C
C
C
            INTEGER FUNCTION ICOMP(X, Y)
            REAL    X, Y, EPS, Z
C
C           The body of ICOMP will be supplied by the user.
C           ICOMP will return 0 if X and Y are equal to the accuracy
C           needed, and return -1 if X < Y, and +1 if X > Y.
C           The example given tests whether X and Y differ by more than
C           0.001 of the larger in magnitude.
C
            DATA ZERO, ONE, EPS /0.0, 1.0, 0.001/
C
C           Avoid comparison of very different X and Y
C
            IF (X .LT. ONE .OR. Y .GT. -ONE) GO TO 10
            ICOMP = 1
            RETURN
   10       IF (X .GT. -ONE .OR. Y .LT. ONE) GO TO 20
            ICOMP = -1
            RETURN
C
   20       Z = MAX(ABS(X), ABS(Y))
            ICOMP = 0
            IF (Z .EQ. ZERO) RETURN
            Z = (X - Y) / Z
```

```
      IF (ABS(Z) .LT. EPS) RETURN
      ICOMP = 1
      IF (Z .GT. ZERO) RETURN
      ICOMP = -1
      RETURN
      END
```

```
      SUBROUTINE STEM(K, SDF, SDN, D, T, EPS, LG, ALGAMA, KW, WK, XPT,
     *  VAR, IFAULT)
C
C        ALGORITHM AS220  APPL. STATIST. (1986) VOL. 35, NO. 2
C
C        COMPUTES EXPECTATION AND VARIANCE OF STEIN AND
C        EFRON-MORRIS LIMITED TRANSLATION ESTIMATORS.
C
      REAL SDF, SDN, D, T(K), XPT(K), VAR(K), ALGAMA(LG), WK(KW)
      REAL EPS, EPS2, EPS3, HF, ONE, TWO, P, PM, RPM, DL, BD, C1, C2,
     *  C22, PNC, PP1, PP2
      REAL SJJ, BJ, BJ2, R0, R1, R2, SLL, BV, BL, BL2, QSM, QSM2, XPG,
     *  TMP
      REAL VC, DN, QTRM, EPS4, Q(101)
      INTEGER THR
      DATA ZERO, HF, ONE, TWO /0.0, 0.5, 1.0, 2.0/
      DATA THR /3/
      DATA EPS4 /1.0E-04/
      DATA ITYPE /1/
      LENQ = 101
      IF (K .LT. THR .OR. D .LT. ZERO .OR. EPS .LE. ZERO .OR. (SDF .GT.
     *  ZERO .AND. SDN .LE. ZERO)) GOTO 21
      IFAULT = 0
      DL = D
      P = K
      PM = P - TWO
      RPM = SQRT(PM)
      IF (D .GT. RPM) DL = RPM
      EPS2 = HF * HF * EPS
      F1 = ONE
      F2 = ONE
      IF (SDF .LE. ZERO) GOTO 1
      F1 = SDF / SDN
      F2 = SDF * (TWO + SDF) / (SDN * SDN)
    1 F3 = TWO * F1 + F2
      BD = DL * DL / PM
      C1 = F1 * TWO * RPM * DL
      C22 = F2 * DL * DL
      C2 = C22 * PM
      PNC = ZERO
      DO 2 J1 = 1, K
      PNC = PNC + T(J1) * T(J1)
    2 CONTINUE
      PNC = HF * PNC
C
C        COMPUTE EXPECTATION OF THE NP-COMPONENT ESTIMATOR
C
      DO 17 NP = 1, K
      DO 3 J2 = 1, KW
      WK(J2) = ZERO
    3 CONTINUE
      PP1 = HF * T(NP) * T(NP)
      PP2 = PNC - PP1
      EPS3 = EPS2
      IF (T(NP) .LE. ONE) GOTO 4
      EPS3 = EPS2 / (T(NP) * T(NP))
    4 XPT(NP) = ZERO
      VAR(NP) = ONE + T(NP) * T(NP)
C
C        CORRECTION FOR JAMES-STEIN-EFRON-MORRIS PARTS
C        SET UP INNER POISSON SUM
```

```
C
      JJ = 1
      WK(1) = EXP(-PP1)
      SJJ = WK(1)
      JXP = 0
      R0 = ONE
      R1 = ONE
    5 JJ = JJ + 1
      IF (JJ .GT. KW) GOTO 19
      BJ = JJ - 1
      WK(JJ) = WK(JJ - 1) * PP1 / BJ
      SJJ = SJJ + WK(JJ)
      IF (SJJ .GT. ONE) SJJ = ONE
      R2 = R1
      R1 = R0
      R0 = (ONE - SJJ) + WK(JJ)
      IF (JXP .GT. 0) GOTO 6
      IF (R0 * T(NP) * (ONE + HF * C1) .LE. EPS3) JXP = JJ + 1
      IF (JXP .EQ. 0) GOTO 5
    6 IF (R0 * C22 + R1 * (C1 * PP1 + F3) + TWO * F3 * PP1 * R2 .GT.
     *  EPS2) GOTO 5
      LL = 1
C
C        SET UP OUTER POISSON SUM
C
      LP = JJ + 1
      IF (LP + 1 .GT. KW) GOTO 19
      LXP = 0
      WK(LP + 1) = EXP(-PP2)
      SLL = WK(LP + 1)
      BV = PP1 * (C1 + 2 . * F3) + C22 + F3
    7 LL = LL + 1
      IF (LP + LL .GT. KW) GOTO 19
      BL = LL - 1
      WK(LP + LL) = WK(LP + LL - 1) * PP2 / BL
      SLL = SLL + WK(LP + LL)
      IF (SLL .GT. ONE) SLL = ONE
      R0 = ONE - SLL
      IF (LXP .GT. 0) GOTO 8
      IF (R0 * T(NP) * (ONE + HF * C1) .LE. EPS3) LXP = LL + 1
      IF (LXP .EQ. 0) GOTO 7
    8 IF (R0 * BV .GT. EPS2) GOTO 7
      KSTP = K + 2 * JXP + 2 * LXP - 3
      IF (KSTP .GT. LG) GOTO 18
      IF (2 * JXP + 2 .GT. LENQ) GOTO 20
      DO 12 L2P = 1, LXP
      L2 = L2P - 1
      BL2 = L2
      JDF = 2 * JXP
      LDF = K - 1 + 2 * L2
      DO 9 JX = 1, LENQ
      Q(JX) = ZERO
    9 CONTINUE
      QSM = ZERO
      QSM2 = ZERO
      DO 10 J2P = 1, JXP
      J2 = J2P - 1
      BJ2 = J2
      JD1 = 2 * J2 + 3
      JD2 = 2 * J2 + 4
      JSB = K + 2 * J2 + 2 * L2
```

```
         XPG = EXP(ALGAMA(JSB) - ALGAMA(JSB + 1) + ALGAMA(2 * J2 + 2)
        *  - ALGAMA(2 * J2 + 3))
         Q(JD1) = -HF * DL * T(NP) * RPM * XPG * WK(J2 + 1) * WK(LP + L2 +
        *  1)
         Q(JD2) = T(NP) * WK(J2 + 1) * WK(LP + L2 + 1) * PM / (P +
        *  TWO * BL2 + TWO * BJ2)
         QSM = QSM - Q(JD1)
         QSM2 = QSM2 + Q(JD1) + Q(JD2)
      10 CONTINUE
         TMP = QSM2
         IF (ABS(DL - RPM) .LE. EPS4) GOTO 11
         TMP = BMIX(BD, JDF, LDF, Q, ITYPE, IFAULT)
         IF (IFAULT .NE. 0) RETURN
      11 XPT(NP) = XPT(NP) - QSM - TMP
      12 CONTINUE
         XPT(NP) = T(NP) + F1 * XPT(NP)
C
C        COMPUTE VARIANCE OF THE NP-COMPONENT ESTIMATOR
C
         VC = ZERO
         LLP = LL + 1
         DO 16 L2P = 1, LLP
         L2 = L2P - 1
         BL2 = L2
         JDF = 2 * JJ
         LDF = K - 1 + 2 * L2
         QSM = C2 * WK(1) * WK(LP + L2 + 1) / (PM + TWO * BL2)
         DO 13 J2 = 1, LENQ
         Q(J2) = ZERO
      13 CONTINUE
         Q(2) = -QSM
         QSM2 = -QSM
         DO 14 J2P = 1, JJ
         J2 = J2P - 1
         BJ2 = J2
         JD1 = 2 * J2 + 3
         JD2 = 2 * J2 + 4
         JSB = K + 2 * J2 + 2 * L2
         DN = JSB
         QTRM = C2 * WK(J2 + 2) * WK(LP + L2 + 1) / DN
         Q(JD2) = -QTRM + WK(J2 + 1) * WK(LP + L2 + 1) * PM * (TWO * BJ2 +
        *  ONE) / DN * (-TWO * F1 + PM * F2 / (DN - TWO))
         XPG = EXP(ALGAMA(JSB) - ALGAMA(JSB + 1) + ALGAMA(2 * J2 + 2)
        *  - ALGAMA(2 * J2 + 1))
         WK(LG + JD1) = C1 * WK(J2 + 1) * WK(LP + L2 + 1) * XPG
         QSM = QSM + QTRM - WK(LG + JD1)
         QSM2 = QSM2 + WK(LG + JD1) + WK(LG + JD2)
      14 CONTINUE
         TMP = QSM2
         IF (ABS(DL - RPM) .LE. EPS4) GOTO 15
         TMP = BMIX(BD, JDF, LDF, Q, ITYPE, IFAULT)
         IF (IFAULT .NE. 0) RETURN
      15 VC = VC + QSM + TMP
      16 CONTINUE
         VAR(NP) = VAR(NP) + VC - XPT(NP) * XPT(NP)
      17 CONTINUE
         RETURN
      18 IFAULT = 6
         RETURN
      19 IFAULT = 7
         RETURN
```

```
      20 IFAULT = 8
         RETURN
      21 IFAULT = 9
         RETURN
         END
C
         SUBROUTINE ALGAM(N, ALGAMA)
C
C           COMPUTES ALOG(GAMMA(J/2)) FOR J = 1(1)N, N EVEN.
C           PL2 IS HALF LOG(PI).
C
         DIMENSION ALGAMA(N)
         DATA ZERO, HF, PL2 /0.0, 0.5, 0.572364943/
         NHLF = N / 2
         ALGAMA(1) = PL2
         ALGAMA(2) = ZERO
         DO 1 J1 = 2, NHLF
         B1 = FLOAT(2 * J1 - 3)
         B2 = FLOAT(J1 - 1)
         ALGAMA(2 * J1) = ALGAMA(2 * J1 - 2) + ALOG(B2)
         ALGAMA(2 * J1 - 1) = ALGAMA(2 * J1 - 3) + ALOG(HF * B1)
       1 CONTINUE
         RETURN
         END
```

```
      SUBROUTINE NPMLG(NI, NJ, ITER, ICYC, JN, DEPS, Y, S, R, P, DFL, X,
     *  W, M, V, AFI, LP, IFAULT)
C
C        ALGORITHM AS221  APPL. STATIST. (1986) VOL. 35, NO. 3
C
C        SUBROUTINE FOR CALCULATING THE NONPARAMETRIC MAXIMUM
C        LIKELIHOOD ESTIMATE OF THE MIXING DISTRIBUTION IN
C        A MIXTURE OF NORMALS
C
C     N.B. The dimension of array R below has been changed in accordance
C          with the correction on page 176 of vol. 39 (1990) of Applied
C          Statistics.
C
      REAL R(0:NI), P(NI), Y(NI), S(NI), M(NI), V(NI), AFI(NI), X, W,
     *  DEPS, DFL, RMIN, RMAX, RCONS, A, Q, D, DENOM, PR, FL, DIFF,
     *  POLD, ROLD, PQ, RR, AI, ANB, TEMP, DFOLD, AN
      INTEGER LP(ICYC), NI, NJ, ITER, ICYC, JN, IFAULT, JJ, IT, IC, IJ,
     *  JNN, NEXLAS, MORE
      DATA DCONS /1.0E-6/, RCONS /1.0E-2/, DINF /1.0E-6/
      DATA ZERO /0.0/, ONE /1.0/, TWO /2.0/, PI /6.2831853/
C
C        CHECK INPUT VALUES FOR SAMPLING VARIANCES
C
      IFAULT = 0
      DO 1 I = 1, NI
      IF (S(I) .LE. ZERO) IFAULT = 2
      IF (IFAULT .EQ. 2) GOTO 999
    1 CONTINUE
C
C        SET THE INITIAL ESTIMATE : UNIFORM DISTRIBUTION OVER
C        THE RANGE OF OBSERVED VALUES
C
      FNJ = FLOAT(NJ)
      FNI = FLOAT(NI)
      POLD = ONE / FNJ
      RMIN = Y(1)
      RMAX = Y(1)
      DO 5 I = 2, NI
      IF (RMIN .GT. Y(I)) RMIN = Y(I)
      IF (RMAX .LT. Y(I)) RMAX = Y(I)
    5 CONTINUE
      PQ = (RMAX - RMIN) / (FNJ - ONE)
      DO 10 J = 1, NJ
      FJ = FLOAT(J) - ONE
      R(J) = RMIN + PQ * FJ
      P(J) = POLD
   10 CONTINUE
C
C        CALCULATE GRID SIZE FOR EVALUATING CONVERGENCE
C        TO GLOBAL MAXIMUM
C
      CALL GRIDN(NI, Y, JN, RMIN, RMAX, RCONS, DEPS)
C
C        START THE ITERATIONS
C
      DFOLD = DINF
C
C        DFOLD REPRESENTS THE VALUE OF THE LOG LIKELIHOOD
C        FUNCTION FROM THE PREVIOUS ITERATION
C
      IT = 0
```

```
      IC = ICYC + 1
   30 IT = IT + 1
C
C        'IT' COUNTS THE NUMBER OF EM CYCLES REQUIRED
C        TO REACH CONVERGENCE AT GLOBAL MAXIMUM
C
      DO 75 LPOUT = 1, ITER
      DFL = ZERO
      LP(IT) = LPOUT
      DO 40 I = 1, NI
      FL = ZERO
      DO 35 J = 1, NJ
      PR = (Y(I) - R(J)) * (Y(I) - R(J)) / (TWO * S(I))
      A = EXP(-PR) / SQRT(PI * S(I))
      FL = FL + P(J) * A
   35 CONTINUE
C
C        A REPRESENTS THE SAMPLING DENSITY
C        FL REPRESENTS THE MARGINAL DENSITY
C        DFL REPERESENTS THE LOG LIKELIHOOD FUNCTION
C
      DFL = DFL + LOG(FL)
      AFI(I) = FL
   40 CONTINUE
      DIFF = ABS((DFL - DFOLD) / DFOLD)
C
C        EVALUATE WHETHER THE RELATIVE CHANGE IN LOG
C        LIKELIHOOD IS LESS THAN DCONS
C
      IF (DIFF .LE. DCONS) GOTO 80
      DFOLD = DFL
C
C        CALCULATE THE NEW ESTIMATE
C
      DO 50 J = 1, NJ
      POLD = P(J)
      ROLD = R(J)
      R(J) = ZERO
      P(J) = ZERO
      DENOM = ZERO
      DO 45 I = 1, NI
      PR = (Y(I) - ROLD) * (Y(I) - ROLD) / (TWO * S(I))
      A = EXP(-PR) / SQRT(PI * S(I))
      Q = A / (S(I) * AFI(I))
      R(J) = R(J) + Y(I) * Q
      P(J) = P(J) + A / AFI(I)
      DENOM = DENOM + Q
   45 CONTINUE
      R(J) = R(J) / DENOM
      P(J) = POLD * P(J) / FNI
   50 CONTINUE
C
C        COLLAPSE THE NUMBER OF SUPPORT POINTS
C
      JJ = 1
      DO 70 J = 2, NJ
      DIFF = ABS(R(J) - R(JJ))
      IF (DIFF .LE. DEPS) GOTO 60
      JJ = JJ + 1
      R(JJ) = R(J)
      P(JJ) = P(J)
```

```
         GOTO 65
      60 CONTINUE
         P(JJ) = P(JJ) + P(J)
      65 CONTINUE
      70 CONTINUE
         NJ = JJ
      75 CONTINUE
C
C        END OF ITERATIONS
C
      80 CONTINUE
C
C        CHECK IF THE ESTIMATE IS THE GLOBAL MAXIMUM
C
         IF (IT .EQ. IC) GOTO 169
         R(0) = RMIN
         IJ = 0
         JNN = 1
         FJN = FLOAT(JN)
         PQ = (RMAX - RMIN) / (FJN - ONE)
         DO 130 J = 1, JN
         FJ = FLOAT(J) - ONE
         RR = RMIN + PQ * FJ
         IF (RR .LT. R(IJ)) GOTO 130
         AI = ZERO
         DO 90 I = 1, NI
         PR = (Y(I) - RR) ** 2
         PR = PR / (TWO * S(I))
         A = EXP(-PR) / SQRT(PI * S(I))
         AI = AI + (A / AFI(I))
      90 CONTINUE
         DIFF = AI - FNI
C
C        CHECK IF THE CRITERION OF GLOBAL MAXIMUM
C        HOLDS FOR ALL VALUES IN THE RANGE OF THE
C        OBSERVATIONS
C
         IF (DIFF .LE. DINF) GOTO 130
         DO 100 IJ = 1, JJ
         ANB = RR - R(IJ)
         AN = ABS(ANB)
         IF (AN .LE. DEPS) GOTO 130
         IF (ANB .LE. ZERO) GOTO 105
    100 CONTINUE
    105 CONTINUE
C
C        IF THE CRITERION OF GLOBAL MAXIMUM IS NOT MET,
C        NEW SUPPORT IS ADDED AT THE POINT OF FAILURE
C
         JNN = 0
         IF (NJ .GE. NI) GOTO 152
         NJ = NJ + 1
         FNJ = FLOAT(NJ)
         DO 110 I = 1, NJ
         P(I) = ONE / FNJ
    110 CONTINUE
         R(NJ) = (R(IJ) + RR) / TWO
         IF (RR .GE. R(JJ)) R(NJ) = (RMAX + RR) / TWO
         IF (RR .GE. R(JJ)) GOTO 140
    130 CONTINUE
    140 CONTINUE
```

```
C
C        CHECK IF THE CRITERION OF GLOBAL MAXIMUM
C        HOLDS AT THE ESTIMATED POINTS OF SUPPORT
C
      DO 150 J = 1, JJ
      AI = ZERO
      DO 145 I = 1, NI
      PR = (Y(I) - R(J)) * (Y(I) - R(J)) / (TWO * S(I))
      A = EXP(-PR) / SQRT(PI * S(I))
      A = A / AFI(I)
      AI = AI + A
  145 CONTINUE
      DIFF = ABS(AI - FNI)
      IF (DIFF .LE. DINF) GOTO 150
      IF (P(J) .LE. DINF) GOTO 150
      JNN = 0
  150 CONTINUE
  152 JJ = NJ
C
C        IF NO NEW SUPPORT IS ADDED, THEN CONVERGENCE TO
C        GLOBAL MAXIMUM IS ASSUMED
C
      IF (JNN .EQ. 1) GOTO 170
C
C        SORT THE POINTS OF SUPPORT IN AN ASCENDING ORDER
C
      NEXLAS = NJ - 1
      MORE = 0
  155 IF (MORE .EQ. 1) GOTO 165
      MORE = 1
      DO 160 I = 1, NEXLAS
      J = I + 1
      IF (R(I) .LE. R(J)) GOTO 160
      TEMP = R(I)
      R(I) = R(J)
      R(J) = TEMP
      MORE = 0
  160 CONTINUE
      GOTO 155
  165 CONTINUE
C
C        IF THE GLOBAL MAXIMUM IS NOT REACHED AND IF THE
C        MAXIMUM ALLOWED NUMBER OF CYCLES IS NOT EXHAUSTED,
C        THEN A NEW CYCLE IS RESTARTED WITH THE NEW SUPPORT
C
      IF (IT .LE. ICYC) GOTO 30
C
C        IF THE GLOBAL MAXIMUM IS NOT REACHED AND IF THE
C        MAXIMUM ALLOWED NUMBER OF CYCLES IS EXHAUSTED,
C        THEN IFAULT IS SET TO 1
C
  169 IFAULT = 1
  170 CONTINUE
      ICYC = IT
      IF (IFAULT .EQ. 1) ICYC = ICYC - 1
      X = ZERO
      W = ZERO
C
C        CALCULATE THE PRIOR MEAN AND VARIANCE
C
      DO 175 J = 1, NJ
```

```
          X = X + P(J) * R(J)
      175 CONTINUE
          DO 180 J = 1, NJ
          W = W + P(J) * ((R(J) - X) ** 2)
      180 CONTINUE
C
C        CALCULATE THE POSTERIOR MEANS AND VARIANCES
C
          DO 195 I = 1, NI
          Q = ZERO
          D = ZERO
          DO 185 J = 1, NJ
          PR = (Y(I) - R(J)) * (Y(I) - R(J)) / (TWO * S(I))
          A = EXP(-PR) / SQRT(PI * S(I))
          A = A / AFI(I)
          Q = Q + P(J) * R(J) * A
      185 CONTINUE
          DO 190 J = 1, NJ
          PR = (Y(I) - R(J)) * (Y(I) - R(J)) / (TWO * S(I))
          A = EXP(-PR) / SQRT(PI * S(I))
          A = A / AFI(I)
          D = D + P(J) * A * ((R(J) - Q) ** 2)
      190 CONTINUE
          M(I) = Q
          V(I) = D
      195 CONTINUE
      999 CONTINUE
          RETURN
          END
C
          SUBROUTINE GRIDN(NI, Y, JN, RMIN, RMAX, RCONS, DEPS)
C
C        ALGORITHM AS221  APPL. STATIST. (1986) VOL. 35, NO. 3
C
C        AUXILIARY SUBROUTINE FOR CALCULATING THE GRID SIZE TO
C        BE USED IN EVALUATING CONVERGENCE TO THE GLOBAL MAXIMUM
C
          REAL Y(NI), RMIN, RMAX, RCONS, DEPS, SUM, AVE, VAR, SD
          INTEGER NI, JN
          DATA ZERO /0.0/
          SUM = ZERO
          VAR = ZERO
          DO 5 I = 1, NI
          SUM = SUM + Y(I)
        5 CONTINUE
          FNI = FLOAT(NI)
          AVE = SUM / FNI
          DO 10 I = 1, NI
          VAR = VAR + (Y(I) - AVE) ** 2
       10 CONTINUE
          VAR = VAR / (FNI - 1.0)
          SD = SQRT(VAR)
          DEPS = RCONS * SD
          JN = (RMAX - RMIN) / DEPS
          RETURN
          END
```

```
      SUBROUTINE RESSMO(X, Y, NIN, CUTOFF, XLO, XHI, BANDW, SMOOTH,
     +            NOUT, NEWCAL, IWK, W, WA, WB, WC, WK, IFAULT)
C
C      ALGORITHM AS222  APPL. STATIST. (1987) VOL. 36, NO. 1
C      Modified by incorporating AS R73 (Appl. Statist. (1988) vol. 37
C      (2), p.316)
C
C      Auxiliary routine required: FASTF = algorithm AS 83 as amended
C      in Griffiths & Hill, 1985 (i.e. with extra final argument)
C
      INTEGER   NIN, NOUT, IWK(NIN), NEWCAL, IFAULT
      REAL      X(NIN), Y(NIN), CUTOFF, XLO, XHI, BANDW, SMOOTH(NOUT),
     +          W(NIN), WA(NOUT), WB(NOUT), WC(NOUT), WK(6, NOUT)
C
C      Local variables
C
      INTEGER NPOW, M, K, N2, JHI, JMAX, I, J, NC, ICAL
      REAL    ZERO, ONE, TWO, PI, BIG, RANGE, XSTEP, A, B, XLO1, TEMP,
     +        RES
      DATA ZERO, ONE, TWO, PI, BIG /0.0, 1.0, 2.0, 3.14159, 30.0/
C
      RANGE = XHI - XLO
C
C      Check cutoff parameter
C
      IF (CUTOFF .LE. ZERO) THEN
        IFAULT = 1
        RETURN
      END IF
C
C      Check if NOUT is a power of 2
C
      NPOW = 8
      M = 16
      DO 2 K = 3, 11
        IF (NPOW .EQ. NOUT) GO TO 3
        M = M * 2
        NPOW = NPOW * 2
    2 CONTINUE
      IFAULT = 2
      RETURN
C
C      Check range
C
    3 IF (RANGE .LE. ZERO) THEN
        IFAULT = 3
        RETURN
      END IF
C
C      Check bandwidth
C
      IF (BANDW .LE. ZERO) THEN
        IFAULT = 4
        RETURN
      END IF
C
C      Check NEWCAL
C
      IF (NEWCAL .LT. 0 .OR. NEWCAL .GT. 2) THEN
        IFAULT = 5
        RETURN
```

```
      END IF
C
      XSTEP = RANGE / FLOAT(NOUT)
      A = FLOAT(NOUT) / (RANGE * FLOAT(NIN))
      N2 = NOUT / 2
      B = TWO * (PI * BANDW / RANGE)**2
      IF (NEWCAL .EQ. 2) GO TO 8
C
      JHI = SQRT(BIG / B)
      JMAX = MIN(N2-1, JHI)
      XLO1 = XLO - XSTEP
      IF (NEWCAL .EQ. 0) THEN
        DO 5 I = 1, NIN
          IWK(I) = (X(I) - XLO1) / XSTEP
          W(I) = ONE
    5   CONTINUE
      END IF
C
C      Refresh former results
C
      IF (NEWCAL .EQ. 1) THEN
        DO 52 J = 1, NOUT
          WA(J) = WK(5, J)
          WB(J) = WK(6, J)
   52   CONTINUE
      END IF
C
C      Find numerator of Nadaraya-Watson estimate
C
      CALL LINSMO(Y, IWK, NIN, SMOOTH, NOUT, NEWCAL, WA, WB, WC, N2,
     +            JHI, JMAX, A, B, M)
C
C     Transfer results to work area
C
      DO 54 J = 1, NOUT
        WK(3, J) = WA(J)
        WK(4, J) = WB(J)
        WK(1, J) = SMOOTH(J)
   54 CONTINUE
C
C     Refresh former results
C
      IF (NEWCAL .EQ. 1) THEN
        DO 55 J = 1, NOUT
          WA(J) = WK(5, J)
          WB(J) = WK(6, J)
   55   CONTINUE
      END IF
C
C      Find density estimate of marginal distribution of X.
C
      CALL LINSMO(W, IWK, NIN, SMOOTH, NOUT, NEWCAL, WA, WB, WC, N2,
     +            JHI, JMAX, A, B, M)
      DO 65 J = 1, NOUT
        WK(5, J) = WA(J)
        WK(6, J) = WB(J)
   65 CONTINUE
C
C      Compute Nadaraya-Watson estimate
C
      DO 7 J = 1, NOUT
```

```
          TEMP = ZERO
          IF (SMOOTH(J) .GT. ZERO) TEMP = WK(1, J) / SMOOTH(J)
          WK(1, J) = TEMP
    7 CONTINUE
C
C      Compute Huber's PSI from the residuals
C
    8 NC = 0
      DO 11 I = 1, NIN
        RES = Y(I) - WK(1, IWK(I))
        IF (RES .GT. -CUTOFF) GO TO 9
        NC = NC + 1
        W(I) = -CUTOFF
        GO TO 11
    9   IF (RES .LT. CUTOFF) GO TO 10
        NC = NC + 1
        W(I) = CUTOFF
        GO TO 11
   10   W(I) = RES
   11 CONTINUE
C
C      Compute non-linear correction
C
      IFAULT = -NC
      ICAL = 0
      CALL LINSMO(W, IWK, NIN, SMOOTH, NOUT, ICAL, WA, WB, WC, N2, JHI,
     +            JMAX, A, B, M)
C
C      Store the results
C
      DO 111 J = 1, NOUT
  111 WK(2, J) = SMOOTH(J)
C
C      Derivative of Huber's PSI from residuals
C
      DO 112 I = 1, NIN
        TEMP = ZERO
        IF (W(I) .LT. CUTOFF .AND. W(I) .GT. -CUTOFF) TEMP = ONE
        W(I) = TEMP
  112 CONTINUE
C
C      Compute denominatot of non-linear correction
C
      CALL LINSMO(W, IWK, NIN, SMOOTH, NOUT, ICAL, WA, WB, WC, N2, JHI,
     +            JMAX, A, B, M)
C
C      Compute the full estimator
C
      DO 13 J = 1, NOUT
        TEMP = WK(1, J)
        IF (SMOOTH(J) .GT. ZERO) TEMP = TEMP + WK(2, J) / SMOOTH(J)
        SMOOTH(J) = TEMP
   13 CONTINUE
C
      RETURN
      END
C
C
C
      SUBROUTINE LINSMO(Y, IWK, NIN, SMOOTH, NOUT, NEWCAL, WA, WB, WC,
     +            N2, JHI, JMAX, A, B, M)
```

```
C
C      ALGORITHM AS222   APPL. STATIST. (1987) VOL. 36, NO. 1
C
      INTEGER   NIN, IWK(NIN), NOUT, NEWCAL, N2, JHI, JMAX, M
      REAL      Y(NIN), SMOOTH(NOUT), WA(NOUT), WB(NOUT),
     +            WC(NOUT), A, B
C
C      Local variables
C
      INTEGER M1, J, I, JI, ITYPE, J1, J2, JHI2, IFAULT
      REAL ZERO, TEMP, C
      DATA ZERO/0.0/
C
      M1 = M / 2
      IF (NEWCAL .NE. 0) GO TO 30
C
C      Transform
C
      DO 10 J = 1, NOUT
        WA(J) = ZERO
        WB(J) = ZERO
   10 CONTINUE
C
      DO 20 I = 1, NIN
        JI = IWK(I)
        IF (JI .LT. 1 .OR. JI .GT. NOUT) GO TO 20
        WA(JI) = WA(JI) + A*Y(I)
   20 CONTINUE
      ITYPE = 1
C
      CALL FASTF(WA, WB, M1, ITYPE, IFAULT)
C
C      Filter Fourier transform
C
   30 SMOOTH(1) = WA(1)
      WC(1) = WB(1)
      DO 40 J = 1, JMAX
        C = EXP(-B * FLOAT(J*J))
        J1 = J + 1
        J2 = NOUT - J + 1
        SMOOTH(J1) = C * WA(J1)
        WC(J1) = C * WB(J1)
        SMOOTH(J2) = C * WA(J2)
        WC(J2) = C * WB(J2)
   40 CONTINUE
C
C      Underflow correction
C
      IF (JHI + 1 - N2 .GT. 0) THEN
        TEMP = EXP(-B * FLOAT(N2*N2))
        SMOOTH(N2 + 1) = TEMP * WA(N2 + 1)
        WC(N2 + 1) = TEMP * WB(N2 + 1)
      ELSE IF (JHI + 1 - N2 .LT. 0) THEN
        JHI2 = JHI + 2
        DO 61 J1 = JHI2, N2
          J2 = NOUT - J1 + 2
          SMOOTH(J1) = ZERO
          WC(J1) = ZERO
          SMOOTH(J2) = ZERO
          WC(J2) = ZERO
   61   CONTINUE
```

```
            SMOOTH(N2 + 1) = ZERO
            WC(N2 + 1) = ZERO
          ELSE
            SMOOTH(N2 + 1) = ZERO
            WC(N2 + 1) = ZERO
          END IF
C
          ITYPE = -1
          CALL FASTF(SMOOTH, WC, M1, ITYPE, IFAULT)
C
          RETURN
          END
```

```
      SUBROUTINE ORPS1(ALPHA, LAMDA, DELTA, IP, SIGMA, KNEW, FM10, FM1K,
     *          FM20, FM2K, ITER, IFAULT)
C
C     ALGORITHM AS223  APPL. STATIST. (1987) VOL. 36, NO. 1
C
C     Calculates the optimum ridge parameter under the criterion of
C     minimizing the mean squared error of parameter estimation.
C
      INTEGER FM20, IMAX, IP, ITER, IFAULT
      REAL ALPHA(IP), LAMDA(IP), DELTA, K, KNEW, KOLD, DK, FM10, FM1K,
     *  FM2K, DFM1K, DDFM1K, S, SIGMA
      REAL ZERO, TWO, THREE
C
      FM1(X, Y) = (Y * S**2 + (X * K)**2) / ((Y + K)**2 * S**2)
      FM2(X, Y) = Y * FM1(X, Y)
      DFM1(X, Y) = Y * (X**2 * K - S**2) / (Y + K)**3
      DDFM1(X, Y) = Y * (Y * X**2 - TWO * X**2 * K + THREE * S**2) /
     *                  (Y + K)**4
      DATA ZERO/0.0/, TWO/2.0/, THREE/3.0/
C
      IFAULT = 1
      IF (DELTA .LE. ZERO) RETURN
      IFAULT = 2
      IF (SIGMA .LE. ZERO) RETURN
      IFAULT = 3
      DO 10 J = 1, IP
        IF (LAMDA(J) .LE. ZERO) RETURN
   10 CONTINUE
      IFAULT = 0
      K = ZERO
      FM10 = ZERO
      S = SIGMA
      DO 20 J = 1, IP
        FM10 = FM10 + FM1(ALPHA(J), LAMDA(J))
   20 CONTINUE
      FM20 = IP
      ITER = 0
      IMAX = 20
      KNEW = ZERO
      DO 40 I = 1, IMAX
        ITER = ITER + 1
        FM1K = ZERO
        FM2K = ZERO
        DFM1K = ZERO
        DDFM1K = ZERO
        DO 30 J = 1, IP
          FM1K = FM1K + FM1(ALPHA(J), LAMDA(J))
          FM2K = FM2K + FM2(ALPHA(J), LAMDA(J))
          DFM1K = DFM1K + DFM1(ALPHA(J), LAMDA(J))
          DDFM1K = DDFM1K + DDFM1(ALPHA(J), LAMDA(J))
   30   CONTINUE
        KOLD = KNEW
        KNEW = KOLD - DFM1K / DDFM1K
        K = KNEW
        DK = ABS(KNEW - KOLD)
        IF (DK .LE. DELTA) GO TO 50
   40 CONTINUE
   50 RETURN
      END
C
C
```

```
C
      SUBROUTINE ORPS2(ALPHA, LAMDA, DELTA, IP, SIGMA, KNEW, FM10, FM1K,
     *          FM20, FM2K, ITER, IFAULT)
C
C     ALGORITHM AS223  APPL. STATIST. (1987) VOL. 36, NO. 1
C
C     Calculates the optimum ridge parameter under the criterion of
C     minimizing the mean squared error of prediction.
C
      INTEGER FM20, IMAX, IP, ITER, IFAULT
      REAL ALPHA(IP), LAMDA(IP), DELTA, K, KNEW, KOLD, DK, FM10, FM1K,
     *  FM2K, DFM2K, DDFM2K, S, SIGMA
      REAL ZERO, TWO, THREE
C
      FM1(X, Y) = (Y * S**2 + (X * K)**2) / ((Y + K)**2 * S**2)
      FM2(X, Y) = Y * FM1(X, Y)
      DFM1(X, Y) = Y * (X**2 * K - S**2) / (Y + K)**3
      DDFM1(X, Y) = Y * (Y * X**2 - TWO * X**2 * K + THREE * S**2) /
     *                 (Y + K)**4
      DDFM2(X, Y) = Y * DDFM1(X, Y)
      DATA ZERO/0.0/, TWO/2.0/, THREE/3.0/
C
      IFAULT = 1
      IF (DELTA .LE. ZERO) RETURN
      IFAULT = 2
      IF (SIGMA .LE. ZERO) RETURN
      IFAULT = 3
      DO 10 J = 1, IP
        IF (LAMDA(J) .LE. ZERO) RETURN
   10 CONTINUE
      IFAULT = 0
      K = ZERO
      FM10 = ZERO
      S = SIGMA
      DO 20 J = 1, IP
        FM10 = FM10 + FM1(ALPHA(J), LAMDA(J))
   20 CONTINUE
      FM20 = IP
      ITER = 0
      IMAX = 20
      KNEW = ZERO
      DO 40 I = 1, IMAX
        ITER = ITER + 1
        FM1K = ZERO
        FM2K = ZERO
        DFM2K = ZERO
        DDFM2K = ZERO
        DO 30 J = 1, IP
          FM1K = FM1K + FM1(ALPHA(J), LAMDA(J))
          FM2K = FM2K + FM2(ALPHA(J), LAMDA(J))
          DFM2K = DFM2K + DFM2(ALPHA(J), LAMDA(J))
          DDFM2K = DDFM2K + DDFM2(ALPHA(J), LAMDA(J))
   30   CONTINUE
        KOLD = KNEW
        KNEW = KOLD - DFM2K / DDFM2K
        K = KNEW
        DK = ABS(KNEW - KOLD)
        IF (DK .LE. DELTA) GO TO 50
   40 CONTINUE
   50 RETURN
      END
```

```
      SUBROUTINE COMBDD(ND, NU, IV, NB, MAXNB, KBS, MAXKBS, NCOMP, IDES,
     *  IFAULT)
C
C        ALGORITHM AS224 APPL. STATIST. (1987) VOL. 36, NO. 2
C
      DIMENSION IB(5), NCOMP(5, 10, 36), KBS(5), IDES(72), NB(5)
      DIMENSION ILIST(72), IR(72)
C
C        COMBINES ND COMPONENT DESIGNS TO FORM A DESIGN WITH
C        ND ORTHOGONAL BLOCKING FACTORS
C
C        NTH COMPONENT DESIGN STORED IN NCOMP(N,.,.)
C        ..    ..       ..   HAS NB(N) BLOCKS OF SIZE KBS(N)
C
C
C        CHECK PARAMETERS
C
      IFAULT = 1
      IF (ND .LT. 2) RETURN
      M = 1
      DO 1 N = 1, ND
      IFAULT = 2
      L = NB(N) * KBS(N)
      IF (L .NE. NU) RETURN
      IFAULT = 3
      IF (NB(N) .GT. MAXNB) RETURN
      IFAULT = 4
      IF (KBS(N) .GT. MAXKBS) RETURN
      M = M * NB(N)
    1 CONTINUE
      IFAULT = 5
      IF (M .NE. NU) RETURN
      IFAULT = 6
C
C        INITIALISE ILIST AND IR
C        ILIST HOLDS TREATMENT LABELS AND IR THE REPLICATIONS
C
      DO 2 I = 1, IV
      ILIST(I) = 0
      IR(I) = 0
    2 CONTINUE
C
C        INITIALISE IDES.  IDES HOLDS COMBINED DESIGN
C
      IL = 0
      ID = 0
      INB = NB(1)
      K = KBS(1)
      DO 5 I = 1, INB
      DO 5 J = 1, K
      ID = ID + 1
      IDES(ID) = -1
C
C        SET ITR EQUAL TO NEXT TREATMENT
C        IN FIRST COMPONENT DESIGN
C
      ITR = NCOMP(1, I, J)
C
C        IF ITR IS ALREADY RECORDED IN ILIST, THEN
C        ADD ONE TO CORRESPONDING ELEMENT IN IR
C
```

```
      IF (IL .EQ. 0) GOTO 4
      DO 3 JL = 1, IL
      IF (ILIST(JL) .NE. ITR) GOTO 3
      IR(JL) = IR(JL) + 1
      GOTO 5
    3 CONTINUE
C
C        RECORD ITR IN ILIST
C        SET CORRESPONDING ELEMENT IN IR EQUAL TO 1
C
    4 IL = IL + 1
      IF (IL .GT. IV) RETURN
      ILIST(IL) = ITR
      IR(IL) = 1
    5 CONTINUE
      IFAULT = 0
C
C        CHECK EVERY COMPONENT DESIGN CONTAINS SAME NUMBER
C        OF OCCURRENCES OF EVERY TREATMENT LABEL
C
      DO 7 JL = 1, IV
      ITR = ILIST(JL)
      DO 7 JN = 2, ND
      KREP = 0
      INB = NB(JN)
      K = KBS(JN)
      DO 6 I = 1, INB
      DO 6 J = 1, K
      IF (NCOMP(JN, I, J) .NE. ITR) GOTO 6
      KREP = KREP + 1
    6 CONTINUE
      IF (KREP .EQ. IR(JL)) GOTO 7
      IFAULT = 7
      RETURN
    7 CONTINUE
      KREP = 0
C
C        SET ITR EQUAL TO FIRST TREATMENT TO BE FITTED
C
      IL = 1
      ITR = ILIST(IL)
      LSTFIT = 0
C
C        RETURN COMBINED DESIGN
C
    8 IF (IL .EQ. IV .AND. KREP .EQ. IR(IL)) RETURN
C
C        INITIALISE IB.  IB(N) HOLDS NUMBER OF
C        CURRENT BLOCK IN NTH COMPONENT DESIGN
C
      DO 9 N = 1, ND
    9 IB(N) = 0
      N = 1
C
C        SET KREP EQUAL TO NEXT OCCURRENCE OF TREATMENT TO BE FITTED
C
      KREP = KREP + 1
      IF (KREP .LE. IR(IL)) GOTO 11
C
C        ALL OCCURRENCES OF PRESENT TREATMENT HAVE BEEN FITTED
C        FIND NEXT TREATMENT
```

```
C
      KREP = 1
      IL = IL + 1
      ITR = ILIST(IL)
C
C        INITIALISE THE MARKER LSTFIT
C        LSTFIT PREVENTS ANY TREATMENT BEING FITTED IN
C        AN EARLIER POSITION IN IDES
C
      LSTFIT = 0
      GOTO 11
C
C        MOVE TO PREVIOUS COMPONENT DESIGN
C
   10 IB(N) = 0
      N = N - 1
      IF (N .EQ. 1) GOTO 18
C
C        MOVE TO NEXT BLOCK IN PRESENT COMPONENT DESIGN
C
   11 IB(N) = IB(N) + 1
      IF (IB(N) .GT. NB(N)) GOTO 10
C
C        LOCATE IN PRESENT COMPONENT DESIGN
C        NEXT OCCURRENCE OF TREATMENT TO BE FITTED
C
   12 IS = IB(N)
      INB = NB(N)
      IBS = KBS(N)
      DO 13 I = IS, INB
      DO 13 J = 1, IBS
      IF (NCOMP(N, I, J) .NE. ITR) GOTO 13
      IB(N) = I
      IF (N .EQ. ND) GOTO 14
C
C        MOVE TO NEXT COMPONENT DESIGN
C
      N = N + 1
      GOTO 11
   13 CONTINUE
      GOTO 10
C
C        SEE IF CURRENT POSITION IN D-DIMENSIONAL DESIGN IS VACANT
C
   14 ID = IB(1) - 1
      DO 15 N1 = 2, ND
   15 ID = (ID * NB(N1)) + IB(N1) - 1
      ID = ID + 1
      IF (ID .LE. LSTFIT .OR. IDES(ID) .NE. -1) GOTO 11
C
C        FIT TREATMENT IN D-DIMENSIONAL DESIGN
C        SET LSTFIT EQUAL TO POSITION JUST FILLED
C
      IDES(ID) = ITR
      LSTFIT = ID
C
C        SET FITTED TREATMENT NEGATIVE IN COMPONENT DESIGNS
C
      DO 17 N = 1, ND
      I = IB(N)
      IBS = KBS(N)
```

```
      DO 16 J = 1, IBS
      IF (NCOMP(N, I, J) .NE. ITR) GOTO 16
      NCOMP(N, I, J) = -ITR
      IF (ITR .EQ. 0) NCOMP(N, I, J) = -9999
      GOTO 17
   16 CONTINUE
   17 CONTINUE
C
      GOTO 8
C
C        CURRENT OCCURRENCE OF TREATMENT CANNOT BE FITTED
C        DETERMINE PREVIOUSLY FITTED OCCURRENCE OF TREATMENT
C
   18 KREP = KREP - 1
      IF (KREP .GE. 1) GOTO 20
      IL = IL - 1
      IF (IL .GT. 0) GOTO 19
C
C        DESIGNS CANNOT BE AMALGAMATED
C
      IFAULT = 8
      RETURN
C
C        DETERMINE LAST OCCURRENCE OF PREVIOUSLY FITTED TREATMENT
C
   19 ITR = ILIST(IL)
      KREP = IR(IL)
C
C        REMOVE OCCURRENCE OF TREATMENT FROM ND-DIMENSIONAL DESIGN
C        SET LSTFIT EQUAL TO VACATED POSITION
C
   20 DO 21 I = 1, NU
      ID = NU - I + 1
      IF (IDES(ID) .EQ. ITR) GOTO 22
   21 CONTINUE
   22 IDES(ID) = -1
      LSTFIT = ID
C
C        FIND LOCATION OF REMOVED TREATMENT IN COMPONENT DESIGNS
C
      ID = ID - 1
      DO 23 N = 1, ND
      I = ND - N + 1
      I1 = ID
      ID = ID / NB(I)
      IB(I) = I1 - (ID * NB(I)) + 1
   23 CONTINUE
C
C        SET REMOVED TREATMENT POSITIVE IN COMPONENT DESIGNS
C
      DO 25 N = 1, ND
      I = IB(N)
      IBS = KBS(N)
      IF (ITR .EQ. 0) ITR = 9999
      DO 24 J = 1, IBS
      IF (NCOMP(N, I, J) .NE. -ITR) GOTO 24
      IF (ITR .EQ. 9999) ITR = 0
      NCOMP(N, I, J) = ITR
      GOTO 25
   24 CONTINUE
   25 CONTINUE
```

```
C
C        TRY TO REFIT REMOVED TREATMENT
C        OR REMOVE ANOTHER OCCURRENCE OF SAME TREATMENT
C
      IF (KREP .EQ. IR(IL)) GOTO 18
      N = ND
      GOTO 12
      END
```

```
      SUBROUTINE LSTSQ(X, S, A, B, IFLAG, N, K, NWORK, ITMAX, EPS, EPS2,
     *  W, XHAT, XKT, ITER, SUPDIF, IFAULT)
      DIMENSION X(N), S(N, N), A(K, N), B(K), IFLAG(K), W(NWORK),
     *  XHAT(N), XKT(K)
C
C        ALGORITHM AS225 APPL. STATIST. (1987) VOL. 36, NO. 2
C
C        COMPUTES THE LEAST-SQUARES PROJECTION OF X ONTO
C        THE INTERSECTION OF K SIMULTANEOUS AFFINE CONSTRAINTS,
C        USING THE MAHALANOBIS DISTANCE DETERMINED BY S.
C        THE ITH CONSTRAINT IS OF THE FORM
C        SUM OVER J OF A(I,J)*X(J) (.LE.,.EQ.) B(I)
C        --.LE. IF IFLAG(I) .EQ. 0
C        --.EQ. IF IFLAG(I) .EQ. 1
C
      DATA ZERO, TWO /0.0E0, 2.0E0/
C
      IFAULT = 0
      NK = N * K
      NK2 = NK + NK
      NK2K = NK2 + K
C
C        CHECK INPUTS.
C
      IF (N .LE. 0 .OR. K .LE. 0 .OR. ITMAX .LE. 0 .OR. EPS .LE.
     *  ZERO .OR. NWORK .LE. 0 .OR. EPS2 .LE. ZERO) GOTO 300
      IF (NWORK .LT. NK2K + N) GOTO 320
      DO 10 I = 1, K
      IF (IFLAG(I) .EQ. 0 .OR. IFLAG(I) .EQ. 1) GOTO 10
      GOTO 300
   10 CONTINUE
C
C        INITIALIZE ARRAYS, AND COMPUTE
C        MATRIX PRODUCTS A*S AND DIAG(A*S*A').
C        WORKSPACE STRUCTURE IS AS FOLLOWS:
C        W(1)-W(N*K):  ARRAY OF INCREMENT VECTORS.  ITH INC VECTOR
C        IS (XKT(I)/2)*S*A(I,.).
C        W(NK+1)-W(N*K*2):  MATRIX A*S.
C        W(NK2+1)-W(NK2+K):  VECTOR DIAG(A*S*A').
C        W(NK2K+1)-W(NK2K+N):  PREVIOUS XHAT VECTOR, TO CHECK FOR CONV.
C
      DO 40 I = 1, K
      DO 30 J = 1, N
      INDEX = NK + (J - 1) * K + I
      W(INDEX) = ZERO
      DO 20 L = 1, N
   20 W(INDEX) = W(INDEX) + A(I, L) * S(L, J)
   30 CONTINUE
   40 CONTINUE
      DO 60 I = 1, K
      INDEX = NK2 + I
      W(INDEX) = ZERO
      DO 50 J = 1, N
      IND2 = NK + (J - 1) * K + I
      W(INDEX) = W(INDEX) + A(I, J) * W(IND2)
   50 CONTINUE
      IF (ABS(W(INDEX)) .LE. EPS2) GOTO 310
   60 CONTINUE
      DO 80 J = 1, N
      INDEX = NK2K + J
      W(INDEX) = X(J)
```

```
      XHAT(J) = X(J)
      DO 70 I = 1, K
      INDEX = (I - 1) * N + J
      W(INDEX) = ZERO
   70 CONTINUE
   80 CONTINUE
      ITER = 0
C
C        THE ITERATION LOOP:  LINES 100-200.
C
  100 ITER = ITER + 1
      SUPDIF = ZERO
      DO 200 I = 1, K
C
C        REMOVE OLD INCREMENT VECTOR:
C
      DO 110 J = 1, N
      INDEX = (I - 1) * N + J
      XHAT(J) = XHAT(J) + W(INDEX)
  110 CONTINUE
C
C        EVALUATE ITH CONSTRAINT:
C
      SUM = ZERO
      DO 120 J = 1, N
  120 SUM = SUM + A(I, J) * XHAT(J)
      SUM = SUM - B(I)
      IF (IFLAG(I) .EQ. 0 .AND. SUM .LE. ZERO) GOTO 140
C
C        COMPUTE NEW INCREMENT VECTOR THAT FORCES EQUALITY IN ITH CONST.
C
      INDEX = NK2 + I
      TEMP = SUM / W(INDEX)
      DO 130 J = 1, N
      IND1 = (I - 1) * N + J
      IND2 = NK + (J - 1) * K + I
      W(IND1) = W(IND2) * TEMP
      XHAT(J) = XHAT(J) - W(IND1)
  130 CONTINUE
      GOTO 160
C
C        IF CONSTRAINT WAS SATISFIED, SET INCREMENT TO ZERO:
C
  140 DO 150 J = 1, N
      INDEX = (I - 1) * N + J
      W(INDEX) = ZERO
  150 CONTINUE
C
C        FIND LARGEST CHANGE, AND CHECK FOR CONVERGENCE:
C
  160 DO 170 J = 1, N
      INDEX = NK2K + J
      ABDIF = ABS(XHAT(J) - W(INDEX))
      IF (SUPDIF .LT. ABDIF) SUPDIF = ABDIF
  170 CONTINUE
  200 CONTINUE
      IF (SUPDIF .LE. EPS) GOTO 400
      DO 210 J = 1, N
      INDEX = NK2K + J
      W(INDEX) = XHAT(J)
  210 CONTINUE
```

```
      IF (ITER .LT. ITMAX) GOTO 100
      IFAULT = 1
      RETURN
  300 IFAULT = 2
      RETURN
  310 IFAULT = 3
      RETURN
  320 IFAULT = 4
      RETURN
C
C        COMPUTE KUHN-TUCKER COEFFICIENTS AND RETURN.
C
  400 DO 430 I = 1, K
C
C        FIND A NON-ZERO DENOMINATOR:
C
      DO 410 J = 1, N
      INDEX = NK + (J - 1) * K + I
      IF (ABS(W(INDEX)) .GT. EPS2) GOTO 420
  410 CONTINUE
  420 IND2 = (I - 1) * N + J
      XKT(I) = TWO * W(IND2) / W(INDEX)
  430 CONTINUE
      RETURN
      END
```

```
      REAL FUNCTION BETANC(X, A, B, LAMBDA, IFAULT)
C
C     ALGORITHM AS226 APPL. STATIST. (1987) VOL. 36, NO. 2
C     Incorporates modification AS R84 from AS vol. 39, pp311-2, 1990
C
C     Returns the cumulative probability of X for the non-central beta
C     distribution with parameters A, B and non-centrality LAMBDA
C
C     Auxiliary routines required: ALOGAM - log-gamma function (ACM
C     291 or AS 245), and BETAIN - incomplete-beta function (AS 63)
C
      REAL A, AX, B, BETA, C, ERRBD, ERRMAX, GX, HALF, LAMBDA, ONE, Q,
     *     SUMQ, TEMP, X, XJ, ZERO
      REAL A0, X0, UALPHA
C
C     Change ERRMAX and ITRMAX if desired ...
C
      DATA ERRMAX, ITRMAX /1.0E-6, 100/, UALPHA /5.0/
C
      DATA ZERO, HALF, ONE /0.0, 0.5, 1.0/
C
      BETANC = X
C
      IFAULT = 2
      IF (LAMBDA .LT. ZERO .OR. A .LE. ZERO .OR. B .LE. ZERO) RETURN
      IFAULT = 3
      IF (X .LT. ZERO .OR. X .GT. ONE) RETURN
      IFAULT = 0
      IF (X .EQ. ZERO .OR. X .EQ. ONE) RETURN
C
      C = LAMBDA * HALF
C
C     Initialize the series ...
C
      X0 = INT(MAX(C - UALPHA*SQRT(C), ZERO)
      A0 = A + X0
      BETA = ALOGAM(A0, IFAULT) + ALOGAM(B, IFAULT) -
     *       ALOGAM(A0+B, IFAULT)
      TEMP = BETAIN(X, A0, B, BETA, IFAULT)
      GX = EXP(A0 * LOG(X) + B * LOG(ONE - X) - BETA - LOG(A0))
      IF (A0 .GT. A) THEN
        Q = EXP(-C + X0*LOG(C) - ALOGAM(X0 + ONE, IFAULT)
      ELSE
        Q = EXP(-C)
      END IF
      XJ = ZERO
      AX = Q * TEMP
      SUMQ = ONE - Q
      BETANC = AX
C
C     Recur over subsequent terms until convergence is achieved...
C
   10 XJ = XJ + ONE
      TEMP = TEMP - GX
      GX = X * (A + B + XJ - ONE) * GX / (A + XJ)
      Q = Q * C / XJ
      SUMQ = SUMQ - Q
      AX = TEMP * Q
      BETANC = BETANC + AX
C
C     Check for convergence and act accordingly...
```

```
C
      ERRBD = (TEMP - GX) * SUMQ
      IF ((INT(XJ) .LT. ITRMAX) .AND. (ERRBD .GT. ERRMAX)) GO TO 10
      IF (ERRBD .GT. ERRMAX) IFAULT = 1
C
      RETURN
      END
```

```
      SUBROUTINE GCOUNT(N, APPLY, IFAULT)
C
C     ALGORITHM AS227  APPL. STATIST. (1987) VOL. 36, NO. 2
C
C     Generates all possible N-bit binary codes, and applies a users
C     procedure for each code generated.
C     N must be <= NMAX, which is set in the PARAMETER statement below.
C     IFAULT = 2 is returned otherwise.
C
C     Translated from Algol 60.
C
      INTEGER N, IFAULT
      EXTERNAL APPLY
C
C     Local variables
C
      INTEGER NMAX
      PARAMETER (NMAX = 100)
      INTEGER CHANGE, I, TPOINT(NMAX)
      LOGICAL STATUS(NMAX)
C
      IF (N .LT. 1) THEN
        IFAULT = 1
        RETURN
      END IF
      IF (N .GT. NMAX) THEN
        IFAULT = 2
        RETURN
      END IF
      IFAULT = 0
C
C     Initialize and make first call to user's routine.
C
      DO 10 I = 1, N
        STATUS(I) = .FALSE.
        TPOINT(I) = I + 1
   10 CONTINUE
      CALL APPLY(N, N, STATUS)
C
C     Generate a new code.   The user's routine is called twice each
C     cycle; the first time the bit which changes is bit 1.
C
   20 IF (STATUS(1)) THEN
        STATUS(1) = .FALSE.
        CHANGE = TPOINT(2)
      ELSE
        STATUS(1) = .TRUE.
        CHANGE = 2
      END IF
      CALL APPLY(N, 1, STATUS)
C
C     Check if count exhausted.
C
      IF (CHANGE .GT. N) RETURN
C
      IF (STATUS(CHANGE)) THEN
        STATUS(CHANGE) = .FALSE.
        TPOINT(CHANGE) = TPOINT(CHANGE + 1)
      ELSE
        STATUS(CHANGE) = .TRUE.
        TPOINT(CHANGE) = CHANGE + 1
```

```
          END IF
          CALL APPLY(N, CHANGE, STATUS)
   C
          GO TO 20
          END
```

```
      SUBROUTINE IPROJ(R, C, B, L, K, EPS1, EPS2, ITMAX1, ITMAX2, NWORK,
     *  W, P, NIT, SUPDIF, DIV, IFAULT)
C
C        ALGORITHM AS228 APPL. STATIST. (1987) VOL. 36, NO. 3
C
      DIMENSION R(L), P(L), C(L, K), B(K), W(NWORK)
C
C        COMPUTES THE I-PROJECTION OF THE VECTOR R
C        ONTO THE INTERSECTION OF K LINEAR INEQUALITY
C        CONSTRAINTS.  THE J-TH CONSTRAINT IS OF THE FORM
C        SUM OVER I OF C(I,J)*P(I) .LE. B(J)
C        THE DATA, R, IS A NON-NEGATIVE VECTOR.
C        THE I-PROJ IS RETURNED IN P, AS A VECTOR OF PROBABILITIES.
C
      DATA ZERO, ZONE /0.0E0, 1.0E0/
      NIT = 0
      TOT = ZERO
      IFAULT = 0
      LK = L * K
      LKL1 = LK + L
      LKL2 = LKL1 + L
      LKL3 = LKL2 + L
      IF (NWORK .LT. LKL3 + K) GOTO 500
C
C        CHECK DATA FOR NEGATIVES, STANDARDIZE DATA (TO SUM TO 1),
C        INITIALIZE INCREMENT ARRAY AND DUAL ESTIMATE ARRAY.
C
      DO 5 J = 1, K
      INDEX = LKL3 + J
      W(INDEX) = ZERO
    5 CONTINUE
      DO 20 I = 1, L
      P(I) = R(I)
      DO 10 J = 1, K
      INDEX = (J - 1) * L + I
      W(INDEX) = ZONE
   10 CONTINUE
      IF (P(I) .LT. ZERO) GOTO 300
      TOT = TOT + P(I)
   20 CONTINUE
      IF (TOT .LE. ZERO) GOTO 300
      DO 30 I = 1, L
   30 P(I) = P(I) / TOT
C
C        CHECK DATA FOR NON-EXISTENCE OF A SOLUTION, AND LOCATE
C        AND INSERT FORCED ZEROS.
C
   35 IFLAG = 0
      DO 50 J = 1, K
      SNEG = ZERO
      SPOS = ZERO
      SZERO = ZERO
      DO 40 I = 1, L
      CMB = C(I, J) - B(J)
      IF (CMB .GT. ZERO) SPOS = SPOS + P(I)
      IF (CMB .LT. ZERO) SNEG = SNEG + P(I)
      IF (CMB .EQ. ZERO) SZERO = SZERO + P(I)
   40 CONTINUE
      IF (SNEG .GT. ZERO) GOTO 50
      IF (SZERO .LE. ZERO) GOTO 400
      IF (SPOS .LE. ZERO) GOTO 50
```

```
      DO 45 I = 1, L
      IF (C(I, J) - B(J) .GT. ZERO) P(I) = ZERO
      P(I) = P(I) / SZERO
   45 CONTINUE
      IFLAG = 1
   50 CONTINUE
      IF (IFLAG .EQ. 1) GOTO 35
C
C        PERFORM AN ITERATION:  LINES 100 TO 200
C
  100 SUPDIF = ZERO
      DO 200 J = 1, K
      SUM = ZERO
      DO 110 I = 1, L
      INDEX = LK + I
      W(INDEX) = P(I)
      INDEX = (J - 1) * L + I
      P(I) = P(I) / W(INDEX)
      INDEX = LKL1 + I
      W(INDEX) = C(I, J) - B(J)
      SUM = SUM + P(I)
  110 CONTINUE
      DO 115 I = 1, L
  115 P(I) = P(I) / SUM
C
C        P PASSES DATA TO PROJ, AND RETURNS THE RESULT.
C        W(LK+I) STORES THE PREVIOUS RESULT, FOR COMPARISON.
C        W(LKL1+I) PASSES THE J-TH COLUMN OF THE CONSTRAINT
C        MATRIX TO PROJ.
C        W(LKL2+I) IS WORKSPACE FOR PROJ.
C        W(LKL3+J) IS THE PREVIOUS DUAL ESTIMATE.
C
      INDEX = LKL3 + J
      CALL PROJ(P, W, NWORK, L, LKL1, LKL2, J, W(INDEX), EPS2, ITMAX2,
     *  IFAULT)
      IF (IFAULT .EQ. 2) RETURN
C
C        UPDATE THE INCREMENT ARRAY W(1)-W(LK) AND GET THE GREATEST
C        DIFFERENCE (TO TEST FOR CONVERGENCE).
C
      DO 120 I = 1, L
      IS = (J - 1) * L + I
      INDEX = LK + I
      IF (W(INDEX) .GT. ZERO) W(IS) = W(IS) * P(I) / W(INDEX)
      DIFF = ABS(P(I) - W(INDEX))
      IF (DIFF .GT. SUPDIF) SUPDIF = DIFF
  120 CONTINUE
  200 CONTINUE
C
C        COUNT ITERATIONS, AND CHECK FOR CONVERGENCE.
C
      NIT = NIT + 1
      IF (SUPDIF .GT. EPS1) GOTO 220
C
C        COMPUTE I-DIVERGENCE
C
      DIV = ZERO
      DO 210 I = 1, L
      IF (P(I) .GT. ZERO) DIV = DIV + P(I) * ALOG(P(I) * TOT / R(I))
  210 CONTINUE
      RETURN
```

```
  220 IF (NIT .LE. ITMAX1) GOTO 100
      IFAULT = 1
      RETURN
  300 IFAULT = 3
      RETURN
  400 IFAULT = 4
      RETURN
  500 IFAULT = 5
      RETURN
      END
C
      SUBROUTINE PROJ(P, W, NWORK, L, LKL1, LKL2, J, ZLAST, EPS2,
     *  ITMAX2, IFAULT)
C
C        ALGORITHM AS228 APPL. STATIST. (1987) VOL. 36, NO. 3
C
      DIMENSION P(L), W(NWORK)
      DATA ZERO /0.0E0/
C
C        THIS SUBROUTINE FINDS THE I-PROJ OF P SUBJECT TO THE LINEAR
C        CONSTRAINT IN W -- OF THE FORM
C        SUM OVER I OF W(LKL1+I)*P(I) .LE. ZERO.
C        THE FENCHEL DUAL SOLUTION IS OBTAINED BY NEWTON'S METHOD,
C        TO WITHIN EPS2, WHICH IS THEN USED TO CONSTRUCT THE I-PROJ.
C
      ITER = 0
      IFAULT = 0
      VAL = ZERO
      DERIV = ZERO
      DO 10 I = 1, L
      IND1 = LKL1 + I
      IND2 = LKL2 + I
      W(IND2) = P(I) * W(IND1)
      VAL = VAL + W(IND2)
      DERIV = DERIV - W(IND1) * W(IND2)
   10 CONTINUE
      IF (VAL .GT. ZERO) GOTO 100
C
C        IF VAL .LE. 0 THEN CONSTRAINT IS SATISFIED; RETURN.
C
      ZLAST = ZERO
      RETURN
C
C        THE ITERATION BLOCK:  LINES 100 TO 110.
C
  100 Z = ZLAST - VAL / DERIV
      IF (ABS(Z - ZLAST) .LE. EPS2) GOTO 200
      IF (ITER .GT. ITMAX2) GOTO 300
      ZLAST = Z
      VAL = ZERO
      DERIV = ZERO
      ITER = ITER + 1
      DO 110 I = 1, L
      IND1 = LKL1 + I
      IND2 = LKL2 + I
      TEMP = EXP(-Z * W(IND1))
      VAL = VAL + W(IND2) * TEMP
      DERIV = DERIV - W(IND1) * W(IND2) * TEMP
  110 CONTINUE
      GOTO 100
C
```

```
C        GENERATE PROJECTION, STANDARDIZE AND RETURN.
C
  200 SUM = ZERO
      DO 210 I = 1, L
      INDEX = LKL1 + I
      P(I) = P(I) * EXP(-Z * W(INDEX))
      SUM = SUM + P(I)
  210 CONTINUE
      DO 220 I = 1, L
  220 P(I) = P(I) / SUM
      RETURN
  300 IFAULT = 2
      RETURN
      END
```

```
      subroutine rq(m,n,m5,n2,a,b,t,toler,ift,x,e,s,wa,wb,
     1  nsol,sol,lsol)
      implicit double precision (a-h,o-z)
c
c     Algorithm AS229 Appl. Statist. (1987) Vol. 36, No. 3
c
      integer i,j,k,kl,kount,kr,l,lsol,m,m1,m2,m3,m4,m5,
     1  ift
      integer n,n1,n2,nsol,out,s(m)
      logical stage,test,init,iend
      double precision a1,aux,b1,big,d,dif,pivot,smax,t,t0,t1,tnt
      double precision min,max,toler,zero,half,one,two,three,four,five
      double precision b(m),sol(n2,nsol),a(m,n),x(n),wa(m5,n2),wb(m)
      double precision sum,e(m)
      double precision xx,y
      data big /1.0d37/
      data zero /0.0d0/
      data half /0.5d0/
      data one /1.0d0/
      data two /2.0d0/
      data three /3.0d0/
      data four /4.0d0/
      data five /5.0d0/
c
c     check dimension parameters
c
      ift = 0
      if (m5.ne.m + 5) ift = 3
      if (n2.ne.n + 2) ift = 4
      if (m.le.zero.or.n.le.zero) ift = 5
      if (ift.gt.two) return
c
c     initialization
c
      m1 = m + 1
      n1 = n + 1
      m2 = m + 2
      m3 = m + 3
      m4 = m + 4
      do 2 i = 1,m
        wb(i) = b(i)
        do 1 j = 1,n
          wa(i,j) = a(i,j)
    1   continue
    2 continue
      wa(m2,n1) = zero
      dif = zero
      iend = .true.
      if (t.ge.zero.and.t.le.one) goto 3
      t0 = one / float(m) - toler
      t1 = one - t0
      t = t0
      iend = .false.
    3 continue
      init = .false.
      lsol = 1
      kount = 0
      do 9 k = 1,n
        wa(m5,k) = zero
        do 8 i = 1,m
          wa(m5,k) = wa(m5,k) + wa(i,k)
```

```
 8    continue
      wa(m5,k) = wa(m5,k) / float(m)
 9  continue
      do 10 j = 1,n
        wa(m4,j) = j
        x(j) = zero
10  continue
      do 40 i = 1,m
        wa(i,n2) = n + i
        wa(i,n1) = wb(i)
        if (wb(i).ge.zero) goto 30
        do 20 j = 1,n2
          wa(i,j) = -wa(i,j)
20    continue
30    e(i) = zero
40  continue
      do 42 j = 1,n
        wa(m2,j) = zero
        wa(m3,j) = zero
        do 41 i = 1,m
          aux = sign(one,wa(m4,j)) * wa(i,j)
          wa(m2,j) = wa(m2,j) + aux * (one - sign(one,wa(i,n2)))
          wa(m3,j) = wa(m3,j) + aux * sign(one,wa(i,n2))
41    continue
        wa(m3,j) = two * wa(m3,j)
42  continue
      goto 48
43  continue
      lsol = lsol + 1
      do 44 i = 1,m
        s(i) = zero
44  continue
      do 45 j = 1,n
        x(j) = zero
45  continue
c
c     compute next t
c
      smax = two
      do 47 j = 1,n
        b1 = wa(m3,j)
        a1 = (-two - wa(m2,j)) / b1
        b1 = -wa(m2,j) / b1
        if (a1.lt.t) goto 46
        if (a1.ge.smax) goto 46
        smax = a1
        dif = (b1 - a1) / two
46    if (b1.le.t) goto 47
        if (b1.ge.smax) goto 47
        smax = b1
        dif = (b1 - a1) / two
47  continue
      tnt = smax + toler * (one + dabs(dif))
      if (tnt.ge.t1 + toler) iend = .true.
      t = tnt
      if (iend) t = t1
48  continue
c
c     compute new marginal costs
c
      do 49 j = 1,n
```

```
          wa(m1,j) = wa(m2,j) + wa(m3,j) * t
   49   continue
        if (init) goto 265
c
c     stage 1
c
c      determine the vector to enter the basis
        stage = .true.
        kr = 1
        kl = 1
   70   max = -one
        do 80 j = kr,n
          if (abs(wa(m4,j)).gt.n) goto 80
          d = abs(wa(m1,j))
          if (d.le.max) goto 80
          max = d
          in = j
   80   continue
        if (wa(m1,in).ge.zero) goto 100
        do 90 i = 1,m4
          wa(i,in) = -wa(i,in)
   90   continue
c
c      determine the vector to leave the basis
c
  100   k = 0
        do 110 i = kl,m
          d = wa(i,in)
          if (d.le.toler) goto 110
          k = k + 1
          wb(k) = wa(i,n1) / d
          s(k) = i
          test = .true.
  110   continue
  120   if (k.gt.0) goto 130
        test = .false.
        goto 150
  130   min = big
        do 140 i = 1,k
          if (wb(i).ge.min) goto 140
          j = i
          min = wb(i)
          out = s(i)
  140   continue
        wb(j) = wb(k)
        s(j) = s(k)
        k = k - 1
c
c     check for linear dependence in stage 1
c
  150   if (test.or. .not.stage) goto 170
        do 160 i = 1,m4
          d = wa(i,kr)
          wa(i,kr) = wa(i,in)
          wa(i,in) = d
  160   continue
        kr = kr + 1
        goto 260
  170   if (test) goto 180
        wa(m2,n1) = two
        goto 390
```

```
 180  pivot = wa(out,in)
      if (wa(m1,in) - pivot - pivot .le. toler) goto 200
      do 190 j = kr,n1
        d = wa(out,j)
        wa(m1,j) = wa(m1,j) - d - d
        wa(m2,j) = wa(m2,j) - d - d
        wa(out,j) = -d
 190  continue
      wa(out,n2) = -wa(out,n2)
      goto 120
c
c      pivot on wa(out,in)
c
 200  do 210 j = kr,n1
        if (j.eq.in) goto 210
        wa(out,j) = wa(out,j) / pivot
 210  continue
      do 230 i = 1,m3
        if (i.eq.out) goto 230
        d = wa(i,in)
        do 220 j = kr,n1
          if (j.eq.in) goto 220
          wa(i,j) = wa(i,j) - d * wa(out,j)
 220    continue
 230  continue
      do 240 i = 1,m3
        if (i.eq.out) goto 240
        wa(i,in) = -wa(i,in) / pivot
 240  continue
      wa(out,in) = one / pivot
      d = wa(out,n2)
      wa(out,n2) = wa(m4,in)
      wa(m4,in) = d
      kount = kount + 1
      if (.not.stage) goto 270
c
c       interchange rows in stage 1
c
      kl = kl + 1
      do 250 j = kr,n2
        d = wa(out,j)
        wa(out,j) = wa(kount,j)
        wa(kount,j) = d
 250  continue
 260  if (kount + kr.ne.n1) goto 70
c
c       stage 2
c
 265  stage = .false.
c
c      determine the vector to enter the basis
c
 270  max = -big
      do 290 j = kr,n
        d = wa(m1,j)
        if (d.ge.zero) goto 280
        if (d.gt. (-two)) goto 290
        d = -d - two
 280  if (d.le.max) goto 290
      max = d
      in = j
```

```
 290  continue
      if (max.le.toler) goto 310
      if (wa(m1,in) .gt.zero) goto 100
      do 300 i = 1,m4
        wa(i,in) = -wa(i,in)
 300  continue
      wa(m1,in) = wa(m1,in) - two
      wa(m2,in) = wa(m2,in) - two
      goto 100
c
c       compute quantiles
 310  continue
      do 320 i = 1,kl - 1
        k = wa(i,n2) * sign(one,wa(i,n2))
        x(k) = wa(i,n1) * sign(one,wa(i,n2))
 320  continue
      sum = zero
      do 330 i = 1,n
        sum = sum + x(i) * wa(m5,i)
        sol(i + 2,lsol) = x(i)
 330  continue
      sol(1,lsol) = t
      sol(2,lsol) = sum
      if (iend) goto 340
      init = .true.
      goto 43
 340  continue
      if (lsol.le.2) goto 355
      do 350 i = 2,lsol
        sol(1,i-1) = sol(1,i)
 350  continue
      lsol = lsol - 1
      sol(1,lsol) = one
 355  continue
      l = kl - 1
      do 370 i = 1,l
        if (wa(i,n1).ge.zero) goto 370
        do 360 j = kr,n2
          wa(i,j) = -wa(i,j)
 360    continue
 370  continue
      wa(m2,n1) = zero
      if (kr.ne.1) goto 390
      do 380 j = 1,n
        d = abs(wa(m1,j))
        if (d.le.toler .or. two - d .le. toler) goto 390
 380  continue
      wa(m2,n1) = one
 390  do 400 i = kl,m
        k = wa(i,n2) * sign(one,wa(i,n2))
        d = wa(i,n1) * sign(one,wa(i,n2))
        k = k - n
        e(k) = d
 400  continue
      wa(m2,n2) = kount
      wa(m1,n2) = n1 - kr
      sum = zero
      do 410 i = 1,m
        sum = sum + e(i) * (half + sign(one,e(i)) * (t - half))
 410  continue
      wa(m1,n1) = sum
```

```
        return
        end
```

```
      SUBROUTINE QDISX(A, B, M, T, MAX, CZERO, MAX1, P0, P, Y, C0, C,
     +          MEAN, SD, SUM, IFAULT)
C
C     ALGORITHM AS230  APPL. STATIST. (1986) VOL. 36, NO. 3
C
C     Tabulates approximate distribution of customers in multi-server
C     queues having random arrivals and Erlangian service times.
C     Hokstad's method is used.
C
      REAL P(MAX), Y(MAX), C(MAX)
      REAL A, B, C0, CZERO, FI, MEAN, ONE, PM1, PNNN, P0, R, R1, SD,
     +          SUM, TERM, FT, TWO, ZERO, WT
      INTEGER T
      DATA ZERO/0.0/, ONE/1.0/, TWO/2.0/, PNNN/0.999/
C
      IFAULT = 0
      FT = FLOAT(T)
      MAX0 = MAX
      IF (MAX .GT. 4000) MAX0 = 4000
      DO 5 I = 1, MAX0
        P(I) = ZERO
        Y(I) = ZERO
        C(I) = ZERO
    5 CONTINUE
C
C     Calculate P0 to P(M-1)
C
      R = A * B / FLOAT(M)
      IF (R .GT. PNNN) GO TO 10
      P0 = ONE - R
      PM1 = P0
      IF (M .EQ. 1) GO TO 15
      P0 = ONE
      TERM = ONE
      M1 = M - 1
      DO 20 I = 1, M1
        TERM = TERM * A * B / FLOAT(I)
        P0 = P0 + TERM
        P(I) = TERM
   20 CONTINUE
      TERM = TERM * A * B / ((ONE - R) * FLOAT(M))
      P0 = ONE / (P0 + TERM)
      DO 25 I = 1, M1
   25 P(I) = P(I) * P0
      PM1 = P(M1)
   15 IF (T .GT. 100) GO TO 30
C
C     Calculate Y-values for Erlangian service
C
      C0 = (ONE + R / FT) ** FT
      P(M) = (C0 - ONE) * PM1
      Y(1) = (ONE + R / FT) ** FT - R / (ONE + R / FT)
      Y(2) = -(FT * (FT + ONE) / TWO) / (FT / R + ONE) ** 2
      DO 35 I = 3, MAX0
   35 Y(I) = ZERO
      DO 40 I = 3, MAX0
        FI = FLOAT(I)
        IF (ABS(Y(I-1)) .LT. CZERO) GO TO 45
        Y(I) = (Y(I-1) * (FT + FI - ONE)) / (FI * (FT / R + ONE))
   40 CONTINUE
      GO TO 45
```

```
C
C     Calculate Y-values for constant service times
C
   30 C0 = EXP(R)
      P(M) = (C0 - ONE) * PM1
      Y(1) = C0 - R
      Y(2) = -R * R / TWO
      DO 50 I = 3, MAX0
        Y(I) = ZERO
   50 CONTINUE
      DO 55 I = 3, MAX0
        IF (ABS(Y(I-1)) .LT. CZERO) GO TO 45
        Y(I) = Y(I-1) * R / FLOAT(I)
   55 CONTINUE
C
C     Calculate P(M+1) to P(MAX0)
C
   45 R1 = ONE / (ONE - R)
      C(1) = C0 * Y(1)
      DO 60 I = 2, MAX0
        C(I) = ZERO
        I1 = I - 1
        DO 65 J = 1, I1
          C(I) = C(I) + C(I-J) * Y(J)
   65   CONTINUE
        C(I) = C(I) + C0 * Y(I)
        IF (C(I) .GE. R1 .AND. C(I-1) .GE. R1) GO TO 95
   60 CONTINUE
      I = MAX0
   95 MAX1 = I
      M2 = MAX1 - M
      P(M+1) = (C(1) - C0) * PM1
      DO 70 I = 2, M2
   70 P(M+I) = ZERO
      DO 75 I = 2, M2
        IF (ABS(P(M+I-1)) .LT. CZERO .AND. P(M+I-1) .LT. P(M+I-2))
     +          GO TO 80
        P(M+I) = (C(I) - C(I-1)) * PM1
        IF (P(M+I) .GE. ZERO) GO TO 75
        P(M+I) = ZERO
        GO TO 80
   75 CONTINUE
C
C     Calculate mean, standard deviation and sum of terms.
C     The algorithm for calculating the mean & standard deviation used
C     here is better than that in the journal.
C
   80 SUM = P0
      MEAN = ZERO
      SD = ZERO
      DO 90 I = 1, MAX1
        FI = FLOAT(I)
        SUM = SUM + P(I)
        WT = P(I) / SUM
        TERM = FI - MEAN
        MEAN = MEAN + WT * TERM
        SD = SD + P(I) * TERM * (FI - MEAN)
   90 CONTINUE
      SD = SQRT(SD / SUM)
C
      RETURN
```

```
C
   10 IFAULT = -1
      RETURN
      END
```

```
FUNCTION chisqnc(idf: nonnegint; nc, arg, eps: real; VAR density: real;
    VAR ifault: nonnegint): real;

{Algorithm AS 231  Appl. Statist. (1987) Vol.36, No.3}

{chisqnc evaluates the probability that a noncentral chi-squared
    random variable is less than a given value, arg}

    CONST tol = -50;
          maxit = 500;
          lnrtpi2 = 0.22579135264473;

    VAR i, k: nonnegint;
        m: 0 .. maxit;
        ao, am, b1, eps2, hold, sum, dans, lans, pans, probability: real;
        converged: Boolean;

    PROCEDURE update(VAR j: nonnegint);
        BEGIN
        IF lans < tol THEN
            BEGIN
            lans := lans + ln(arg / j); dans := exp(lans)
            END
        ELSE dans := dans * arg / j;
        pans := pans - dans; j := j + 2
        END; {update}

    BEGIN
    IF (nc < 0.0) OR (arg < 0.0) OR (eps <= 0.0) THEN
        BEGIN
        chisqnc := -1.0; density := -1.0;
        ifault := 1
        END
    ELSE IF (arg = 0.0) AND (idf > 0) THEN
        BEGIN
        chisqnc := 0.0; density := 0.0;
        ifault := 0
        END
    ELSE
        BEGIN

        {Determine initial values}
        k := idf; b1 := 0.5 * nc;
        ao := exp(-b1); eps2 := eps / ao;
        IF odd(k) THEN
            BEGIN
            i := 1; lans := -0.5 * (arg + ln(arg)) - lnrtpi2;
            dans := exp(lans); pans := centnorm(sqrt(arg))
            END
        ELSE
            BEGIN
            i := 2; lans := -0.5 * arg;
            dans := exp(lans); pans := 1.0 - dans
            END;

        {Evaluate first term}
        IF k = 0 THEN
            BEGIN
            m := 1; k := 2;
            am := b1; sum := 1.0 / ao - 1.0 - am;
            density := am * dans; probability := 1.0 + am * pans
```

```
            END
        ELSE
            BEGIN
            m := 0; k := k - 1;
            am := 1.0; sum := 1.0 / ao - 1.0;
            WHILE i < k DO update(i);
            k := k + 1;
            density := dans; probability := pans
            END;

        {Evaluate successive terms of the expansion}
        REPEAT
            m := m + 1; am := b1 * am / m;
            update(k); sum := sum - am;
            density := density + am * dans; hold := am * pans;
            probability := probability + hold;
            converged := (pans * sum < eps2) AND (hold < eps2)
        UNTIL (m = maxit) OR converged;

        IF converged THEN ifault := 0 ELSE ifault := 2;
        density := 0.5 * ao * density;
        chisqnc := ao * probability
        END
    END; {of chisqnc}
```

```
      SUBROUTINE XCOVAR(K, P, Q, R, LMAX, KKRP1, PHI, THETA, SIGMA,
     *  GAMMA, PSI, C, W, W1, IFAULT)
C
C        ALGORITHM AS232 APPL. STATIST. (1988) VOL. 37, NO. 1
C
      INTEGER P, Q, R, S, ZROW, SMI, RP1, RML
      LOGICAL POS, NEG, PZERO
      DIMENSION PHI(K, K, 0:R), THETA(K, K, 0:R), PSI(K, K, 0:LMAX),
     *  GAMMA(K, K, 0:LMAX), C(KKRP1, KKRP1), SIGMA(K, K), W(K, K),
     *  W1(K, K)
C
C        THE NEXT LINE IS THE STATEMENT FUNCTION GIVING THE MAPPING
C        USED FOR THE NUMBERING OF THE GAMMA(I,J,L) ELEMENTS IN THE
C        S.L.E. SYSTEM.  C*Z=B  (SEE EQUATIONS (1A) IN THE DESCRIPTION).
C        NOTE THAT IT TAKES THEM IN THE STORAGE ORDER FOR 3-D ARRAYS.
C
      ZROW(I, J, L, K) = (L * K + J - 1) * K + I
C
      IFAULT = 0
      IF (R .EQ. MAX0(P, Q)) GOTO 10
      IFAULT = -1
      RETURN
   10 RP1 = R + 1
      IF (KKRP1 .EQ. K * K * RP1) GOTO 20
      IFAULT = -2
      RETURN
   20 PZERO = P .EQ. 0
C
C        ZEROISE PLANES OF PHI,THETA BEYOND THE INPUT PLANES (IF ANY).
C
      IF (P .EQ. Q) GOTO 30
C
C        NONE IN THIS CASE.
C
      IF (R .EQ. Q) CALL LINZER(PHI(1, 1, P + 1), K * K * (R - P))
      IF (R .EQ. P) CALL LINZER(THETA(1, 1, Q + 1), K * K * (R - Q))
C
C        NEXT, FORM THE PSI-MATRICES, UP TO THE LMAX'TH.
C        NOTE THAT ONLY THE FIRST R OF THESE ARE NEEDED FOR THE CROSS-
C        CORRELATIONS.
C
   30 CALL SCALAR(PSI(1, 1, 0), K, 1.0)
C
C        PSI(0) = IDENTITY
C
      DO 50 S = 1, LMAX
C
C        FORM PSI(S)
C
      IF (S .LE. R) CALL MATSUB(PSI(1, 1, S), PHI(1, 1, S), THETA(1, 1,
     *  S), K, K)
C
      IF (S .EQ. 1) GOTO 50
C
C        NO SUMMATION IN THIS CASE.
C
      IF (S .GT. R) CALL LINZER(PSI(1, 1, S), K * K)
C
C        ZERO INITIAL TERM
C
      IF (PZERO) GOTO 50
```

```
      IMAX = MIN0(P, S - 1)
C
C        BECAUSE PHI(I)=0 FOR I>P
C
      DO 40 I = 1, IMAX
      SMI = S - I
      CALL MATMUL(W, PHI(1, 1, I), PSI(1, 1, SMI), K, K, K, K)
      CALL MATADD(PSI(1, 1, S), PSI(1, 1, S), W, K, K)
   40 CONTINUE
C
   50 CONTINUE
C
C        NEXT,FORM THE S.L.E. SYSTEM [EQNS. (1A)] FOR THE FIRST (1+R)
C        GAMMA'S. THE RHS CONSTANTS ARE ASSEMBLED IN THE GAMMA ARRAY,
C        BECAUSE THAT IS WHERE THE 'SOLVE' ROUTINE IS REQUIRED TO
C        DEPOSIT THE SOLUTIONS. THE FORMULAE FOR THESE ARE:
C
C        [B(L)](I,J) = [ SUM( PSI(S)*SIGMA*THETA'(S+L) )](I,J)  ,
C
C        WHERE THE SUM IS OVER  S=0,1,....,R-L.   NOTE THAT THE FIRST
C        TERM (S=0) IN THE SUMMATION IS ALWAYS  PSI(0)*SIGMA*THETA'(L),
C        AND WE KNOW THAT  PSI(0)=IDENTITY MATRIX.
C
      DO 70 LP1 = 1, RP1
C
C        COMPUTE B(L)
C
      L = LP1 - 1
      CALL MULTRA(GAMMA(1, 1, L), SIGMA, THETA(1, 1, L), K, K, K, K)
      IF (L .EQ. R) GOTO 70
C
C        ONLY ONE TERM IN THIS CASE
C
      RML = R - L
      DO 60 S = 1, RML
C
C        FORM S'TH TERM AND ADD UP
C
      LPS = L + S
      CALL MULTRA(W, SIGMA, THETA(1, 1, LPS), K, K, K, K)
      CALL MATMUL(W1, PSI(1, 1, S), W, K, K, K, K)
      CALL MATADD(GAMMA(1, 1, L), GAMMA(1, 1, L), W1, K, K)
   60 CONTINUE
   70 CONTINUE
C
C        NOW ASSEMBLE THE COEFFICIENT ARRAY  C. THE MAPPING FUNCTION
C        FOR THE LINEAR NUMBERING OF THE GAMMA(L)(I,J) ELEMENTS IS
C        CHOSEN TO MATCH THE NATURAL STORE ORDERING; HENCE THE POINTER
C        FUNCTION IS:  ZROW(I,J,L,K)=L*K**2 + (J-1)*K + I
C        HOWEVER, IF P=0 THEN  GAMMA(L)=-B(L), AND NO SLE'S NEED BE
C        SOLVED.
C
      IF (P .GT. 0) GOTO 80
      CALL NEGATE(GAMMA, KKRP1)
C
C        LINEARISED VERSION
C
      GOTO 100
C
   80 CALL LINZER(C, KKRP1 * KKRP1)
C
```

```
      DO 90 LP1 = 1, RP1
C
      L = LP1 - 1
C
C        L'TH EQUATION OF SYSTEM (1A)
C
      LMR = L - R
C
      DO 90 I = 1, K
C
C        I'TH ROW
C
      DO 90 J = 1, K
C
C        J'TH COLUMN
C
      IROW = ZROW(I, J, L, K)
C
      DO 90 S = LMR, L
C
C        OF S'TH TERM IN L-H SUMMATION
C
      MS = IABS(S)
      NEG = S .LT. 0
      POS = .NOT.NEG
      LMS = L - S
C
      DO 90 IA = 1, K
C
C        S'TH TERM IS A MATRIX PRODUCT
C

      IF (POS) JCOL = ZROW(I, IA, S, K)
      IF (NEG) JCOL = ZROW(IA, I, MS, K)
C
C        GAMMA(-S)=GAMMA'(S)
C
      C(IROW, JCOL) = PHI(J, IA, LMS)
   90 CONTINUE
C
C        PHI TRANSPOSED
C
C        THIS COMPLETES THE ASSEMBLY OF SLE SYSTEM (1). NOW SOLVE THEM;
C        THE SOLUTION IS OVERWRITTEN ON GAMMA.
C
      CALL SOLVE(KKRP1, .TRUE., C, GAMMA, DDET, GAMMA, IFAULT)
      IF (IFAULT .NE. 0) GOTO 130
C
C        FINALLY, COMPUTE THE REMAINING GAMMA(L) FOR L=R+1,....,LMAX
C
  100 IF (LMAX .LE. R) GOTO 130
C
      CALL LINZER(GAMMA(1, 1, RP1), K * K * (LMAX - R))
C
C        ZERO REMAINING
C
      IF (PZERO) GOTO 130
C
      DO 120 L = RP1, LMAX
      DO 110 S = 1, P
C
```

```
C        COMPUTE S'TH PRODUCT OF SUM
C
      LMS = L - S
      CALL MULTRA(W, GAMMA(1, 1, LMS), PHI(1, 1, S), K, K, K, K)
      CALL MATADD(GAMMA(1, 1, L), GAMMA(1, 1, L), W, K, K)
  110 CONTINUE
  120 CONTINUE
C
  130 RETURN
      END

      SUBROUTINE CORREL(K, LMAX, G, RHO, W)
C
C        ALGORITHM AS232.1 APPL. STATIST. (1988) VOL. 37, NO. 1
C
      REAL WI, DENOM
      DIMENSION G(K, K, 0:LMAX), RHO(K, K, 0:LMAX), W(K)
C
C        W IS A SCRATCH ARRAY;  G(1:K,1:K,L) HOLDS THE GAMMA(L)
C        MATRICES. FIRST SET UP THE VALUES  SQRT(G(I,I,0) IN W(I)
C
      DO 10 I = 1, K
   10 W(I) = SQRT(G(I, I, 0))
C
C        NOW DIVIDE THE G(I,J,L) BY THE W(I)*W(J)
C
      LM1 = LMAX + 1
      DO 20 I = 1, K
      WI = W(I)
      DO 20 J = 1, K
C
      DENOM = WI * W(J)
C
      DO 20 LP1 = 1, LM1
      L = LP1 - 1
      RHO(I, J, L) = G(I, J, L) / DENOM
   20 CONTINUE
C
      RETURN
      END

      SUBROUTINE PARTLS(K, PSTAMX, KKLMAX, NOOPUT, GAMMA, PHISTA, C,
     *  IFAULT)
C
C        ALGORITHM AS232.2 APPL. STATIST. (1988) VOL. 37, NO. 1
C
      INTEGER PSTAR, PSTAMX
      DIMENSION GAMMA(K, K, 0:PSTAMX), PHISTA(K, K, 0:PSTAMX), C(KKLMAX,
     *  KKLMAX)
C
C        TO COMPUTE THE PARTIAL CORRELATION MATRICES FOR
C        PSTAR=1,2,..,PSTAMX. THESE ARE COMPUTED IN REVERSE ORDER SO
C        THAT AT THE END THE 3-D ARRAY WILL KEEP THEM ALL IN ORDER
C        (I.E.  PHISTA( , ,L) CONTAINS THE L'TH).
C
      PSTAR = PSTAMX
   10 KKPST = K * K * PSTAR
C
C        ORDER OF SYSTEM (2) FOR THIS PSTAR
C
      CALL PUREAR(K, PSTAR, KKPST, GAMMA, PHISTA, C, IFAULT)
```

```
      IF (IFAULT .NE. 0) RETURN
C
      PSTAR = PSTAR - 1
      IF (PSTAR .GE. 10) GOTO 10
C
      RETURN
      END

      SUBROUTINE PUREAR(K, PSTAR, KKPST, GAMMA, PHISTA, C, IFAULT)
C
C        ALGORITHM AS232.3 APPL. STATIST. (1988) VOL. 37, NO. 1
C
      INTEGER PSTAR, S, ZROW
      DIMENSION GAMMA(K, K, 0:PSTAR), PHISTA(K, K, 0:PSTAR), C(KKPST,
     *  KKPST)
C
C        TO SOLVE FOR THE PHI* ARRAYS, GIVEN P*;  KKPST=K*K*PSTAR
C        THE LINEAR MAPPING FUNCTION IS THE SAME AS IN S/R XCOVAR :
C
      ZROW(I, J, L, K) = ((L - 1) * K + J - 1) * K + I
C
C        NO ZERO PLANE HERE, THOUGH
C
      IFAULT = 0
      IF (KKPST .EQ. K * K * PSTAR) GOTO 10
      IFAULT = -1
      RETURN
   10 CONTINUE
      CALL SCALAR(PHISTA(1, 1, 0), K, -1.0)
      CALL LINZER(C, KKPST ** 2)
C
      DO 20 L = 1, PSTAR
      DO 20 I = 1, K
      DO 20 J = 1, K
C
      IROW = ZROW(I, J, L, K)
C
      DO 20 S = 1, PSTAR
C
      LMS = L - S
      MLMS = IABS(LMS)
C
      DO 20 IA = 1, K
C
      JCOL = ZROW(J, IA, S, K)
C
      IF (LMS .GE. 0) G = GAMMA(I, IA, LMS)
      IF (LMS .LT. 0) G = GAMMA(IA, I, MLMS)
C
C        GAMMA(-L)=GAMMA'(L)
C
      C(IROW, JCOL) = G
   20 CONTINUE
C
C        THIS COMPLETES THE FORMATION OF THE COEFF MATRIX FOR SYSTEM (2)
C
      CALL SOLVE(KKPST, .TRUE., C, GAMMA(1, 1, 1), DDET,
     * PHISTA(1, 1, 1), IFAULT)
      IF (IFAULT .NE. 0) IFAULT = PSTAR
C
      RETURN
```

```
      END
C
      SUBROUTINE DETERM(K, LMAX, A, DETS, C)
C
C        ALGORITHM AS232.4 APPL. STATIST. (1988) VOL. 37, NO. 1
C
      DIMENSION A(K, K, 0:LMAX), DETS(0:LMAX), C(K, K, 0:LMAX)
C
C        EVALUATE THE DETERMINANTS (BY LU FACTORISATION) OF EACH K-BY-K
C        MATRIX IN THE PLANES OF 'A'; STORE DET(A(,,L))  IN  DETS(L).
C
      KKLM1 = K * K * (LMAX + 1)
C
C        TOTAL NUMBER OF ELEMENTS
C
      CALL LINCOP(C, A, KKLM1)
C
C        PRESERVE A
C
      DO 1 L = 0, LMAX
    1 CALL SOLVE(K, .FALSE., C(1, 1, L), DETS, DETS(L), DETS, IFAULT)
C
C        THE TWO ARGUMENTS 'DETS' ARE NOT ACCESSED BY S/R SOLVE, SINCE
C        THE 'RHS' PARAMETER HAS THE VALUE .FALSE.
C
      RETURN
      END
```

```
      SUBROUTINE BABSRT(N, M, L, NN, SI, X, AVAIL, NBRNCH, POINTR, LIST,
     *  INDEX, VSORT, XIN, CSORT, C, CMAX, S, CRIT, BEST, NEVAL,
     *  IFAULT)
C
C        ALGORITHM AS233  APPL. STATIST. (1988) VOL. 37, NO. 1
C
C        BRANCH AND BOUND ALGORITHM FOR FEATURE SUBSET SELECTION
C        USING SORTING TO IMPROVE EFFICIENCY
C
      INTEGER N, M, MBAR, P, OMIT, NOMIT, PI, VJ, PP
      INTEGER LIST(N), NBRNCH(N), POINTR(N), INDEX(N), VSORT(N),
     *  NEVAL(N), BEST(N)
      LOGICAL AVAIL(N)
      REAL SI(N, N), X(N), XIN(N), CSORT(N), C(N), S(NN), CMAX(N),
     *  ZERO, TWO
C
      DATA ZERO, TWO /0.0, 2.0/
C
C        CHECK VALID VALUES OF L,M,N
C
      IFAULT = 1
      IF (M .LT. 2 .OR. M .GE. N .OR. L .LT. 0) RETURN
      IFAULT = 0
C
C        COPY SI INTO START OF WORKSPACE ARRAY S. EVALUATE
C        CRITERION FUNCTION WITH ALL VARIABLES INCLUDED.
C        INITIALIZE WORKSPACE ARRAYS.
C
      DOLD = ZERO
      K = 0
      DO 5 I = 1, N
      DO 4 J = 1, I
      K = K + 1
      S(K) = SI(I, J)
      DOLD = DOLD + TWO * S(K) * X(I) * X(J)
    4 CONTINUE
      DOLD = DOLD - S(K) * X(I) * X(I)
      AVAIL(I) = .TRUE.
      INDEX(I) = I
      CMAX(I) = ZERO
      NEVAL(I) = 0
    5 CONTINUE
C
C        INITIALIZATION
C
      NN6 = 6 * NN
      MBAR = N - M + 1
      P = M + 1
      I = 2
      POINTR(1) = 0
C
C        COPY INTO XIN VALUES OF X FOR VARIABLES INCLUDED AT LEVEL I-1
C
  100 K = 0
      DO 110 J = 1, N
      IF (INDEX(J) .LT. 0) GOTO 110
      K = K + 1
      XIN(K) = X(J)
  110 CONTINUE
C
C        EVALUATE CRITERION FUNCTION AFTER DROPPING EACH OF THE
```

```
C        VARIABLES WHICH ARE CURRENTLY AVAILABLE.  UPDATE
C        EVALUATION COUNTER.
C
      MM = N - I + 2
      II = (NN6 - MM * (MM + 1) * (MM + 12)) / 6
      K = 0
      DO 130 OMIT = 1, N
      IF (.NOT.AVAIL(OMIT)) GOTO 130
      NOMIT = INDEX(OMIT)
      KK = II + (NOMIT * NOMIT + NOMIT) / 2
      IK = KK - NOMIT
      D = ZERO
      DO 120 J = 1, MM
      IK = IK + I
      IF (J .GT. NOMIT) IK = IK + J - 2
      D = D + S(IK) * XIN(J)
  120 CONTINUE
      K = K + 1
      CSORT(K) = DOLD - D * D / S(KK)
      VSORT(K) = OMIT
      NEVAL(I - 1) = NEVAL(I - 1) + 1
  130 CONTINUE
C
C        SORT SMALLEST P VALUES INTO INCREASING ORDER
C
      PP = P
      IF (K .EQ. P) PP = PP -1
      DO 150 IK = 1, PP
      J1 = IK
      K1 = IK + 1
      DO 140 J = K1, K
      IF (CSORT(J) .LT. CSORT(J1)) J1 = J
  140 CONTINUE
      IF (J1 .EQ. IK) GOTO 150
      TEMP = CSORT(IK)
      CSORT(IK) = CSORT(J1)
      CSORT(J1) = TEMP
      ITEMP = VSORT(IK)
      VSORT(IK) = VSORT(J1)
      VSORT(J1) = ITEMP
  150 CONTINUE
C
C        STORE NUMBERS OF VARIABLES SELECTED IN LIST AND CRITERION
C        FUNCTION VALUES IN C. SIGNAL THAT THESE FEATURES ARE
C        NO LONGER AVAILABLE.
C
      K = POINTR(I - 1)
      DO 170 J = 1, P
      K = K + 1
      VJ = VSORT(J)
      LIST(K) = VJ
      C(K) = CSORT(J)
      AVAIL(VJ) = .FALSE.
      NBRNCH(K) = P - J + 1
  170 CONTINUE
      POINTR(I) = K + 1
C
C        BACKTRACK IF LIST AT LEVEL I IS EXHAUSTED OTHERWISE SELECT
C        NEXT VARIABLE TO THE LEFT ALONG LIST
C
  200 IF (POINTR(I) .EQ. POINTR(I - 1) + 1) GOTO 400
```

```
      POINTR(I) = POINTR(I) - 1
C
C        ABANDON FURTHER SEARCH IF CRITERION FUNCTION IS LESS THAN
C        CURRENT MAXIMUM L LEVELS FURTHER DOWN TREE.
C
  300 PI = POINTR(I)
      D = C(PI)
      IF (D .GT. CMAX(I)) CMAX(I) = D
      IL = I + L
      IF (IL .GT. MBAR) IL = MBAR
      IF (D .LT. CMAX(IL)) GOTO 600
C
C        IF AT LOWEST LEVEL OF TREE A NEW MAXIMUM HAS BEEN FOUND
C
      OMIT = LIST(PI)
      IF (I .EQ. MBAR) GOTO 500
C
C        UPDATE INVERSE MATRIX S READY FOR MOVE TO NEXT LEVEL DOWN
C
      NOMIT = INDEX(OMIT)
      MM = N - I + 2
      II = (NN6 - MM * (MM + 1) * (MM + 2)) / 6
      IJK = II + (MM * MM + MM) / 2
      KK = II + (NOMIT * NOMIT + NOMIT) /2
      IK = KK - NOMIT
      IJ = II
      DO 316 J = 1, MM
      IK = IK + 1
      IF (J - NOMIT) 304, 302, 303
  302 IJ = IJ + NOMIT
      GOTO 316
  303 IK = IK + J - 2
  304 JK = KK - NOMIT
      TEMP = S(IK) / S(KK)
      DO 310 K = 1, J
      IJ = IJ + 1
      JK = JK + 1
      IF (K - NOMIT) 307, 310, 306
  306 JK = JK + K - 1
  307 IJK = IJK + 1
      S(IJK) = S(IJ) - TEMP * S(JK)
  310 CONTINUE
  316 CONTINUE
C
C       UPDATE NUMBERING OF VARIABLES WITH RESPECT TO S
C       AND MOVE DOWN TO NEXT LEVEL
C
      INDEX(OMIT) = -N
      K = OMIT + 1
      DO 330 J = K, N
  330 INDEX(J) = INDEX(J) - 1
      DOLD = D
      P = NBRNCH(PI)
      I = I + 1
      GOTO 100
C
C        IF AT LEVEL 2 THEN SEARCH IS COMPLETE. OTHERWISE BACKTRACK
C        ONE LEVEL
C
  400 IF (I .EQ. 2) GOTO 999
      I = I - 1
```

```
        PI = POINTR(I)
        LPI = LIST(PI)
        AVAIL(LPI) = .TRUE.
        K = LPI + 1
        DO 410 J = K, N
  410   INDEX(J) = INDEX(J) + 1
        INDEX(LPI) = 1
        J = LPI
  420   J = J - 1
        IF (J .EQ. 0) GOTO 200
        IF (INDEX(J) .LT. 0) GO TO 420
        INDEX(LPI) = INDEX(J) + 1
        GOTO 200
C
C          STORE VARIABLE NUMBERS OF SUBSET WHICH HAS PRODUCED NEW MAX
C
  500   K = 0
        DO 510 J = 1, N
        IF (INDEX(J) .LT. 0 .OR. J .EQ. OMIT) GOTO 510
        K = K + 1
        BEST(K) = J
  510   CONTINUE
C
C          SIGNAL THAT VARIABLE AT CURRENT NODE AND ITS LEFT SIBLINGS
C          ARE AVAILABLE
C
  600   LPI = LIST(PI)
        AVAIL(LPI) = .TRUE.
        PI = PI - 1
        IF (PI .GT. POINTR(I - 1)) GOTO 600
        GOTO 400
C
C          SEARCH COMPLETE - SET MAXIMUM AND RETURN
C
  999   CRIT = CMAX(MBAR)
        RETURN
        END
```

```
      SUBROUTINE PPRANK(TA, UA, ET, SD, K3, K4, N, M, ISEL, IFAULT)
C
C        ALGORITHM AS234  APPL. STATIST. (1988) VOL. 37, NO. 2
C
C        Declaration formal parameters
C
      INTEGER N, M, ISEL, IFAULT
      REAL TA, UA, ET, SD, K3, K4
C
C        Auxiliary algorithm
C
      REAL SCORE
      EXTERNAL SCORE
C
C        Constants
C
      REAL ZERO, ONE, TWO, THREE, SIX, HALF, TWOHLF
      REAL CMT1, CMT2, CMT3, CTA1, CTA2, CTA3
      DATA ZERO, ONE, TWO, THREE /0.0E0, 1.0E0, 2.0E0, 3.0E0/
      DATA SIX, HALF, TWOHLF /6.0E0, 0.5E0, 2.5E0/
      DATA CMT1, CMT2, CMT3 /0.0833333E0, 2.3333333E0, 0.0125E0/
      DATA CTA1, CTA2, CTA3 /0.4166667E0, 0.0555555E0, 0.1666667E0/
C
C        Declaration internal variables
C
      INTEGER I
      REAL S1A, S2A, S3A, S4A, S1C, S2C, S3C, S4C, W, H1, H2, H3, H4,
     *  MAXU
      DATA MAXU /1.0E+10/
C
C        Checking input
C
      IFAULT = 1
      IF ((ISEL .LE. 0) .OR. (ISEL .GT. 4)) RETURN
      IF (ISEL .EQ. 4) GOTO 400
      IFAULT = 2
      IF (N .LE. 3) RETURN
C
C        Initializing auxiliary variables
C
      H1 = N - THREE
      H2 = N - TWO
      H3 = N - ONE
      H4 = N + ONE
C
C        Compute S1a, S2A, S3a, S4a
C
      S1A = ZERO
      DO 10 I = 1, N
      S1A = S1A + SCORE(I / H4)
   10 CONTINUE
      S1A = S1A / N
      S2A = ZERO
      S3A = ZERO
      S4A = ZERO
      DO 20 I = 1, N
      W = SCORE(I / H4) - S1A
      S2A = S2A + W ** 2
      S3A = S3A + W ** 3
      S4A = S4A + W ** 4
   20 CONTINUE
```

```
      IFAULT = 3
      IF (S2A .LE. ZERO) RETURN
      S2A = S2A / H3
      S3A = S3A / H3
      S4A = S4A / H3
      GOTO (100, 200, 300), ISEL
C
C        Two-sample
C
  100 IFAULT = 4
      IF (N .LT. 2 * M) RETURN
      S1C = (ONE * M) / N
      W = ONE - S1C
      S2C = (M * W ** 2 + (N - M) * S1C ** 2) / H3
      S3C = (M * W ** 3 + (N - M) * S1C ** 3) / H3
      S4C = (M * W ** 4 + (N - M) * S1C ** 4) / H3
      GOTO 500
C
C        Monotone trend
C
  200 S1C = H4 * HALF
      S2C = H4 * N * CMT1
      S3C = ZERO
      S4C = (N * N - CMT2) * H4 * N * CMT3
      GOTO 500
C
C        Independence
C
  300 S1C = S1A
      S2C = S2A
      S3C = S3A
      S4C = S4A
      GOTO 500
C
C        Compute ET, SD, K3, K4
C
  500 W = (N / H3) * (H4 / H3)
      ET = N * S1A * S1C
      SD = SQRT(H3 * S2A * S2C)
      K3 = (N / H2) * (S3A / S2A) * (S3C / S2C) / SD
      K4 = (H3 / H1) * (W * S4A / (S2A ** 2) - THREE) * (W * S4C /
     *  (S2C ** 2) - THREE) / (W * H2) - SIX / H4
C
C        Compute percentage point
C
  400 IFAULT = 5
      IF (ABS(UA) .GT. MAXU) RETURN
      W = UA ** 2
      TA = (ONE + (W - THREE) * K4 * CTA1 - (W - TWOHLF) * (K3 ** 2)
     *  * CTA2) * UA + (W - ONE) * K3 * CTA3
      TA = TA * SD + ET
      IFAULT = 0
      RETURN
      END
```

```
      SUBROUTINE RTALLY(NEWVAL, EMAX, MVCODE, MMAX, N, VALUES, COUNT,
     +          K, IFAULT)
C
C       ALGORITHM AS235  APPL. STATIST. (1988) VOL. 37, NO. 2
C
        INTEGER KCUT
        PARAMETER (KCUT = 25)
        INTEGER COUNT(*)
        INTEGER HI, I, J, K, KMAX, LO, M, N, IFAULT, MMAX
        REAL VALUES(*), NEWVAL, MVCODE(*)
C
C       Tallies (tabulates) input values NEWVAL with their frequencies.
C       The K distinct values are stored in array VALUES in ascending
C       order, and their frequencies are stored in COUNT.
C       N is the current number of observations.
C
        IFAULT = 0
        IF (MMAX .GT. 0) THEN
          DO 99 I = 1, MMAX
            IF (NEWVAL .EQ. MVCODE(I)) RETURN
   99     CONTINUE
        END IF
        N = N + 1
C
C       Initialize
C
        IF (N .LE. 1) THEN
          N = 1
          K = 1
          J = 1
          GO TO 5
        END IF
        J = K + 1
        IF (NEWVAL .GT. VALUES(K)) GO TO 3
C
        IF (K .LT. KCUT) THEN
C
C       Simple searching of VALUES from first element, for small K
C
          DO 1 I = 1, K
            J = I
            IF (NEWVAL - VALUES(J)) 3, 6, 1
    1     CONTINUE
        ELSE
C
C       Bisection search
C
          J = 1
          IF (NEWVAL .LT. VALUES(1)) GO TO 3
          LO = 0
          HI = K + 1
    2     J = (HI + LO) / 2
          IF (NEWVAL .EQ. VALUES(J)) GO TO 6
C
C       J = LO means that NEWVAL is bracketed between VALUES(LO) and
C       VALUES(HI).   Insert NEWVAL at position HI and move the rest of
C       VALUES and COUNT up.
C
          IF (J .EQ. LO) THEN
            J = HI
            GO TO 3
```

```
            END IF
            IF (NEWVAL .LT. VALUES(J)) THEN
              HI = J
            ELSE
              LO = J
            END IF
            GO TO 2
          END IF
C
      3   IF (K .EQ. KMAX) THEN
            IFAULT = 1
            N = N - 1
            RETURN
          END IF
          K = K + 1
          IF (K .GT. J) THEN
            DO 4 M = K, J+1, -1
              VALUES(M) = VALUES(M-1)
              COUNT(M) = COUNT(M-1)
      4     CONTINUE
          END IF
      5   VALUES(J) = NEWVAL
          COUNT(J) = 1
          RETURN
C
      6   COUNT(J) = COUNT(J) + 1
          RETURN
C
          END
```

```
{
       Algorithm AS236  Appl. Statist. (1988) Vol. 37, No. 2
                                                                    }

{  To Enumerate all RxC tables having given row and column sums

   To Evaluate the conditional probability of each table under
   hypergeometric and multinomial models

   To Evaluate likelihood across all tables according to the multinomial
   model                                                            }



{ ********************************************************************** }

{                          DATA STRUCTURES                             }

{ ********************************************************************** }

  CONST
  maxtablesize=10;               { maximum dimensions of table }
  maxN=300;                      { maximum global total of table entries - the
                                  values 10 and 300 may be changed, if desired }

  TYPE
  intvalues=0..maxN;             { permissible table entry type }
  faultint=0..123;               { fault indicator type
                                        0,100,20,3,120,103,23,123  }
  dimension=1..maxtablesize;     { table dimension type }
  vector = ARRAY[1..maxtablesize] OF intvalues;{ row and column vector type }
  matrix = ARRAY[1..maxtablesize,1..maxtablesize] OF real; { matrix type }

  return = ^returnlist;          { pointer to function returns }
  head = ^headlist;              { pointer to NODE information }
  table  =  ^tablist;  {  pointer to NODULE.  All  tables  are stored in
                          in a single linked list of head and table records. }
  tablist = RECORD               { Each table/nodule record contains the
                                             value of the:-      }
              entry:intvalues;   { current table entry }
              loghyper:real;     { current contribution to hypergeometric }
              multi:real;        { current contribution to multinomial }
              rest: head ;       { a pointer to the rest of the family of
                                   tables built onto the current entry
                                   ie to the next node }
               next:table        { a pointer to the next possible entry value
                                   for the current position and associated
                                   family of tables ie to the next nodule }
            END;


  headlist = RECORD
               number:integer;   { number of subtables emanating from the
                                   current node }
               position:ARRAY[1..2] OF dimension;  { current table position }
               first:table       { pointer to the first cell value and
                                  associated tables ie to the first nodule }
             END;

  returnlist = RECORD
                 tables:head;    {  a function call returns a pointer to a  }
```

```
                  value:real      {  family of tables and the value of the
                                     likelihood function evaluated across these }
                END;

    VAR
    rowsum:vector;                 { required row sums for tables }
    colsum:vector;                 { required column sums for tables }
    rows,cols:dimension;           { number of rows and columns in table }
    pmatrix:matrix;                { proposed transition matrix }
    result:return;                 { variable to receive function return }
    loghypercons,multicons:real;   { constants for hypergeometric and
                                     multinomial probabilities }
    ifault:faultint;               { fault indicator }


{ ********************************************************************** }

{                         OUTPUT  PROCEDURES                             }

{ ********************************************************************** }

PROCEDURE Report(r:return;i:faultint);

    TYPE
    tabmatrix = ARRAY[1..maxtablesize,1..maxtablesize] OF intvalues;

    PROCEDURE Printmat(tub:tabmatrix;prob1,prob2:real);

      VAR
      i,j:dimension;

      BEGIN                              { write out probabilities and current
                                                                      table }
      Writeln;
      Writeln('Hypergeometric probability = ',prob1);
      Writeln('Multinomial probability = ',prob2);
      FOR i:=1 TO rows DO
        BEGIN
        Writeln;
        Write('row ',i:1,' *  ');
        FOR j:=1 TO cols DO
          Write(tub[i,j],' ')
        END;

      Writeln
      END;

      PROCEDURE Printtables(v:real;example:head);

        CONST
        assumedzero=1.0e-15;

        VAR
        tab:tabmatrix;
        exam:table;
        i,j:dimension;
        hyperprob,multiprob:real;

        BEGIN
        WITH example^ DO
          BEGIN
```

```
          exam:=first;
          i:=position[1];
          j:=position[2]
          END;
        REPEAT
          WITH exam^ DO
            BEGIN
            tab[i,j]:=entry;           { copy current table entry into
                                         matrix for output }
            IF (i=rows) AND (j=cols) THEN { if table complete print out }
              BEGIN
              hyperprob:=Exp(loghyper+loghypercons);
              IF v>assumedzero THEN multiprob:=multi/v
              ELSE multiprob:=0.0;
              Printmat(tab,hyperprob,multiprob)
              END
            ELSE Printtables(v,rest)    { assemble rest of table through
                                          a recursive call to Printtables }
            END;
          exam:=exam^.next;
        UNTIL exam=NIL
        END;

    BEGIN {  Report  }
    Writeln;
    Writeln('IFAULT=',i:1);Writeln;
    IF i=0 THEN                        { if evaluation successful print out
                                           all tables and probabilities }
      BEGIN
      Writeln('Number of Tables Generated = ',r^.tables^.number);
      Writeln('Multinomial likelihood = ',r^.value*multicons);
      Printtables(r^.value,r^.tables)
      END
    END; {  Report  }



{ ************************************************************************ }

{                          PRINCIPAL FUNCTION                              }

{ ************************************************************************ }

FUNCTION Enum(rows,cols:dimension; rowsum,colsum:vector;
              pmatrix:matrix; VAR ifault:faultint):return;

    CONST
    oneminus=0.99999;
    oneplus=1.00001;

    TYPE
    dvector = ARRAY[0..maxN] OF real;   { real vector type }

    VAR
    logfact:dvector;
    sfault,tfault:faultint;
    i,j:dimension;
    rowtotal,coltotal:intvalues;
    total:real;

    FUNCTION max(a,b:integer):integer;
```

```
  BEGIN
  IF a>b THEN max:=a ELSE max:=b
  END; {  max  }

FUNCTION min(a,b:integer):integer;
  BEGIN
  IF a<b THEN min:=a ELSE min:=b
  END; {  min  }

FUNCTION Eval(n:intvalues;x:real):real;
  CONST assumedzero=1.0e-15;
  BEGIN
  IF n=0 THEN Eval:=1.0 ELSE
    IF x<assumedzero THEN Eval:=0.0 ELSE
      Eval:=Exp(n*Ln(x)-logfact[n])
  END; {  Eval  }

PROCEDURE Evalconst(rowtotal:intvalues);
  VAR
  i:intvalues;
  logprod1,logprod2:real;
  BEGIN
  logfact[0]:=0.0;
  FOR i:=1 TO rowtotal DO
    logfact[i]:=logfact[i-1]+Ln(i);
  logprod1:=0.0;
  FOR i:=1 TO rows DO
    logprod1:=logprod1+logfact[rowsum[i]];
  logprod2:=0.0;

  FOR i:=1 TO cols DO
    logprod2:=logprod2+logfact[colsum[i]];
  loghypercons:=logprod1+logprod2-logfact[rowtotal];
  multicons:=Exp(logprod1)
  END; {  Evalconst  }


FUNCTION RxC(rno,cno:dimension;rt,ct:intvalues;hp,mp:real;
                               rowsum,colsum:vector):return;
  VAR
  count,val,upper,lower,newrt,newct:intvalues;
  curloghyper,curmulti,newhp,newmp,total:real;
  newrowsum:vector;
  newcolsum:vector;
  newhead:head;
  result,newresult:return;
  newtable,nextable:table;

  BEGIN  {  RxC  }

  New(newtable);
  newrowsum:=rowsum;
  newcolsum:=colsum;

  IF (rno<rows) AND (cno<cols) THEN {  current position rno,cno
                                                        in body of table }
    BEGIN
    nextable:=newtable;
    upper:=min(rowsum[rno],colsum[cno]);
    lower:=max(0,rowsum[rno]+colsum[cno]-ct);
    newct:= ct-colsum[cno];
```

```
    count:=0;
    total:=0.0;
    FOR val:= lower TO upper DO     { step 2 - consider all possible values
                                      for the current position }
      BEGIN
      WITH nextable^ DO
        BEGIN
        entry:=val;
                                    { evaluate contributions to likelihood }
        multi:=Eval(val,pmatrix[rno,cno]);
        loghyper:=-logfact[val];
        newrowsum[rno]:=rowsum[rno]-val;
        newcolsum[cno]:=colsum[cno]-val;
        newrt:=rt-val;
        newhp:=hp+loghyper;
        newmp:=mp*multi;
        New(result);
                                    { step 3 - obtain results for remainder
                                      of the table by recursive call to RxC }
        result:=RxC(rno,cno+1,newrt,newct,newhp,newmp,newrowsum,newcolsum);
        rest:=result^.tables;


        count:=count+rest^.number;
        total:=total+result^.value*multi;
        IF val<upper THEN
          BEGIN
          New(next);
          nextable:=next
          END
        ELSE next:=NIL
        END
      END

  ELSE
    BEGIN
    New (result);

    IF rno<rows THEN                { current position in right-hand
                                                              margin }
      BEGIN
      val:=rowsum[rno];             { only one possible value }
      newrowsum[rno]:=0;
      newcolsum[cno]:=colsum[cno]-val;
      newrt:=rt-val;
      newct:=newrt;
                                    { evaluate contributions to likelihood }
      curloghyper:=-logfact[val];
      curmulti:=Eval(val,pmatrix[rno,cno]);
      newhp:=hp+curloghyper;
      newmp:=mp*curmulti;
                                    { step 3 - obtain results for remainder
                                      of the table by recursive call to RxC }
      result:=RxC(rno+1,1,newrt,newct,newhp,newmp,newrowsum,newcolsum);
      count:=result^.tables^.number;
      total:=result^.value*curmulti
      END

    ELSE
      IF cno<cols THEN              { current position in last row }
```

```
          BEGIN
          val:=colsum[cno];           { only one possible value }
          newcolsum[cno]:=0;
          newrowsum[rno]:=rowsum[rno]-val;
          newrt:=rt-val;
          newct:=newrt;
                                      { evaluate contributions to likelihood }
          curloghyper:=-logfact[val];
          curmulti:=Eval(val,pmatrix[rno,cno]);
          newhp:=hp+curloghyper;
          newmp:=mp*curmulti;
                                      { step 3 - obtain results for remainder
                                        of the table by recursive call to RxC }
          result:=RxC(rno,cno+1,newrt,newct,newhp,newmp,newrowsum,newcolsum);
          count:=result^.tables^.number;

          total:=result^.value*curmulti
          END

        ELSE                          { final cell }
          BEGIN
          val:=rt;                    { only one possible value }
          result^.tables:=NIL;        { table complete }
          count:=1;
                                      { evaluate contributions to likelihood }
          curloghyper:=hp-logfact[val];
          curmulti:=Eval(val,pmatrix[rno,cno]);
          total:=curmulti;
          curmulti:=curmulti*mp
          END;

      WITH newtable^ DO
        BEGIN
        entry:=val;
        loghyper:=curloghyper;
        multi:=curmulti;
        rest:=result^.tables;
        next:=NIL
        END
      END;

    New(newhead);                     { assemble entries for current NODE }
    WITH newhead^ DO
      BEGIN
      number:=count;
      position[1]:=rno;
      position[2]:=cno;
      first:=newtable
      END;
    New(newresult);
    WITH newresult^ DO
      BEGIN
      tables:=newhead;
      value:=total
      END;
    RxC:=newresult
    END; {  RxC  }

  BEGIN {  Enum  }
                                      { check validity of input values }
  ifault:=0;
```

```
                                    { totals of row and column sums must be
                                                  equal }
    rowtotal:=0;
    coltotal:=0;
    FOR i:= 1 TO rows DO
      rowtotal:=rowtotal+rowsum[i];
    FOR i:=1 TO cols DO
      coltotal:=coltotal+colsum[i];

    IF rowtotal<>coltotal THEN ifault:=1;

                                    { pmatrix entries must be probabilities
                                        and rows must sum to one }
    sfault:=0;
    tfault:=0;
    FOR i:=1 TO rows DO
      BEGIN
      total:=0.0;
      FOR j:=1 TO cols DO
        BEGIN
        IF (pmatrix[i,j]<0.0) OR (pmatrix[i,j]>1.0) THEN sfault:=1;
        total:=total+pmatrix[i,j]
        END;
      IF (total<oneminus) OR (total>oneplus) THEN tfault:=1
      END;
    ifault:=100*ifault+20*sfault+3*tfault;

                                    { if valid input, evaluate tables and
                                              probabilities }
    IF ifault=0 THEN
      BEGIN
      Evalconst(rowtotal);
      Enum:=RxC(1,1,rowtotal,coltotal,0.0,1.0,rowsum,colsum)
      END
    ELSE
      Enum:=NIL
    END;  {  Enum  }
```

```
      SUBROUTINE BGM(N, RHO, KMAX1, IWIND, TRUNC, TOLER, DELTA, TSTAT,
     *  CHSTAT, TSIGN, CHSIGN, MAXP, K, WKSP, LGTH, ISWCHI, ISWSGN,
     *  IFAULT)
C
C        ALGORITHM AS237  APPL. STATIST. (1988) VOL. 37, NO. 2
C
C        COMPUTES THE DELTA, T- AND CHI SQUARED-STATISTIC ARRAYS FOR THE
C        CORNER METHOD OF BEGUIN, GOURIEROUX AND MONFORT (IMPROVED)
C        SUBPROGRAMS : CEACMA, CALBGM, QFBGM, NORMAL(AS 2)
C
      DIMENSION DELTA(MAXP, K), TSTAT(MAXP, K), CHSTAT(MAXP, K)
      DIMENSION TSIGN(MAXP, K), CHSIGN(MAXP, K)
      DIMENSION RHO(KMAX1), WKSP(LGTH)
C
      REAL TRUNC, TOLER
      REAL EPS, TENTH
C
      DATA EPS, ETA, TENTH /1.0E-12, 1.0E-60, 0.1/
C
C        STORAGE MANAGEMENT AND ERROR DETECTION
C
      K2 = K + 2
      LCOVAR = (K + 1) * K / 2
      LWRHO = KMAX1
      LWLAM = 2 * KMAX1 - 1
      IWKSP1 = 1
      IWKSP2 = IWKSP1 + K2
      IWKSP3 = IWKSP2 + K2
      IWKSP4 = IWKSP3 + LCOVAR
      IWKSP5 = IWKSP4 + LWRHO
      IWKSP6 = IWKSP5 + LWLAM
      IWKSP7 = IWKSP6 + K2 * K
      IWKSP8 = IWKSP7 + K2 * K
      LWKSP = LGTH - (IWKSP8 - 1)
      IFAULT = 0
      IF (MAXP .LT. K) IFAULT = 6
      IF (ISWCHI .LT. (-1) .OR. ISWCHI .GT. 1) IFAULT = 7
      IF (ISWSGN .NE. 0 .AND. ISWSGN .NE. 1) IFAULT = 7
      IF (TOLER .GT. TENTH .OR. TOLER .LT. EPS) IFAULT = 8
      IF (LWKSP .LT. 3 * K2) IFAULT = 9
      IF (K .LE. 0) IFAULT = 1
      IF (IFAULT .NE. 0) RETURN
C
C        PRELIMINARIES
C
      CALL CEACMA(N, RHO, KMAX1, IWIND, TRUNC, WKSP(IWKSP4), LWRHO,
     *  WKSP(IWKSP5), LWLAM, WKSP(IWKSP3), LCOVAR, K, IFAULT)
      IF (IFAULT .NE. 0) RETURN
C
C        CALL TO COMPUTING SUBROUTINE
C
      CALL CALBGM(RHO, KMAX1, DELTA, TSTAT, CHSTAT, TSIGN, CHSIGN, MAXP,
     *  K, WKSP(IWKSP3), LCOVAR, WKSP(IWKSP1), WKSP(IWKSP2), K2,
     *  WKSP(IWKSP6), WKSP(IWKSP7), WKSP(IWKSP8), EPS, ETA, TOLER,
     *  ISWCHI, ISWSGN, IFAULT)
      RETURN
      END

      SUBROUTINE CEACMA(N, RHO, KMAX1, IWIND, TRUNC, WRHO, LWRHO, WLAM,
     *  LWLAM, COVAR, LCOVAR, K, IFAULT)
C
```

```
C        ALGORITHM AS237.1  APPL. STATIST. (1988) VOL. 37, NO. 2
C
C        COMPUTATION OF A CONSISTENT ESTIMATOR OF THE ASYMPTOTIC
C        COVARIANCE MATRIX OF THE FIRST K AUTOCORRELATION ESTIMATORS
C
      DIMENSION RHO(KMAX1), WRHO(LWRHO), WLAM(LWLAM), COVAR(LCOVAR)
C
      REAL TRUNC
      REAL ZERO, ONE, TWO, SIX, HALF
C
      DATA ZERO, ONE, TWO, SIX, HALF /0.0, 1.0, 2.0, 6.0, 0.5/
C
      FN = N
      IF (K .LE. 0) IFAULT = 1
      IF (IWIND .LT. 0 .OR. IWIND .GT. 2) IFAULT = 2
      IF (KMAX1 .LE. K) IFAULT = 3
      IF (TRUNC .LT. HALF .OR. TRUNC .GT. SIX) IFAULT = 4
      IF (N .LE. K) IFAULT = 5
      IF (LCOVAR .NE. (K + 1) * K / 2 .OR. LWRHO .NE. KMAX1 .OR.
     *  LWLAM .NE. 2 * KMAX1 - 1) IFAULT = 20
      IF (IFAULT .NE. 0) RETURN
C
C        COMPUTING WINDOWED AUTOCORRELATIONS WRHO
C
      BN = INT(SQRT(FN) * TRUNC)
      DO 100 MP1 = 1, LWRHO
      FACT = ONE
      IF (IWIND .EQ. 0) GOTO 90
      X = FLOAT(MP1 - 1) / BN
      FACT = ZERO
      IF (IWIND .EQ. 2) GOTO 20
      IF (X .LE. ONE) FACT = ONE - X
      GOTO 90
   20 IF (X .LE. HALF) FACT = ONE - SIX * X ** 2 * (ONE - X)
      IF (X .LE. ONE .AND. X .GT. HALF) FACT = TWO * (ONE - X) ** 3
   90 WRHO(MP1) = RHO(MP1) * FACT
  100 CONTINUE
C
C        COMPUTING WINDOWED LAMBDA
C
      DO 250 MP1 = 1, LWLAM
      WLAM(MP1) = ZERO
      NTERM = LWLAM - MP1 + 1
      DO 200 LP1 = 1, NTERM
      LABS = IABS(LP1 - LWRHO) + 1
      LPMABS = IABS(LP1 + MP1 - LWRHO - 1) + 1
      WLAM(MP1) = WLAM(MP1) + WRHO(LABS) * WRHO(LPMABS)
  200 CONTINUE
  250 CONTINUE
C
C        EVALUATING THE ASYMPTOTIC COVARIANCE MATRIX OF THE
C        AUTOCORRELATIONS
C
      IND = 0
      K1 = K + 1
      DO 300 M = 2, K1
      DO 300 L = M, K1
      LMM = L - M + 1
      LPM = L + M - 1
      IND = IND + 1
      COVAR(IND) = (WLAM(LMM) + WLAM(LPM) - (WRHO(M) * WLAM(L) + WRHO(L)
```

```
     *  * WLAM(M) - WRHO(M) * WRHO(L) * WLAM(1)) * TWO) / FN
  300 CONTINUE
      RETURN
      END

      SUBROUTINE CALBGM(RHO, KMAX1, DELTA, TSTAT, CHSTAT, TSIGN, CHSIGN,
     *  MAXP, K, COVAR, LCOVAR, DELTA0, DELTA1, K2, DERIV0, DERIV1,
     *  WKSP, EPS, ETA, TOLER, ISWCHI, ISWSGN, IFAULT)
C
C       ALGORITHM AS237.2  APPL. STATIST. (1988) VOL. 37, NO. 2
C
      DIMENSION RHO(KMAX1), WKSP(3, K2), COVAR(LCOVAR)
      DIMENSION DELTA(MAXP, K), TSTAT(MAXP, K), CHSTAT(MAXP, K)
      DIMENSION TSIGN(MAXP, K), CHSIGN(MAXP, K)
      DIMENSION DELTA0(K2), DELTA1(K2)
      DIMENSION DERIV0(K2, K), DERIV1(K2, K)
      DIMENSION CHI(1), P(1), Q(1), Z(1)
C
      LOGICAL ZDELTA
      REAL TOLER
      REAL EPS, ZERO, ONE, TWO, RT2PI
C
      DATA ZERO, ONE, TWO, RT2PI /0.0, 1.0, 2.0, 2.5066282746/
C
C       INITIALIZATION OF TWO LINES OF THE DELTA AND DERIV ARRAYS
C
      IFAIL1 = 0
      IFAIL2 = 0
      DO 440 IQ = 1, K2
      DELTA1(IQ) = ONE
      DO 420 M = 1, K
      DERIV0(IQ, M) = ZERO
      DERIV1(IQ, M) = ZERO
  420 CONTINUE
  440 CONTINUE
      DO 460 I2 = 1, K2
      IQ1 = IABS(I2 - 2) + 1
      DELTA0(I2) = RHO(IQ1)
      IF (I2 .NE. 2) DERIV0(I2, IQ1 - 1) = ONE
  460 CONTINUE
C
C       LOOP ON P (THE AR ORDER)
C
      DO 1000 IP = 1, K
      K3MP = K - IP + 3
C
C       LOOP ON IQ (THE MA ORDER) FOR COMPUTING A LINE OF DELTA
C       AND DERIV
C
      DO 900 I1 = 2, K3MP
      ZDELTA = ABS(DELTA0(I1))  .LT. ETA
      IF (IP .EQ. 1 .OR. ZDELTA) GOTO 600
      TEMP = (DELTA1(I1) ** 2 - DELTA1(I1 - 1) * DELTA1(I1 + 1))
     *  / DELTA0(I1)
      IF (ISWCHI .EQ. (-1)) GOTO 550
      DO 500 M = 1, K
      DERIV0(I1, M) = (TWO * DELTA1(I1) * DERIV1(I1, M) -
     *                 DELTA1(I1 - 1)   * DERIV1(I1 + 1, M) -
     *                 DELTA1(I1 + 1)   * DERIV1(I1 - 1, M) -
     *                 TEMP * DERIV0(I1, M)) / DELTA0(I1)
  500 CONTINUE
```

```
  550 DELTA0(I1) = TEMP
C
C       EVALUATING THE ASYMPTOTIC VARIANCE OF AN ELEMENT OF DELTA
C       AND THE COVARIANCES WITH THREE OTHER DELTA ELEMENTS
C
  600 IQ = I1 - 2
      IF (IQ .LT. 1) GOTO 900
      DELTA(IP, IQ) = DELTA0(I1)
      IF (ISWCHI .EQ. (-1)) GOTO 900
      TSTAT(IP, IQ) = ZERO
      IF (ISWSGN .EQ. 1) TSIGN(IP, IQ) = ONE
      IF (ZDELTA) GOTO 700
      VAR = ZERO
      IF (ISWCHI .EQ. 0) GOTO 610
      WKSP(2, IQ) = ZERO
      A31 = ZERO
      A32 = ZERO
  610 INDM = 0
      KMAX = IP + IQ - 1
      DO 640 M = 1, KMAX
      FACTOR = ONE
      IND = INDM
      DO 630 L = M, KMAX
      IND = IND + 1
      VAR = VAR + FACTOR * DERIV0(I1, M) * COVAR(IND) * DERIV0(I1, L)
      FACTOR = TWO
      IF (ISWCHI .EQ. 0) GOTO 630
      TEMPM = COVAR(IND) * DERIV0(I1, M)
      WKSP(2, IQ) = WKSP(2, IQ) + TEMPM * DERIV0(I1 - 1, L)
      IF (IP .EQ. 1) GOTO 620
      A31 = A31 + TEMPM * DERIV1(I1, L)
      A32 = A32 + TEMPM * DERIV1(I1 + 1, L)
  620 IF (L .EQ. M) GOTO 630
      TEMPL = COVAR(IND) * DERIV0(I1, L)
      WKSP(2, IQ) = WKSP(2, IQ) + TEMPL * DERIV0(I1 - 1, M)
      IF (IP .EQ. 1) GOTO 630
      A31 = A31 + TEMPL * DERIV1(I1, M)
      A32 = A32 + TEMPL * DERIV1(I1 + 1, M)
  630 CONTINUE
      INDM = INDM + K - M + 1
  640 CONTINUE
      IF (VAR .GT. ETA) GOTO 650
      IFAIL1 = 1
      GOTO 700
C
C       COMPUTING T-STATISTIC
C
  650 TSTAT(IP, IQ) = DELTA0(I1) / SQRT(VAR)
C
C       COMPUTING T-PROBABILITY OF SIGNIFICANCE
C
      IF (ISWSGN .EQ. 0) GOTO 700
      CALL NORMAL(TSTAT(IP, IQ), 1, P, Q, Z, IFLT)
      PQ = P(1)
      IF (Q(1) .LT. PQ) PQ = Q(1)
      TSIGN(IP, IQ) = TWO * PQ
C
C       COMPUTING CHI2-STATISTIC
C
  700 IF (ISWCHI .EQ. 0) GOTO 900
      CHSTAT(IP - 1, IQ) = ZERO
```

```
      IF (ISWSGN .EQ. 1) CHSIGN(IP - 1, IQ) = ONE
      IF (ZDELTA) GOTO 900
      WKSP(3, IQ) = VAR
      IF (IP .EQ. 1) GOTO 780
      CHSTAT(IP - 1, IQ) = QFBGM(WKSP(1, IQ), WKSP(2, IQ + 1),
     *  WKSP(3, IQ + 1), A31, A32, VAR, DELTA1(I1), DELTA1(I1 + 1),
     *  DELTA0(I1), ETA, TOLER, NDF)
      IF (NDF .GT. 0) GOTO 750
      IFAIL2 = 1
      GOTO 780
C
C        COMPUTING CHI2-PROBABILITY OF SIGNIFICANCE
C
  750 IF (ISWSGN .EQ. 0) GOTO 780
      CHI(1) = SQRT(CHSTAT(IP - 1, IQ) )
      CALL NORMAL(CHI, 1, P, Q, Z, IFLT)
      IF (NDF .EQ. 1) CHSIGN(IP - 1, IQ) = TWO * Q(1)
      IF (NDF .EQ. 2) CHSIGN(IP - 1, IQ) = RT2PI * Z(1)
      IF (NDF .EQ. 3) CHSIGN(IP - 1, IQ) = TWO * (Q(1) + Z(1) * CHI(1))
C
  780 WKSP(1, IQ) = VAR
  900 CONTINUE
C
C        UPDATE AUXILIARY ARRAYS
C
      DELTA0(1) = DELTA0(3)
      DO 920 M = 1, K
  920 DERIV0(1, M) = DERIV0(3, M)
      DO 980 I = 1, K3MP
      TEMP = DELTA1(I)
      DELTA1(I) = DELTA0(I)
      DELTA0(I) = TEMP
      DO 960 M = 1, K
      TEMP = DERIV1(I, M)
      DERIV1(I, M) = DERIV0(I, M)
      DERIV0(I, M) = TEMP
  960 CONTINUE
  980 CONTINUE
C
 1000 CONTINUE
C
      IFAULT = -(IFAIL1 + 2 * IFAIL2)
      IF (ZDELTA) IFAULT = -4
      RETURN
      END

      REAL FUNCTION QFBGM(A11, A21, A22, A31, A32, A33, V1, V2, V3, ETA,
     *  TOLER, NDF)
C
C        ALGORITHM AS237.3  APPL. STATIST. (1988) VOL. 37, NO. 2
C
C                                                    -1
C        COMPUTATION OF THE QUADRATIC FORM Q = V'.A  .V  WITH A 3X3
C        SYMMETRIC MATRIX 'A', BY USING A CHOLESKY DECOMPOSITION
C
      REAL TOLER
      REAL ETA, ZERO
C
      DATA ZERO /0.0/
      NDF = 0
      QFBGM = ZERO
```

```
      D11 = A11
      IF (D11 .LT. ETA) GOTO 30
      W1 = V1
      S21 = A21 / D11
      D22 = A22 - D11 * S21 ** 2
      IF (D22 .LT. ETA) GOTO 20
      T21 = -S21
      W2 = T21 * V1 + V2
      S31 = A31 / D11
      S32 = (A32 - D11 * S21 * S31) / D22
      D33 = A33 - D11 * S31 ** 2 - D22 * S32 ** 2
      IF (D33 .LT. ABS(TOLER * A33)) GOTO 10
      IF (D33 .LT. ETA) GOTO 10
      T31 = -S31 - S32 * T21
      T32 = -S32
      W3 = T31 * V1 + T32 * V2 + V3
      QFBGM = QFBGM + W3 ** 2 / D33
      NDF = NDF + 1
   10 IF (D22 .LT. ABS(TOLER * A22)) GOTO 20
      QFBGM = QFBGM + W2 ** 2 / D22
      NDF = NDF + 1
   20 IF (D11 .LT. ABS(TOLER * A11)) GOTO 30
      QFBGM = QFBGM + W1 ** 2 / D11
      NDF = NDF + 1
   30 CONTINUE
      RETURN
      END

      SUBROUTINE CORNER(TABLE, MAXP, LAG, LAG1, CRIT, MINP, MDLP, MDLQ,
     *  NB, IFAULT)
C
C        ALGORITHM AS237.4  APPL. STATIST. (1988) VOL. 37, NO. 2
C
C        DETECTS ALL THE CORNERS IN TABLE
C        (THE ABSOLUTE VALUE OF THE ELEMENTS OF TABLE COMPARED TO CRIT)
C
      DIMENSION TABLE(MAXP, LAG)
      REAL ONE, CRIT
      INTEGER MINP(LAG1), MDLP(LAG1), MDLQ(LAG1)
C
      DATA ONE /1.0/
C
      IFAULT = 0
      IF (MAXP .LT. LAG .OR. LAG .LE. 0) IFAULT = 12
      IF (LAG1 .NE. LAG + 1) IFAULT = 13
      NB = 0
      IF (IFAULT .NE. 0) RETURN
C
      IQ = 0
      DO 300 M = 1, LAG
      IP = LAG + 1 - M
      J = IQ + 1
      DO 100 L = J, M
      IQ = M + J - L
C
C        IF TABLE DOES NOT CONTAIN PROBABILITIES OF SIGNIFICANCE
C        BUT T OR CHI2 STATISTICS, REPLACE .LT. BY .GT. IN :
C
      IF (ABS(TABLE(IP, IQ)) .LT. CRIT) GOTO 300
  100 CONTINUE
  200 IQ = J - 1
```

```
      300 MINP(IP) = IQ
          MINP(LAG1) = 0
          IF (MINP(1) .EQ. 0) GOTO 500
          INDP = LAG1
          DO 400 M = 1, LAG1
          IF (MINP(M) .EQ. INDP) GOTO 400
          NB = NB + 1
          MDLP(NB) = M - 1
          MDLQ(NB) = MINP(M)
          INDP = MINP(M)
      400 CONTINUE
      500 RETURN
          END
```

```
PROCEDURE karstor(n:nonnegint;VAR x,y,d:realarray;VAR alpha,beta,sad:real;
VAR ifault:nonnegint);

{Algorithm AS 238  Appl. Statist. (1988) Vol.37, No.3}

{Pascal translation of the original bubblesort variant of algorithm AS74
 for the L1 norm fitting of a straight line with real storage of integers
 Applied Statistics (1974),Vol 23,No 2}

CONST big=1000000;
     prec=0.000001;
VAR  a,b,i,j,it,p,q:nonnegint;
     sum,xa,ya,xi,yi:real;
     stop:boolean;
     t:ARRAY[nonnegint] of real; {pairwise slopes}

BEGIN
i:=1;xa:=x[1];
stop:=true;
WHILE (i<n) AND stop DO
   BEGIN
   i:=i+1;stop:=abs(x[i]-xa)<prec
   END;
IF stop THEN ifault:=1
ELSE
   BEGIN
   a:=1;b:=0;
   it:=0;ifault:=2;
   REPEAT
      it:=it+1;

      {calculate slopes}
      sum:=0.0;
      FOR i:=1 TO n DO
         BEGIN
         xa:=x[a];ya:=y[a];
         xi:=x[i]-xa;yi:=y[i]-ya;
         d[i]:=i;
         IF abs(xi)<prec THEN t[i]:=big
         ELSE t[i]:=yi/xi;
         sum:=sum+abs(xi)
         END;
      sum:=0.5*sum;

      {order slopes}
      FOR j:=n-1 DOWNTO 1 DO
      FOR i:=1 TO j DO
         BEGIN
         p:=trunc(d[i]+0.5);q:=trunc(d[i+1]+0.5);
         IF t[p]>t[q] THEN
            BEGIN
            d[i]:=q;d[i+1]:=p
            END
         END;

      {select best line}
      i:=0;xi:=0.0;
      REPEAT
         i:=i+1;p:=trunc(d[i]+0.5);
         xi:=xi+abs(x[p]-xa);
         IF xi>=sum THEN i:=n
```

```
      UNTIL i=n;

      IF p=b THEN
         BEGIN
         ifault:=0;it:=n
         END
      ELSE
         BEGIN
         b:=a;a:=p
         END
   UNTIL it=n;

   {L1 estimates,norm and residuals}
   beta:=t[p];alpha:=y[a]-beta*x[a];
   sum:=0.0;
   FOR i:= 1 TO n DO
      BEGIN
      yi:=y[i]-alpha-beta*x[i];d[i]:=yi;
      sum:=sum+abs(yi)
      END;
   sad:=sum
   END
END;{of karstor}


PROCEDURE karst(n:nonnegint;VAR x,y,e:realarray;VAR alpha,beta,sad:real;
VAR ifault:nonnegint);

{Algorithm AS 238  Appl. Statist. (1988) Vol.37, No.3}

{partial insertionsort variant of algorithm AS74
for the L1 norm fitting of a straight line with a boolean trap
Applied Statistics (1974),Vol 23,No 2}

CONST big=10000000;
     prec=0.000001;
VAR  a,c,i,j,k:nonnegint;
     hold,sum,sum2,xa,ya:real;
     jump,stop:boolean;
     d:ARRAY[nonnegint] of nonnegint; {permuted indices}
     t:ARRAY[nonnegint] of real;      {pairwise slopes}
   old:ARRAY[nonnegint] of boolean;   {previous values of a}

BEGIN
i:=1;xa:=x[1];
stop:=true;
WHILE (i<n) AND stop DO
   BEGIN
   i:=i+1;stop:=abs(x[i]-xa)<prec
   END;
IF stop THEN ifault:=1
ELSE
   BEGIN
   a:=1;stop:=false;
   ifault:=0;
   FOR i:=2 TO n DO old[i]:=false;

   REPEAT

      {calculate slopes}
      xa:=x[a];ya:=y[a];
```

```
         old[a]:=true;sum:=0.0;
         FOR i:=1 TO n DO
            BEGIN
            hold:=x[i]-xa;
            IF abs(hold)<prec
            THEN t[i]:=big
            ELSE t[i]:=(y[i]-ya)/hold;
            sum:=sum+abs(hold)
            END;
         sum2:=0.5*sum;

         {locate weighted median}
         k:=1;d[1]:=1;
         sum:=abs(x[1]-xa);
         FOR i:=2 TO n DO
            BEGIN k:=k+1;j:=k;
            sum:=sum+abs(x[i]-xa);hold:=t[i];
            REPEAT
               j:=j-1;c:=d[j];
               IF hold<t[c] THEN d[j+1]:=c
               ELSE
                  BEGIN
                  d[j+1]:=i;j:=0
                  END
            UNTIL j<2;
            IF j=1 THEN d[1]:=i;

            jump:=false;
            REPEAT
               IF sum>sum2 THEN
                  BEGIN
                  hold:=sum-abs(x[d[k]]-xa);
                  IF hold>sum2 THEN
                     BEGIN
                     sum:=hold;k:=k-1
                     END
                  ELSE jump:=true
                  END
               ELSE jump:=true

            UNTIL jump
            END;

         c:=d[k];
         IF old[c] THEN stop:=true ELSE a:=c
      UNTIL stop;

      {L1 estimates,norm and residuals}
      beta:=t[c];alpha:=y[a]-beta*x[a];
      sum:=0.0;
      FOR i:= 1 TO n DO
         BEGIN
         hold:=y[i]-alpha-beta*x[i];e[i]:=hold;
         sum:=sum+abs(hold)
         END;
      sad:=sum
      END
   END;{of karst}


   PROCEDURE simbarrob(n:nonnegint;VAR x,y,d:realarray;VAR alpha,beta,sad:real;
```

```
   VAR ifault:nonnegint);

   {Algorithm AS 238  Appl. Statist. (1988) Vol.37, No.3}

   {Pascal translation of algorithm AS132
    for the L1 norm fitting of a straight line
    Applied Statistics (1978),Vol 27,No 3}

   LABEL 160;
   CONST big=10000000;
         acu=0.000001;
   VAR  a1,a2,aaa,aaaa,ahalf,aone,bbb,bbbb,ddd,det,hold,ratio,sum,t,test,
        tot1,tot2,y1,y2:real;
        i,j,ibas1,ibas2,iin,iout,isave,iter:nonnegint;
        rho,zzz:-2..2;
        flag,more,stop:boolean;
        inext:ARRAY[nonnegint] of posnegint;

   FUNCTION isign(i:posnegint;k:posnegint):posnegint;
   {Pascal implementation of Fortran isign function}
   BEGIN
   IF k>=0 THEN isign:=i ELSE isign:=-i
   END;{of isign}

   BEGIN

   {determine initial basis}
   a1:=x[1];y1:=y[1];
   i:=1;stop:=true;
   WHILE (i<n) AND stop DO
      BEGIN
      i:=i+1;stop:=abs(a1-x[i])<acu
      END;
   IF stop THEN ifault:=1
   ELSE
      BEGIN
      ifault:=0;iter:=0;
      ahalf:=0.5+acu;aone:=ahalf+ahalf;
      a2:=x[i];y2:=y[i];
      ibas1:=1;ibas2:=i;
      det:=1.0/(a2-a1);

      {initial values of a,b,d}
      aaaa:=(a2*y1-a1*y2)*det;bbbb:=(y2-y1)*det;
      FOR i:=2 TO n DO
      IF y[i]>=(aaaa+bbbb*x[i]) THEN d[i]:=1.0 ELSE d[i]:=-1.0;
      d[ibas1]:= 0.0;d[ibas2]:=-1.0;
      tot1:=1.0;tot2:=x[ibas2];
      FOR i:=2 TO n DO
      BEGIN
         tot1:=tot1+d[i];tot2:=tot2+d[i]*x[i];
      END;
      t:=(a2*tot1-tot2)*det;flag:=abs(t)>aone;

      {main iterative loop begins}
      REPEAT
      flag:= NOT flag;
      IF flag THEN
         BEGIN
         t:=(tot2-a1*tot1)*det;
         IF abs(t) < aone THEN GOTO 160;
```

```
      iout:=ibas2;x[iout]:=a1;
      aaa:=a1;bbb:=a2;
      END
   ELSE
      BEGIN
      IF iter=0 THEN det:=-det
      ELSE
         BEGIN
         t:=(tot2-a2*tot1)*det;
         IF abs(t) < aone THEN GOTO 160;
         END;
      aaa:=a2;bbb:=a1;
      iout:=ibas1;
      END;{not flag}
   IF t>0.0 THEN rho:=1
   ELSE
   BEGIN
      rho:=-1;t:=-t;
      det:=-det
   END;
   t:=0.5*t;

   {perform partial sort of ratios}
   ratio:=big;sum:=ahalf;
   inext[ibas1]:=ibas2;
   FOR i:=1 TO n DO
   BEGIN
      ddd:=(x[i]-aaa)*det;
      IF ddd*d[i]>acu THEN
      BEGIN
         test:=(y[i]-(aaaa+bbbb*x[i]))/ddd;
         IF test<ratio THEN
            BEGIN
            j:=ibas1;sum:=sum+abs(ddd);
            more:=true;stop:=false;
               REPEAT
               isave:=abs(inext[j]);
               IF test >= d[isave] THEN
                  BEGIN
                  more:=false;stop:=true
                  END
               ELSE
                  BEGIN
                  IF sum<t THEN stop:=true
                  ELSE
                     BEGIN
                     hold:=sum-abs((x[isave]-aaa)*det);
                     IF hold<t THEN stop:=true
                     ELSE
                        BEGIN
                        d[isave]:=isign(1,inext[j]);sum:=hold;
                        inext[j]:=inext[isave]
                        END{hold>=t}
                     END{sum>=t}
                  END;{test<d{isave]}
               UNTIL stop;

            IF more THEN
               REPEAT
               j:=isave;isave:=abs(inext[j]);
               UNTIL test>=d[isave];
```

```
            inext[i]:=inext[j];
            IF d[i]>=0.0 THEN inext[j]:=i ELSE inext[j]:=-i;
            d[i]:=test;
            IF sum>=t THEN
               BEGIN
               iin:=abs(inext[ibas1]);ratio:=d[iin]
               END
            END;{test<ratio}
        END;{ddd*d[i]>acu}
     END;{i loop}

  {update basis indicaTOrs}
  iin:=abs(inext[ibas1]);j:=iin;
  stop:=false;
     REPEAT
     isave:=abs(inext[j]);
     IF isave=ibas2 THEN stop:=true
     ELSE
        BEGIN
        zzz:=isign(1,inext[j]);d[isave]:=-zzz;
        zzz:=zzz+zzz;j:=isave;
        tot1:=tot1-zzz;tot2:=tot2-zzz*x[isave]
        END;
     UNTIL stop;

  zzz:=isign(1,inext[ibas1]);d[iout]:=-rho;
  tot1:=tot1-rho-zzz;tot2:=tot2-rho*bbb-zzz*x[iin];
  iter:=iter+1;
  IF flag THEN
     BEGIN
     x[ibas2]:=a2;ibas2:=iin;
     a2:=x[iin];y2:=y[iin];
     det:=1.0/(a1-a2);d[ibas2]:=-1.0;
     aaaa:=(a1*y2-a2*y1)*det;bbbb:=(y1-y2)*det
     END
  ELSE
     BEGIN
     ibas1:=iin;ibas1:=iin;
     a1:=x[iin];y1:=y[iin];
     det:=1.0/(a2-a1);d[ibas1]:=0.0;
     aaaa:=(a2*y1-a1*y2)*det;bbbb:=(y2-y1)*det
  END;
  UNTIL false;

  {calculate estimates,deviations,sad}
160:sum:=0.0;
  FOR i:=1 TO n DO
     BEGIN
     ddd:=y[i]-(aaaa+bbbb*x[i]);d[i]:=ddd;
     sum:=sum+abs(ddd)
  END;
  alpha:=aaaa;beta :=bbbb;
  sad:=sum;

  END
END;{of simbarrob}
```

```
      DOUBLE PRECISION FUNCTION GAMMAD(X, P, IFAULT)
C
C       ALGORITHM AS239  APPL. STATIST. (1988) VOL. 37, NO. 3
C
C       Computation of the Incomplete Gamma Integral
C
C       Auxiliary functions required: ALOGAM = logarithm of the gamma
C       function, and ALNORM = algorithm AS66
C
        INTEGER IFAULT
        DOUBLE PRECISION PN1, PN2, PN3, PN4, PN5, PN6, X, TOL, OFLO,
     *          XBIG, ARG, C, RN, P, A, B, ONE, ZERO, ALOGAM,
     *          AN, TWO, ELIMIT, PLIMIT, ALNORM, THREE, NINE
        PARAMETER (ZERO = 0.D0, ONE = 1.D0, TWO = 2.D0, OFLO = 1.D+37,
     *          THREE = 3.D0, NINE = 9.D0, TOL = 1.D-14, XBIG = 1.D+8,
     *          PLIMIT = 1000.D0, ELIMIT = -88.D0)
        EXTERNAL ALOGAM, ALNORM
C
        GAMMAD = ZERO
C
C       Check that we have valid values for X and P
C
        IF (P .LE. ZERO .OR. X .LT. ZERO) THEN
          IFAULT = 1
          RETURN
        END IF
        IFAULT = 0
        IF (X .EQ. ZERO) RETURN
C
C       Use a normal approximation if P > PLIMIT
C
        IF (P .GT. PLIMIT) THEN
          PN1 = THREE * SQRT(P) * ((X / P) ** (ONE / THREE) + ONE /
     *          (NINE * P) - ONE)
          GAMMAD = ALNORM(PN1, .FALSE.)
          RETURN
        END IF
C
C       If X is extremely large compared to P then set GAMMAD = 1
C
        IF (X .GT. XBIG) THEN
          GAMMAD = ONE
          RETURN
        END IF
C
        IF (X .LE. ONE .OR. X .LT. P) THEN
C
C       Use Pearson's series expansion.
C       (Note that P is not large enough to force overflow in ALOGAM).
C       No need to test IFAULT on exit since P > 0.
C
          ARG = P * LOG(X) - X - ALOGAM(P + ONE, IFAULT)
          C = ONE
          GAMMAD = ONE
          A = P
   40     A = A + ONE
          C = C * X / A
          GAMMAD = GAMMAD + C
          IF (C .GT. TOL) GO TO 40
          ARG = ARG + LOG(GAMMAD)
          GAMMAD = ZERO
```

```
            IF (ARG .GE. ELIMIT) GAMMAD = EXP(ARG)
C
          ELSE
C
C         Use a continued fraction expansion
C
            ARG = P * LOG(X) - X - ALOGAM(P, IFAULT)
            A = ONE - P
            B = A + X + ONE
            C = ZERO
            PN1 = ONE
            PN2 = X
            PN3 = X + ONE
            PN4 = X * B
            GAMMAD = PN3 / PN4
   60       A = A + ONE
            B = B + TWO
            C = C + ONE
            AN = A * C
            PN5 = B * PN3 - AN * PN1
            PN6 = B * PN4 - AN * PN2
            IF (ABS(PN6) .GT. ZERO) THEN
              RN = PN5 / PN6
              IF (ABS(GAMMAD - RN) .LE. MIN(TOL, TOL * RN)) GO TO 80
              GAMMAD = RN
            END IF
C
            PN1 = PN3
            PN2 = PN4
            PN3 = PN5
            PN4 = PN6
            IF (ABS(PN5) .GE. OFLO) THEN
C
C         Re-scale terms in continued fraction if terms are large
C
              PN1 = PN1 / OFLO
              PN2 = PN2 / OFLO
              PN3 = PN3 / OFLO
              PN4 = PN4 / OFLO
            END IF
            GO TO 60
   80       ARG = ARG + LOG(GAMMAD)
            GAMMAD = ONE
            IF (ARG .GE. ELIMIT) GAMMAD = ONE - EXP(ARG)
          END IF
C
          RETURN
          END
```

```
      SUBROUTINE SSPINV(X, XM, XSSPI, E, WT, SUMWT, NP, NPP, IFAULT)
C
C        ALGORITHM AS240  APPL. STATIST. (1988) VOL. 37, NO. 3
C
C        THIS SUBROUTINE UPDATES THE MEAN VECTOR XM, AND THE INVERSE
C        MATRIX OF CORRECTED SUMS OF SQUARES AND PRODUCTS XSSPI.
C        A DATA VECTOR X WITH WEIGHT WT .GT. 0 IS INCLUDED,
C        OR EXCLUDED WHEN WT .LT. 0.
C        IFAULT = 2, IF UPDATED SUMWT .LE. 0,
C               = 1, IF MANIPULATED MATRIX XSSPI DOESN'T EXIST,
C               = 0, OTHERWISE.
C
      REAL X(NP), XM(NP), E(NP), XSSPI(NPP), WT, SUMWT, ZERO, ONE, EPS,
     *  A, B, C, ZABS
C
      DATA ZERO, ONE, EPS /0.0, 1.0, 1.0E-5/
C
      ZABS(A) = ABS(A)
C
      IFAULT = 2
      SUMWT = SUMWT + WT
      IF (SUMWT .LE. ZERO) RETURN
      IFAULT = 1
C
C        UPDATE MEANS AND INVERSE MATRIX OF SUMS OF SQUARES AND
C        PRODUCTS.
C
      B = WT / SUMWT
      C = WT - B * WT
      DO 10 I = 1, NP
      X(I) = X(I) - XM(I)
   10 XM(I) = XM(I) + B * X(I)
C
      DO 30 I = 1, NP - 1
      E(I) = ZERO
      K = I * (I - 1) / 2
      DO 20 J = 1, I
      K = K + 1
   20 E(I) = E(I) + XSSPI(K) * X(J)
      DO 25 J = I + 1, NP
      K = K + J - 1
   25 E(I) = E(I) + XSSPI(K) * X(J)
   30 CONTINUE
      E(NP) = ZERO
      K = NP * (NP - 1) / 2
      DO 35 J = 1, NP
      K = K + 1
   35 E(NP) = E(NP) + XSSPI(K) * X(J)
C
      A = ZERO
      DO 40 I = 1, NP
   40 A = A + X(I) * E(I)
      A = A + ONE / C
      IF (ZABS(A) .LE. EPS) RETURN
      IFAULT = 0
C
      K = 0
      DO 70 I = 1, NP
      DO 70 J = 1, I
      K = K + 1
      XSSPI(K) = XSSPI(K) - E(I) * E(J) / A
```

```
   70 CONTINUE
      RETURN
      END
```

```
      REAL FUNCTION PPND7 (P, IFAULT)
C
C       ALGORITHM AS241  APPL. STATIST. (1988) VOL. 37, NO. 3, 477-
C       484.
C
C       Produces the normal deviate Z corresponding to a given lower
C       tail area of P; Z is accurate to about 1 part in 10**7.
C
C       The hash sums below are the sums of the mantissas of the
C       coefficients.   They are included for use in checking
C       transcription.
C
      REAL ZERO, ONE, HALF, SPLIT1, SPLIT2, CONST1, CONST2, A0, A1,
     *          A2, A3, B1, B2, B3, C0, C1, C2, C3, D1, D2, E0, E1, E2,
     *          E3, F1, F2, P, Q, R
      PARAMETER (ZERO = 0.0, ONE = 1.0, HALF = 0.5,
     *          SPLIT1 = 0.425, SPLIT2 = 5.0,
     *          CONST1 = 0.180625, CONST2 = 1.6)
C
C       Coefficients for P close to 0.5
C
      PARAMETER (A0 = 3.38713 27179E+00, A1 = 5.04342 71938E+01,
     *           A2 = 1.59291 13202E+02, A3 = 5.91093 74720E+01,
     *           B1 = 1.78951 69469E+01, B2 = 7.87577 57664E+01,
     *           B3 = 6.71875 63600E+01)
C       HASH SUM AB    32.31845 77772
C
C       Coefficients for P not close to 0, 0.5 or 1.
C
      PARAMETER (C0 = 1.42343 72777E+00, C1 = 2.75681 53900E+00,
     *           C2 = 1.30672 84816E+00, C3 = 1.70238 21103E-01,
     *           D1 = 7.37001 64250E-01, D2 = 1.20211 32975E-01)
C       HASH SUM CD    15.76149 29821
C
C       Coefficients for P near 0 or 1.
C
      PARAMETER (E0 = 6.65790 51150E+00, E1 = 3.08122 63860E+00,
     *           E2 = 4.28682 94337E-01, E3 = 1.73372 03997E-02,
     *           F1 = 2.41978 94225E-01, F2 = 1.22582 02635E-02)
C       HASH SUM EF    19.40529 10204
C
      IFAULT = 0
      Q = P - HALF
      IF (ABS(Q) .LE. SPLIT1) THEN
        R = CONST1 - Q * Q
        PPND7 = Q * (((A3 * R + A2) * R + A1) * R + A0) /
     *              (((B3 * R + B2) * R + B1) * R + ONE)
        RETURN
      ELSE
        IF (Q .LT. ZERO) THEN
          R = P
        ELSE
          R = ONE - P
        END IF
        IF (R .LE. ZERO) THEN
          IFAULT = 1
          PPND7 = ZERO
          RETURN
        END IF
        R = SQRT(-LOG(R))
        IF (R .LE. SPLIT2) THEN
```

```
            R = R - CONST2
            PPND7 = (((C3 * R + C2) * R + C1) * R + C0) /
     *               ((D2 * R + D1) * R + ONE)
          ELSE
            R = R - SPLIT2
            PPND7 = (((E3 * R + E2) * R + E1) * R + E0) /
     *               ((F2 * R + F1) * R + ONE)
          END IF
          IF (Q .LT. ZERO) PPND7 = - PPND7
          RETURN
        END IF
        END
C
C
        DOUBLE PRECISION FUNCTION PPND16 (P, IFAULT)
C
C       ALGORITHM AS241  APPL. STATIST. (1988) VOL. 37, NO. 3
C
C       Produces the normal deviate Z corresponding to a given lower
C       tail area of P; Z is accurate to about 1 part in 10**16.
C
C       The hash sums below are the sums of the mantissas of the
C       coefficients.   They are included for use in checking
C       transcription.
C
        DOUBLE PRECISION ZERO, ONE, HALF, SPLIT1, SPLIT2, CONST1,
     *         CONST2, A0, A1, A2, A3, A4, A5, A6, A7, B1, B2, B3,
     *         B4, B5, B6, B7,
     *         C0, C1, C2, C3, C4, C5, C6, C7, D1, D2, D3, D4, D5,
     *         D6, D7, E0, E1, E2, E3, E4, E5, E6, E7, F1, F2, F3,
     *         F4, F5, F6, F7, P, Q, R
        PARAMETER (ZERO = 0.D0, ONE = 1.D0, HALF = 0.5D0,
     *         SPLIT1 = 0.425D0, SPLIT2 = 5.D0,
     *         CONST1 = 0.180625D0, CONST2 = 1.6D0)
C
C       Coefficients for P close to 0.5
C
        PARAMETER (A0 = 3.38713 28727 96366 6080D0,
     *             A1 = 1.33141 66789 17843 7745D+2,
     *             A2 = 1.97159 09503 06551 4427D+3,
     *             A3 = 1.37316 93765 50946 1125D+4,
     *             A4 = 4.59219 53931 54987 1457D+4,
     *             A5 = 6.72657 70927 00870 0853D+4,
     *             A6 = 3.34305 75583 58812 8105D+4,
     *             A7 = 2.50908 09287 30122 6727D+3,
     *             B1 = 4.23133 30701 60091 1252D+1,
     *             B2 = 6.87187 00749 20579 0830D+2,
     *             B3 = 5.39419 60214 24751 1077D+3,
     *             B4 = 2.12137 94301 58659 5867D+4,
     *             B5 = 3.93078 95800 09271 0610D+4,
     *             B6 = 2.87290 85735 72194 2674D+4,
     *             B7 = 5.22649 52788 52854 5610D+3)
C       HASH SUM AB    55.88319 28806 14901 4439
C
C       Coefficients for P not close to 0, 0.5 or 1.
C
        PARAMETER (C0 = 1.42343 71107 49683 57734D0,
     *             C1 = 4.63033 78461 56545 29590D0,
     *             C2 = 5.76949 72214 60691 40550D0,
     *             C3 = 3.64784 83247 63204 60504D0,
     *             C4 = 1.27045 82524 52368 38258D0,
```

```
     *             C5 = 2.41780 72517 74506 11770D-1,
     *             C6 = 2.27238 44989 26918 45833D-2,
     *             C7 = 7.74545 01427 83414 07640D-4,
     *             D1 = 2.05319 16266 37758 82187D0,
     *             D2 = 1.67638 48301 83803 84940D0,
     *             D3 = 6.89767 33498 51000 04550D-1,
     *             D4 = 1.48103 97642 74800 74590D-1,
     *             D5 = 1.51986 66563 61645 71966D-2,
     *             D6 = 5.47593 80849 95344 94600D-4,
     *             D7 = 1.05075 00716 44416 84324D-9)
C        HASH SUM CD    49.33206 50330 16102 89036
C
C        Coefficients for P near 0 or 1.
C
         PARAMETER (E0 = 6.65790 46435 01103 77720D0,
     *             E1 = 5.46378 49111 64114 36990D0,
     *             E2 = 1.78482 65399 17291 33580D0,
     *             E3 = 2.96560 57182 85048 91230D-1,
     *             E4 = 2.65321 89526 57612 30930D-2,
     *             E5 = 1.24266 09473 88078 43860D-3,
     *             E6 = 2.71155 55687 43487 57815D-5,
     *             E7 = 2.01033 43992 92288 13265D-7,
     *             F1 = 5.99832 20655 58879 37690D-1,
     *             F2 = 1.36929 88092 27358 05310D-1,
     *             F3 = 1.48753 61290 85061 48525D-2,
     *             F4 = 7.86869 13114 56132 59100D-4,
     *             F5 = 1.84631 83175 10054 68180D-5,
     *             F6 = 1.42151 17583 16445 88870D-7,
     *             F7 = 2.04426 31033 89939 78564D-15)
C        HASH SUM EF    47.52583 31754 92896 71629
C
         IFAULT = 0
         Q = P - HALF
         IF (ABS(Q) .LE. SPLIT1) THEN
           R = CONST1 - Q * Q
           PPND16 = Q * (((((((A7 * R + A6) * R + A5) * R + A4) * R + A3)
     *                  * R + A2) * R + A1) * R + A0) /
     *               (((((((B7 * R + B6) * R + B5) * R + B4) * R + B3)
     *                 * R + B2) * R + B1) * R + ONE)
           RETURN
         ELSE
           IF (Q .LT. ZERO) THEN
             R = P
           ELSE
             R = ONE - P
           END IF
           IF (R .LE. ZERO) THEN
             IFAULT = 1
             PPND16 = ZERO
             RETURN
           END IF
           R = SQRT(-LOG(R))
           IF (R .LE. SPLIT2) THEN
             R = R - CONST2
             PPND16 = (((((((C7 * R + C6) * R + C5) * R + C4) * R + C3)
     *                  * R + C2) * R + C1) * R + C0) /
     *               (((((((D7 * R + D6) * R + D5) * R + D4) * R + D3)
     *                 * R + D2) * R + D1) * R + ONE)
           ELSE
             R = R - SPLIT2
             PPND16 = (((((((E7 * R + E6) * R + E5) * R + E4) * R + E3)
```

```
*                  * R + E2) * R + E1) * R + E0) /
*               (((((((F7 * R + F6) * R + F5) * R + F4) * R + F3)
*                  * R + F2) * R + F1) * R + ONE)
      END IF
      IF (Q .LT. ZERO) PPND16 = - PPND16
      RETURN
   END IF
   END
```

```
      SUBROUTINE MARMA(K, N, P, Q, W, IK, PHI, THETA, MEAN, MEANS,
     *                 QQ, XTOL, RLOGL, V, WORK, LWORK, IWORK, LIWORK,
     *                 IFAULT)
C
C     ALGORITHM AS 242.1  APPL. STATIST. (1989), VOL.38, NO. 1
C
C     Auxiliary routines required: DECOMP & SOLVE from CACM algorithm 423
C
      INTEGER           P, Q, R, K, N, LWORK, IFAULT, KR, KMAT, KC, LW1,
     *                  LW2, LW3, LW4, LW5, LW6, LW7, LW8, LW9, LW10,
     *                  LW11, LW12, LW13, IK, I, J, LIWORK, IWORK(LIWORK)
      LOGICAL           MEAN
      REAL              MEANS(K), PHI(IK,P*K+1), QQ(IK,K), REAL,
     *                  V(IK,N), W(IK,N), WORK(LWORK), THETA(IK,Q*K+1),
     *                  TWO, LOG2PI, CONST, RLOGL, XTOL, TSIG
      PARAMETER         (TWO = 2.0, LOG2PI = 1.8378771)
      INTRINSIC         MAX, REAL
      EXTERNAL          VARMA, CHOL
C
C     test for errors in the input data
C
      IFAULT = 0
      IF(K .LT. 1) IFAULT = 1
      IF(IFAULT .EQ. 1) RETURN
      IF(N .LT. 1) IFAULT = 2
      IF(IFAULT .EQ. 2) RETURN
      IF(P .LT. 0) IFAULT = 3
      IF(IFAULT .EQ. 3) RETURN
      IF(Q .LT. 0) IFAULT = 4
      IF(IFAULT .EQ. 4) RETURN
      IF(P .EQ. 0 .AND. Q .EQ. 0) IFAULT = 5
      IF(IFAULT .EQ. 5) RETURN
      IF(IK .LT. K) IFAULT = 6
      IF(IFAULT .EQ. 6) RETURN
C
C     check that workspace arrays are big enough
C
      R = MAX(P,Q)
      KR = K * R
      KMAT = K * MAX(R, K * (P + 1))
      CONST = (REAL(N * K) / TWO) * LOG2PI
C
      KC = K * K
      LW1 = 1
      LW2 = K * KR + LW1
      LW3 = K * KR + LW2
      LW4 = K * KR + LW3
      LW5 = KC + LW4
      LW6 = KC + LW5
      LW7 = KC + LW6
      LW8 = KC + LW7
      LW9 = KC + LW8
      LW10 = KC + LW9
      LW11 = KC * (P + 1) + LW10
      LW12 = KC * (R + 1) + LW11
      LW13 = KMAT * KMAT + LW12
C
      I = MAX(IK * K, LW13 + KC - 1)
      IF(LWORK .LT. I) IFAULT = 7
      IF(IFAULT .EQ. 7) RETURN
      IF(LIWORK .LT. KMAT) IFAULT = 8
```

```
      IF(IFAULT .EQ. 8) RETURN
C
C     set upper triangle of QQ to lower triangle of QQ
C
      DO 30 I = 2, K
         DO 31 J = 1, I - 1
            QQ(J,I) = QQ(I,J)
   31    CONTINUE
   30 CONTINUE
C
C     check whether QQ is positive-definite and evaluate its determinant
C     (storing it in TSIG)
C
      CALL CHOL(QQ, IK, K, WORK, IFAULT)
      IF(IFAULT .GT. 0) THEN
         IFAULT = 9
         RETURN
      ELSE
         TSIG = WORK(1)
         DO 20 I = 2, K
            TSIG = TSIG * WORK((I - 1) * IK + I)
   20    CONTINUE
         TSIG = TSIG * TSIG
      ENDIF
C
C     now call VARMA to calculate the log likelihood function
C
      CALL VARMA(K, N, P, Q, R, W, IK, PHI, THETA, MEAN, MEANS, QQ,
     *           TSIG, CONST, V, XTOL, RLOGL, KR, KMAT, WORK(LW1),
     *           WORK(LW2), WORK(LW3), WORK(LW4), WORK(LW5),
     *           WORK(LW6), WORK(LW7), WORK(LW8), WORK(LW9),
     *           WORK(LW10), WORK(LW11), WORK(LW12), WORK(LW13),
     *           IWORK, IFAULT)
C
      END
      SUBROUTINE VARMA(K, N, P, Q, R, W, IK, PHI, THETA, MEAN, MEANS,
     *                 QQ, TSIG, CONST, V, XTOL, RLOGL, KR, KMAT,
     *                 GAMMA, TEMP, TEMPK, TEMPL, F, INVF, TEMPM, A,
     *                 MT, B, Z, MAT, WA, IWORK, IFAULT)
C
C     ALGORITHM AS 242.2  APPL. STATIST. (1989), VOL.38, NO. 1
C
      INTEGER           IFAULT, K, KMAT, KR, N, P, Q, R, IK, I, J,
     *                  J7, K1, K2, L, L8, M, T, IWORK(KMAT)
      LOGICAL           MEAN, ANOTT, DELTA, PEARL
      REAL              CONST, RLOGL, TSIG, XTOL, ZERO, ONE, TWO,
     *                  A(K,K), B(K*K*(P+1)), F(K,K), GAMMA(K,KR),
     *                  INVF(K,K), MAT(KMAT,KMAT), MEANS(K), MT(K,K),
     *                  PHI(IK,P*K+1), QQ(IK,K), TEMP(K,KR),
     *                  TEMPL(K,K), TEMPM(K,K), THETA(IK,Q*K+1),
     *                  W(IK,N), Z(K*K*(R+1)), WA(K,K), TEMPK(K,KR),
     *                  DETP, SIG, SM, SSQ, SUM, REAL, V(IK,N)
      PARAMETER         (ZERO = 0.0, ONE = 1.0, TWO = 2.0)
      INTRINSIC         LOG, REAL, ABS, MAX
      EXTERNAL          COVARS, CHOL, BKSB
C
C     This subroutine computes the log likelihood function of a vector
C     ARMA model
C
      ANOTT = .FALSE.
      IF(P .GT. Q) ANOTT = .TRUE.
```

```
C
C     call COVARS to calculate the autocovariances of W(t) and the
C     cross-covariances between W(t) and E(t)
C
      CALL COVARS(K, P, Q, R, IK, PHI, THETA, QQ, GAMMA, Z, KMAT,
     *            MAT, B, IWORK, IFAULT)
C
      IF(IFAULT .GT. 0) IFAULT = 10
      IF(IFAULT .EQ. 10) RETURN
C
C     calculate the first k columns of P(1/0) and store as TEMPK
C
      DO 140 K1 = 1, R
         DO 120 I = 1, K
            DO 100 J = 1, K
               SUM = ZERO
               DO 40 K2 = 1, K
                  DO 20 M = K1, P
                     SUM = SUM + PHI(I, (M - 1) * K + K2) *
     *                   Z((M - K1 + 1) * K * K + (K2 - 1) * K + J)
   20             CONTINUE
   40          CONTINUE
               DO 80 M = K1, Q
                  DO 60 K2 = 1, K
                     SUM = SUM - THETA(I, (M - 1) * K + K2) *
     *                   GAMMA(J, (M - K1) * K + K2)
   60             CONTINUE
   80          CONTINUE
C
               TEMPK(I, (K1 - 1) * K + J) = SUM
  100       CONTINUE
  120    CONTINUE
  140 CONTINUE
C
C     initialize A(1/0), V(1), SSQ, F(1) and DETP
C
      DO 160 I = 1, KR
         Z(I) = ZERO
  160 CONTINUE
C
      DO 200 I = 1, K
         DO 180 J = 1, K
            F(I,J) = TEMPK(I,J) + QQ(I,J)
  180    CONTINUE
         V(I,1) = W(I,1)
         IF(MEAN) V(I,1) = V(I,1) - MEANS(I)
         B(I) = V(I,1)
  200 CONTINUE
C
C     factorise F(1) as LL' and store L as A
C
      CALL CHOL(F, K, K, A, IFAULT)
      IF(IFAULT .NE. 0) THEN
         IFAULT = 11
         RETURN
      ENDIF
C
C     calculate determinant (DETP) of F(1)
C
      DETP = A(1,1)
      DO 220 I = 2, K
```

```
            DETP = DETP * A(I,I)
      220 CONTINUE
          DETP = DETP * DETP
C
C     set MT = F(1) inverse
C
          DO 260 I = 1, K
             DO 240 J = 1, K
                MT(I,J) = ZERO
      240    CONTINUE
             MT(I,I) = ONE
      260 CONTINUE
C
C     note that BKSB cannot fail
C
          CALL BKSB(A, K, K, K, .FALSE., MT, IFAULT)
C
C     set upper triangle of INVF to A'
C
          DO 300 J = 1, K
             DO 280 I = 1, J
                INVF(I,J) = A(J,I)
      280    CONTINUE
      300 CONTINUE
          CALL BKSB(INVF, K, K, K, .TRUE., MT, IFAULT)
C
          CALL BKSB(A, K, K, 1, .FALSE., B, IFAULT)
          SSQ = ZERO
          DO 320 I = 1, K
             SSQ = SSQ + B(I) * B(I)
      320 CONTINUE
C
          DETP = LOG(DETP)
C
C     calculate L(1) and K(1)
C
          DO 360 I = 1, K
             DO 340 J = 1, KR
                MAT(I,J) = ZERO
                GAMMA(I,J) = ZERO
      340    CONTINUE
      360 CONTINUE
C
          DO 480 L = 1, R
             DO 460 I = 1, K
                DO 440 J = 1, K
                   SUM = ZERO
                   IF(L .LE. P) THEN
                      DO 380 K2 = 1, K
                         SUM = SUM + PHI(I, (L - 1) * K + K2) *
     *                         TEMPK(K2,J)
      380             CONTINUE
                   ENDIF
                   IF(L .LT. R) SUM = SUM + TEMPK(I, L * K + J)
                   DO 420 K2 = 1, K
                      SM = ZERO
                      IF(L .LE. P) SM = PHI(I, (L - 1) * K + K2)
                      IF(L .LE. Q) SM = SM - THETA(I, (L - 1) * K + K2)
                      SUM = SUM + SM * QQ(K2,J)
      420             CONTINUE
                   MAT(I, (L - 1) * K + J) = SUM
```

```
C
  440       CONTINUE
  460    CONTINUE
  480 CONTINUE
C
      IF(ANOTT) THEN
C
         DO 580 L = 1, R
            DO 560 I = 1, K
               DO 540 J = 1, K
                  SUM = ZERO
                  IF(L .LE. Q) THEN
                     DO 500 K2 = 1, K
                        SUM = SUM + THETA(I, (L - 1) * K + K2) *
     *                        TEMPK(K2,J)
  500                CONTINUE
                  ENDIF
                  IF(L .LT. R) SUM = SUM + TEMPK(I, L * K + J)
                  GAMMA(I, (L - 1) * K + J) = SUM
  540          CONTINUE
  560       CONTINUE
  580    CONTINUE
C
      ELSE
C
         DO 640 L = 1, R
            DO 620 I = 1, K
               DO 600 J = 1, K
                  GAMMA(I, (L - 1) * K + J) = MAT(I, (L - 1) * K + J)
  600          CONTINUE
  620       CONTINUE
  640    CONTINUE
C
      ENDIF
C
C     start the recursions
C
      DELTA = .FALSE.
      PEARL = .FALSE.
      J7 = N
C
      DO 2240 T = 2, N
C
         IF(ANOTT  .AND.  T .GT. P - Q) PEARL = .TRUE.
C
C        calculate TEMPK
C
         IF(.NOT.DELTA) THEN
C
C           find L' (inverse) and store as INVF
C
            DO 680 J = 1, K
               DO 660 I = 1, K
                  TEMPK(I,J) = A(J,I)
                  INVF(I,J) = ZERO
  660          CONTINUE
               INVF(J,J) = ONE
  680       CONTINUE
            CALL BKSB(TEMPK, K, K, K, .TRUE., INVF, IFAULT)
C
            DO 760 L = 1, R
```

```
                DO 740 I = 1, K
                   DO 720 J = 1, K
                      SUM = ZERO
                      DO 700 K2 = 1, K
                         SUM = SUM + GAMMA(I, (L - 1) * K + K2) *
     *                         INVF(K2,J)
  700                 CONTINUE
                      TEMPK(I, (L - 1) * K + J) = SUM
  720              CONTINUE
  740           CONTINUE
  760        CONTINUE
C
C            calculate TEMP
C
           ENDIF
           DO 800 L = 1, KR
              TEMP(1,L) = ZERO
  800     CONTINUE
C
           IF(ANOTT) GOTO 900
C
C          TEMP = T * A(t-1/t-2)
C
           IF(T .EQ. 2) GOTO 1080
           DO 880 L = 1, R
              DO 860 I = 1, K
                 SUM = ZERO
                 IF(L .LE. P) THEN
                    DO 820 K2 = 1, K
                       SUM = SUM + PHI(I, (L - 1) * K + K2) * Z(K2)
  820               CONTINUE
                 ENDIF
                 IF(L .LT. R) SUM = SUM + Z(L * K + I)
                 TEMP(1, (L - 1) * K + I) = SUM
  860        CONTINUE
  880     CONTINUE
           GOTO 1080
  900     IF(T .GT. 2) THEN
C
C             TEMP = A * A(t-1/t-2) + R * W(t-1)
C
              DO 980 L = 1, R
                 DO 960 I = 1, K
                    SUM = ZERO
                    IF(L .LE. Q) THEN
                       DO 920 K2 = 1, K
                          SUM = SUM + THETA(I, (L - 1) * K + K2) * Z(K2)
  920                  CONTINUE
                    ENDIF
                    IF(L .LT. R) SUM = SUM + Z(L * K + I)
                    TEMP(1, (L - 1) * K + I) = SUM
  960            CONTINUE
  980         CONTINUE
C
           ENDIF
           DO 1060 L = 1, R
              DO 1040 I = 1, K
                 SUM = ZERO
                 DO 1020 K2 = 1, K
                    SM = ZERO
                    IF(L .LE. P) SM = PHI(I, (L - 1) * K + K2)
```

```
                  IF(L .LE. Q) SM = SM - THETA(I, (L - 1) * K + K2)
                  IF(MEAN) THEN
                     SUM = SUM + SM * (W(K2,T - 1) - MEANS(K2))
                  ELSE
                     SUM = SUM + SM * W(K2,T - 1)
                  ENDIF
 1020          CONTINUE
               TEMP(1, (L - 1) * K + I) = TEMP(1, (L -1 ) * K + I)
     *                                    + SUM
 1040       CONTINUE
 1060    CONTINUE
C
C        A(t/t-1) = TEMP + TEMPK * V(t-1)
C
 1080    DO 1160 L = 1, R
            DO 1140 L8 = 1, K
               SUM = TEMP(1, (L - 1) * K + L8)
               IF((.NOT.DELTA) .OR. (.NOT.ANOTT)) THEN
                  DO 1100 K2 = 1, K
                     SUM = SUM + TEMPK(L8, (L - 1) * K + K2) * B(K2)
 1100             CONTINUE
               ENDIF
               Z((L - 1) * K + L8) = SUM
 1140       CONTINUE
 1160    CONTINUE
C
         IF(DELTA) GOTO 2180
C
C        calculate TEMPL
C
         DO 1200 I = 1, K
            DO 1180 J = 1, K
               TEMPL(I,J) = MAT(I,J)
 1180       CONTINUE
 1200    CONTINUE
C
C        recalculate TEMP
C
         IF(.NOT.ANOTT) THEN
C
C           TEMP = T * L(t-1)
C
            DO 1300 L = 1, R
               DO 1280 I = 1, K
                  DO 1260 J = 1, K
                     SUM = ZERO
                     IF(L .LE. P) THEN
                        DO 1220 K2 = 1, K
                           SUM = SUM + PHI(I, (L - 1) * K + K2) *
     *                           MAT(K2,J)
 1220                   CONTINUE
                     ENDIF
                     IF(L .LT. R) SUM = SUM + MAT(I, L * K + J)
                     TEMP(I, (L - 1) * K + J) = SUM
C
 1260             CONTINUE
 1280          CONTINUE
 1300       CONTINUE
C
         ELSE
C
```

```
C           TEMP = A * L(t-1)
C
            DO 1420 L = 1, R
               DO 1400 I = 1, K
                  DO 1380 J = 1, K
                     SUM = ZERO
                     IF(L .LE. Q) THEN
                        DO 1340 K2 = 1, K
                           SUM = SUM + THETA(I, (L - 1) * K + K2) *
     *                           MAT(K2,J)
 1340                   CONTINUE
                     ENDIF
                     IF(L .LT. R) SUM = SUM + MAT(I, L * K + J)
                     TEMP(I, (L - 1) * K + J) = SUM
C
 1380             CONTINUE
 1400          CONTINUE
 1420       CONTINUE
         ENDIF
C
C        calculate TEMPM = M(t-1) * (h'L(t-1))'
C
C        calculate choleski factorization of MT and store as INVF
C
         CALL CHOL(MT, K, K, INVF, IFAULT)
C
C        compute h'L(t-1) * INVF
C
         DO 1500 I = 1, K
            DO 1480 J = 1, K
               SUM = ZERO
               DO 1460 K2 = J, K
                  SUM = SUM + TEMPL(I,K2) * INVF(K2,J)
 1460          CONTINUE
               WA(I,J) = SUM
 1480       CONTINUE
 1500    CONTINUE
C
C        compute INVF * WA'
C
         DO 1560 I = 1, K
            DO 1540 J = 1, K
               SUM = ZERO
               DO 1520 K2 = 1, I
                  SUM = SUM + INVF(I,K2) * WA(J,K2)
 1520          CONTINUE
               TEMPM(I,J) = SUM
 1540       CONTINUE
 1560    CONTINUE
C
C        update K(t)
C
         DO 1640 L = 1, R
            DO 1620 I = 1, K
               DO 1600 J = 1, K
                  SUM = ZERO
                  IF(PEARL .AND. L .GE. Q + 1) GAMMA(I, (L - 1) * K + J)
     *                                          = SUM
                  IF((.NOT.PEARL) .OR. L .LT. Q + 1) THEN
                     DO 1580 K2 = 1, K
                        SUM = SUM - TEMP(I, (L - 1) * K + K2) *
```

```
     *                        TEMPM(K2,J)
 1580                CONTINUE
                     GAMMA(I, (L - 1) * K + J) = GAMMA(I, (L - 1) * K
     *                                           + J) + SUM
                  ENDIF
 1600          CONTINUE
 1620       CONTINUE
 1640    CONTINUE
C
C        update F(t)
C
         DO 1700 I = 1, K
            DO 1680 J = I, K
               SUM = F(I,J)
               DO 1660 K2 = 1, K
                  SUM = SUM - WA(I,K2) * WA(J,K2)
 1660          CONTINUE
               F(I,J) = SUM
               F(J,I) = SUM
 1680       CONTINUE
 1700    CONTINUE
C
C        test for convergence of F(t)'s
C
         SUM = ZERO
         DO 1720 I = 1, K
            IF(QQ(I,I) .GT. ZERO) THEN
               SUM = MAX(SUM, ABS(F(I,I) - QQ(I,I)) / QQ(I,I))
            ELSE
               SUM = MAX(SUM, ABS(F(I,I) - QQ(I,I)))
            ENDIF
 1720    CONTINUE
         IF(SUM .LT. XTOL) DELTA = .TRUE.
C
         IF(.NOT. DELTA) GOTO 1920
C
C        set TEMPK = R * L'
C
         J7 = T
         DO 1780 I = 1, K
            DO 1760 J = 1, K
               DO 1740 L = 1, R
                  SUM = ZERO
                  IF(L .LE. P) SUM = PHI(I, (L - 1) * K + J)
                  IF(L .LE. Q) SUM = SUM - THETA(I, (L - 1) * K + J)
                  TEMPK(I, (L - 1) * K + J) = SUM
 1740          CONTINUE
 1760       CONTINUE
 1780    CONTINUE
C
         DO 1900 L = 1, R
            DO 1840 I = 1, K
               DO 1820 J = 1, K
                  SUM = ZERO
                  DO 1800 K2 = J, K
                     SUM = SUM + TEMPK(I, (L - 1) * K + K2) * A(K2,J)
 1800             CONTINUE
                  WA(I,J) = SUM
 1820          CONTINUE
 1840       CONTINUE
            DO 1880 I = 1, K
```

```
                  DO 1860 J = 1, K
                     TEMPK(I, (L - 1) * K + J) = WA(I,J)
 1860             CONTINUE
 1880          CONTINUE
 1900       CONTINUE
C
C           calculate choleski decomposition of F(t) and det(F(t))
C
C           copy old A onto WA
C
 1920       DO 1960 I = 1, K
               DO 1940 J = 1, K
                  WA(I,J) = A(I,J)
 1940          CONTINUE
 1960       CONTINUE
            CALL CHOL(F, K, K, A, IFAULT)
            IF(IFAULT .NE. 0) THEN
               IFAULT = 11
               RETURN
            ENDIF
            SIG = A(1,1)
            DO 1980 I = 2, K
               SIG = SIG * A(I,I)
 1980       CONTINUE
            SIG = SIG * SIG
C
            IF(DELTA) GOTO 2180
C
C           update M(t)
C
C           transpose TEMPM
C
            DO 2020 I = 1, K
               DO 2000 J = 1, I
                  SUM = TEMPM(I,J)
                  TEMPM(I,J) = TEMPM(J,I)
                  TEMPM(J,I) = SUM
 2000          CONTINUE
 2020       CONTINUE
            CALL BKSB(A, K, K, K, .FALSE., TEMPM, IFAULT)
            DO 2080 I = 1, K
               DO 2060 J = I, K
                  SUM = MT(I,J)
                  DO 2040 K2 = 1, K
                     SUM = SUM + TEMPM(K2,I) * TEMPM(K2,J)
 2040             CONTINUE
                  MT(I,J) = SUM
                  MT(J,I) = SUM
 2060          CONTINUE
 2080       CONTINUE
C
C           update L(t)
C
            CALL BKSB(WA, K, K, K, .FALSE., TEMPL, IFAULT)
            DO 2160 L = 1, R
               DO 2140 I = 1, K
                  DO 2120 J = 1, K
                     SUM = ZERO
                     IF(PEARL .AND. L .GE. Q + 1) MAT(I, (L - 1) * K + J)
     *                                              = SUM
                     IF((.NOT.PEARL) .OR. L .LT. Q + 1) THEN
```

```
                     DO 2100 K2 = 1, K
                        SUM = SUM + TEMPK(I, (L - 1) * K + K2)
     *                        * TEMPL(K2,J)
 2100                CONTINUE
                     MAT(I, (L - 1) * K + J) = TEMP(I, (L - 1) * K + J)
     *                                          - SUM
                  ENDIF
 2120          CONTINUE
 2140       CONTINUE
 2160    CONTINUE
C
 2180    DO 2200 I = 1, K
            V(I,T) = W(I,T) - Z(I)
            IF(MEAN) V(I,T) = V(I,T) - MEANS(I)
            B(I) = V(I,T)
 2200    CONTINUE
C
         CALL BKSB(A, K, K, 1, .FALSE., B, IFAULT)
         DO 2220 I = 1, K
            SSQ = SSQ + B(I) * B(I)
 2220    CONTINUE
         IF((.NOT.DELTA) .OR. T .LE. J7) DETP = DETP + LOG(SIG)
C
 2240 CONTINUE
C
      RLOGL = - CONST - (SSQ + DETP + REAL(N - J7) * LOG(TSIG)) / TWO
C
      END
C
      SUBROUTINE COVARS(K, P, Q, R, IK, PHI, THETA, QQ, GAMMA,
     *                  Z, KMAT, MAT, B, IWORK, IFAULT)
C
C     ALGORITHM AS242.3  APPL. STATIST. (1989), VOL.38, NO. 1
C
      INTEGER           P, Q, R, K, KMAT, IFAULT, M, I, J, I2, K2, J2,
     *                  KW, L, L4, IK, IWORK(KMAT)
      REAL              QQ(IK,K), PHI(IK,P*K+1), THETA(IK,Q*K+1),
     *                  GAMMA(K,K*R), Z(K*K*(R+1)), MAT(KMAT,KMAT),
     *                  B(K*K*(P+1)), SUM, ZERO, ONE
      PARAMETER         (ZERO = 0.0, ONE = 1.0)
      EXTERNAL          DECOMP, SOLVE
C
C     This auxiliary routine calculates the theoretical cross
C     covariances between the W(t)'s and between the W(t)'s and the
C     E(t)'s
C
      IFAULT = 0
C
C     generate the q gamma's
C
      DO 140 M = 1, Q
         DO 120 I = 1, K
            DO 100 J = 1, K
               SUM = ZERO
               DO 20 I2 = 1, K
                  SUM = SUM - THETA(I, (M - 1) * K + I2) * QQ(I2,J)
   20          CONTINUE
               DO 60 K2 = 1, P
                  DO 40 J2 = 1, K
                     IF(M .GT. K2) SUM = SUM + PHI(I, (K2 - 1) * K + J2)
     *                                  * GAMMA(J2, (M - K2 - 1) * K + J)
```

```
                     IF(M .EQ. K2) SUM = SUM + PHI(I, (K2 - 1) * K + J2)
     *                                     * QQ(J2,J)
      40             CONTINUE
      60          CONTINUE
                  GAMMA(I, (M - 1) * K + J) = SUM
     100       CONTINUE
     120    CONTINUE
     140 CONTINUE
C
C     calculate the first p c(j)'s
C
C     first initialise MAT and Z to zero
C
      KW = K * K * (P + 1)
      DO 180 J = 1, KW
         DO 160 I = 1, KW
            MAT(I,J) = ZERO
     160    CONTINUE
            Z(J) = ZERO
     180 CONTINUE
C
      DO 360 M = 0, P
         DO 340 I = 1, K
            DO 320 J = 1, K
               L = M * K * K + (I - 1) * K + J
C
C              first calculate right-hand side vector Z
C
               IF(M .EQ. 0) Z(L) = QQ(I,J)
               IF(M .GT. 0 .AND. M .LE. Q) THEN
                  DO 200 I2 = 1, K
                     Z(L) = Z(L) - QQ(I,I2) * THETA(J, (M - 1) * K + I2)
     200             CONTINUE
               ENDIF
               DO 260 L4 = M + 1, Q
                  DO 240 I2 = 1, K
                     Z(L) = Z(L) - GAMMA(I, (L4 - M - 1) * K + I2) *
     *                    THETA(J, (L4 - 1) * K + I2)
     240             CONTINUE
     260          CONTINUE
C
C              now set up MAT array
C
               MAT(L,L) = ONE
               DO 300 I2 = 1, P
                  DO 280 K2 = 1, K
                     IF(M .GE. I2) L4 = (M - I2) * K * K + (I - 1) * K
     *                                 + K2
                     IF(M .LT. I2) L4 = (I2 - M) * K * K + (K2 - 1)
     *                                 * K + I
                     MAT(L,L4) = MAT(L,L4) - PHI(J, (I2 - 1) * K + K2)
     280             CONTINUE
     300          CONTINUE
C
     320       CONTINUE
     340    CONTINUE
     360 CONTINUE
C
      IF(P .GT. 0) THEN
         CALL DECOMP(KW, KMAT, MAT, IWORK)
         IF(IWORK(KW) .EQ. 0) THEN
```

```
               IFAULT = 1
               RETURN
            ELSE
              CALL SOLVE(KW, KMAT, MAT, Z, IWORK)
            ENDIF
         ENDIF
C
C     calculate C(j)'s for j = p + 1,...,r (if needed)
C
         DO 540 M = P + 1, R
            DO 520 I = 1, K
               DO 500 J = 1, K
                  SUM = ZERO
                  DO 400 L4 = 1, P
                     DO 380 I2 = 1, K
                        SUM = SUM + Z((M - L4) * K * K + (I - 1) * K + I2)
     *                        * PHI(J, (L4 - 1) * K + I2)
  380                CONTINUE
  400             CONTINUE
C
                  IF(M .LE. Q) THEN
                     DO 420 I2 = 1, K
                        SUM = SUM - QQ(I,I2) * THETA(J, (M - 1) * K + I2)
  420                CONTINUE
                     DO 460 I2 = M + 1, Q
                        DO 440 L4 = 1, K
                           SUM = SUM - GAMMA(I, (I2 - M - 1) * K + L4)
     *                           * THETA(J, (I2 - 1) * K + L4)
  440                   CONTINUE
  460                CONTINUE
                  ENDIF
                  Z(M * K * K + (I - 1) * K + J) = SUM
  500          CONTINUE
  520       CONTINUE
  540    CONTINUE
C
         END
C
         SUBROUTINE CHOL(A, IK, K, L, IFAULT)
C
C     ALGORITHM AS 242.4  APPL. STATIST. (1989), VOL.38, NO. 1
C
         INTEGER          IK, K, I, J, IFAULT, K2
         REAL             A(IK,K), L(IK,K), SUM, ZERO
         PARAMETER        (ZERO = 0.0)
         INTRINSIC        SQRT
C
         IFAULT = 0
         DO 180 J = 1, K
            SUM = A(J,J)
            DO 60 K2 = 1, J - 1
               SUM = SUM - L(J,K2) * L(J,K2)
   60       CONTINUE
            IF(SUM .LE. ZERO) THEN
               IFAULT = 1
               RETURN
            ELSE
               L(J,J) = SQRT(SUM)
            ENDIF
            DO 160 I = J + 1, K
               SUM = A(I,J)
```

```
            DO 120 K2 = 1, J - 1
               SUM = SUM - L(I,K2) * L(J,K2)
  120       CONTINUE
            L(I,J) = SUM / L(J,J)
            L(J,I) = ZERO
  160    CONTINUE
  180 CONTINUE
C
      END
C
      SUBROUTINE BKSB(L, IK, K, M, UPPER, B, IFAULT)
C
C     ALGORITHM AS 242.5  APPL. STATIST. (1989), VOL.38, NO. 1
C
      INTEGER          IK, K, IFAULT, I, J, M, J2
      REAL             L(IK,K), B(IK,M), SUM, ZERO
      LOGICAL          UPPER
      PARAMETER        (ZERO = 0.0)
C
      IFAULT = 0
      IF(UPPER) THEN
         DO 21 J2 = 1, M
            DO 40 I = K, 1, -1
               SUM = B(I,J2)
               DO 20 J = I + 1, K
                  SUM = SUM - L(I,J) * B(J,J2)
   20          CONTINUE
               IF(L(I,I) .NE. ZERO) THEN
                  B(I,J2) = SUM / L(I,I)
               ELSE
                  IFAULT = 1
                  RETURN
               ENDIF
   40       CONTINUE
   21    CONTINUE
      ELSE
         DO 22 J2 = 1, M
            DO 80 I = 1, K
               SUM = B(I,J2)
               DO 60 J = 1, I - 1
                  SUM = SUM - L(I,J) * B(J,J2)
   60          CONTINUE
               IF(L(I,I) .NE. ZERO) THEN
                  B(I,J2) = SUM / L(I,I)
               ELSE
                  IFAULT = 1
                  RETURN
               ENDIF
   80       CONTINUE
   22    CONTINUE
      ENDIF
C
      END
C
      SUBROUTINE DECOMP(N, NDIM, A, IP)
      INTEGER          N, NDIM, K, KP1, M, I, J, IP(NDIM)
      REAL             A(NDIM,NDIM), T
      INTRINSIC        ABS
C
C     matrix triangularization by Gaussian elimination.
C     INPUT..
```

```
C       N = order of matrix.
C       NDIM = declared dimension of array A.
C       A = matrix to be triangularized.
C     OUTPUT..
C       A(I,J), I .LE. J = upper triangular factor, U.
C       A(I,J), I .GT. J = multipliers = lower triangular
C                          factor, I - L.
C       IP(K),  K .LT. N = index of k-th pivot row.
C       IP(N) = (-1) ** (number of interchanges) or 0.
C     use 'SOLVE' to obtain solution of linear system.
C     determ(A) = IP(N) * A(1,1) * A(2,2) * ... * A(N,N).
C     if IP(N) = 0, A is singular, 'SOLVE' will divide by zero.
C     interchanges finished in U, only partly in L.
C
      IP(N) = 1
      DO 6 K = 1, N
         IF(K .EQ. N) GOTO 5
         KP1 = K + 1
         M = K
         DO 1 I = KP1, N
            IF(ABS(A(I,K)) .GT. ABS(A(M,K))) M = I
    1    CONTINUE
         IP(K) = M
         IF(M .NE. K) IP(N) = - IP(N)
         T = A(M,K)
         A(M,K) = A(K,K)
         A(K,K) = T
         IF(T .EQ. 0.0) GOTO 5
         DO 2 I = KP1, N
    2    A(I,K) = - A(I,K) / T
         DO 4 J = KP1, N
            T = A(M,J)
            A(M,J) = A(K,J)
            A(K,J) = T
            IF(T .EQ. 0.0) GOTO 4
            DO 3 I = KP1, N
    3       A(I,J) = A(I,J) + A(I,K) * T
    4    CONTINUE
    5    IF(A(K,K) .EQ. 0.0) IP(N) = 0
    6 CONTINUE
      RETURN
      END
C
      SUBROUTINE SOLVE(N, NDIM, A, B, IP)
      INTEGER          I, N, NDIM, IP(NDIM), NM1, K, KP1, M, KM1, KB
      REAL             A(NDIM,NDIM), B(NDIM), T
C
C     solution of linear system, A * x = b.
C     INPUT..
C       N = order of matrix.
C       NDIM = declared dimension of array A.
C       A = triangularized matrix obtained from 'DECOMP'.
C       B = right-hand side vector.
C       IP = pivot vector obtained from 'DECOMP'.
C     DO NOT USE if 'DECOMP' has set IP(N) = 0.
C     OUTPUT..
C       B = solution vector, X.
C
      IF(N .EQ. 1) GOTO 9
      NM1 = N - 1
      DO 7 K = 1, NM1
```

```
      KP1 = K + 1
      M = IP(K)
      T = B(M)
      B(M) = B(K)
      B(K) = T
      DO 7 I = KP1, N
    7 B(I) = B(I) + A(I,K) * T
      DO 8 KB = 1, NM1
      KM1 = N - KB
      K = KM1 + 1
      B(K) = B(K) / A(K,K)
      T = - B(K)
      DO 8 I = 1, KM1
    8 B(I) = B(I) + A(I,K) * T
    9 B(1) = B(1) / A(1,1)
      RETURN
      END
```

```
      REAL FUNCTION TNC(T, DF, DELTA, IFAULT)
C
C     ALGORITHM AS 243  APPL. STATIST. (1989), VOL.38, NO. 1
C
C     Cumulative probability at T of the non-central t-distribution
C     with DF degrees of freedom (may be fractional) and non-centrality
C     parameter DELTA.
C
C     Note - requires the following auxiliary routines
C     ALOGAM (X)                          - ACM 291 or AS 245
C     BETAIN (X, A, B, ALBETA, IFAULT)    - AS 63 (updated in ASR 19)
C     ALNORM (X, UPPER)                   - AS 66
C
      REAL A, ALBETA, ALNRPI, B, DEL, DELTA, DF, EN, ERRBD, ERRMAX,
     *  GEVEN, GODD, HALF, ITRMAX, LAMBDA, ONE, P, Q, R2PI, RXB, S, T,
     *  TT, TWO, X, XEVEN, XODD, ZERO
      LOGICAL NEGDEL
C
C     Note - ITRMAX and ERRMAX may be changed to suit one's needs.
C
      DATA ITRMAX/100.1/, ERRMAX/1.E-06/
C
C     Constants - R2PI = 1/ {GAMMA(1.5) * SQRT(2)} = SQRT(2 / PI)
C                 ALNRPI = LN(SQRT(PI))
C
      DATA ZERO/0.0/, HALF/0.5/, ONE/1.0/, TWO/2.0/,
     *  R2PI/0.79788 45608 02865 35588/,
     *  ALNRPI/0.57236 49429 24700 08707/
C
      TNC = ZERO
      IFAULT = 2
      IF (DF .LE. ZERO) RETURN
      IFAULT = 0
C
      TT = T
      DEL = DELTA
      NEGDEL = .FALSE.
      IF (T .GE. ZERO) GO TO 1
      NEGDEL = .TRUE.
      TT = -TT
      DEL = -DEL
    1 CONTINUE
C
C     Initialize twin series (Guenther, J. Statist. Computn. Simuln.
C     vol.6, 199, 1978).
C
      EN = ONE
      X = T * T / (T* T + DF)
      IF (X .LE. ZERO) GO TO 20
      LAMBDA = DEL * DEL
      P = HALF * EXP(-HALF * LAMBDA)
      Q = R2PI * P * DEL
      S = HALF - P
      A = HALF
      B = HALF * DF
      RXB = (ONE - X) ** B
      ALBETA = ALNRPI + ALOGAM(B, IFAULT) - ALOGAM(A + B, IFAULT)
      XODD = BETAIN(X, A, B, ALBETA, IFAULT)
      GODD = TWO * RXB * EXP(A * LOG(X) - ALBETA)
      XEVEN = ONE - RXB
      GEVEN = B * X * RXB
```

```
      TNC = P * XODD + Q * XEVEN
C
C     Repeat until convergence
C
   10 A = A + ONE
      XODD = XODD - GODD
      XEVEN = XEVEN - GEVEN
      GODD = GODD * X * (A + B - ONE) / A
      GEVEN = GEVEN * X * (A + B - HALF) / (A + HALF)
      P = P * LAMBDA / (TWO * EN)
      Q = Q * LAMBDA / (TWO * EN + ONE)
      S = S - P
      EN = EN + ONE
      TNC = TNC + P * XODD + Q * XEVEN
      ERRBD = TWO * S * (XODD - GODD)
      IF (ERRBD .GT. ERRMAX .AND. EN .LE. ITRMAX) GO TO 10
C
   20 IFAULT = 1
      IF (EN .GT. ITRMAX) RETURN
      IFAULT = 0
      TNC = TNC + ALNORM(DEL, .TRUE.)
      IF (NEGDEL) TNC = ONE - TNC
C
      RETURN
      END
```

```
{
        Algorithm AS244  Appl. Statist. (1989) Vol. 38, No. 1  }

{ The procedure HiModel can be used to
     (1) Check decomposability for a hierarchical loglinear model;
     (2) Obtain the sets of variables onto which the model is collapsible;
     (3) Find the factor formula of MLEs for the model;
     (4) Find the reduced formula of the likelihood ratio test statistic. }


{ ********************************************************************

                          DATA STRUCTURES

  ******************************************************************** }

CONST  MaxVar=50;
       MaxGenerator=50;
       MaxSubGeClass=20;
       MaxInt=51;

TYPE   IntSet   = SET OF 1..MaxVar;
       SetArr   = ARRAY[1..MaxGenerator] OF IntSet;
       SetArr2  = ARRAY[1..MaxSubGeClass,1..MaxGenerator] OF IntSet;
       IntArr   = ARRAY [1..MaxSubGeClass] OF 1..MaxGenerator;
       IntGC    = 0..MaxGenerator;
       IntSubGC = 0..MaxSubGeClass;
       NonnegInt= 0..MaxInt;


{ ********************************************************************

                          PRINCIPAL PROCEDURE

  ******************************************************************** }

PROCEDURE HiModel(NumModels:                                          NonnegInt;
          M1SizeGeClass, M2SizeGeClass:                               IntGC;
          M1GeClass, M2GeClass:                                       SetArr;
       VAR M1NumNumer, M2NumNumer, M1NumDenom, M2NumDenom,
          LRNumNumer, LRNumDenom:                                     IntGC;
       VAR M1NumSubGC, M2NumSubGC, LRNumGCNumer, LRNumGCDenom:        IntSubGC;
       VAR Fault:                                                     NonnegInt;
       VAR M1Numerator, M2Numerator, M1Denominator,
          M2Denominator, LRNumerator, LRDenominator:                  SetArr;
       VAR M1SubGeClass, M2SubGeClass, LRGCNumer, LRGCDenom:          SetArr2;
       VAR M1SizeSubGC, M2SizeSubGC, LRSizeNumerGC, LRSizeDenomGC:    IntArr;
       VAR VariableSet:                                               IntSet);

VAR i,j,k,m,n: NonnegInt; VarSetModel: IntSet;

PROCEDURE Error(VAR Size: IntGC; MaxSize: IntGC; ErrorNumber: NonnegInt;
                    AnotherError: Boolean);
BEGIN  IF (Size>MaxSize) OR AnotherError THEN
         BEGIN  Fault:=ErrorNumber;  Size:=MaxSize
         END
END; { of Error }

{ Reduce the fraction: NumeratorSet/DenominatorSet. }
PROCEDURE Reduce(VAR NumNumerSet,NumDenomSet: IntGC;
                 VAR NumeratorSet, DenominatorSet: SetArr);
```

```
VAR i,j,k,m: IntGC;
BEGIN  k:=0;
 FOR i:=1 TO NumNumerSet DO
  BEGIN  j:=1;
   WHILE (NumeratorSet[i]<>DenominatorSet[j]) AND
         (j<=NumDenomSet) DO j:=j+1;
   IF j>NumDenomSet THEN
    BEGIN  k:=k+1;  NumeratorSet[k]:=NumeratorSet[i]
    END
   ELSE
    BEGIN
     FOR m:=j+1 TO NumDenomSet DO
      DenominatorSet[m-1]:=DenominatorSet[m];
     NumDenomSet:=NumDenomSet-1
    END
  END;
 NumNumerSet:=k
END; { of Reduce }

{ Check collapsibility and decomposability. }
PROCEDURE CollDeco (SizeGeClass: IntGC; GeClass: SetArr; VAR NumNumer,
  NumDenom: IntGC; VAR NumSubGC: IntSubGC; VAR Numerator,Denominator:
  SetArr; VAR SubGeClass: SetArr2; VAR SizeSubGC: IntArr);
VAR i: IntGC;

{ Check whether the generating class is decomposable or not. }
PROCEDURE Deco(VAR SizeGeClass: IntGC; VAR GeClass: SetArr);
VAR i,j: IntGC;  OneClass: IntSet;
    GeClassUnchange,GeneratorUnchange,NoSubGenerator: Boolean;
BEGIN
 REPEAT  GeClassUnchange:=true;
{ Delete variables which exist in only a single generator. }
  REPEAT  GeneratorUnchange:=true;
   FOR i:=1 TO SizeGeClass DO
    BEGIN  OneClass:=GeClass[i];
     FOR j:=1 TO SizeGeClass DO
      IF i<>j THEN OneClass:=OneClass-GeClass[j];
     IF OneClass<>[] THEN
      BEGIN
       NumNumer:=NumNumer+1;  Error(NumNumer,MaxGenerator,3,false);
       Numerator[NumNumer]:=GeClass[i];
       IF GeClass[i]-OneClass<>[] THEN
        BEGIN  NumDenom:=NumDenom+1;
         Error(NumDenom,MaxGenerator,4,false);
         Denominator[NumDenom]:=GeClass[i]-OneClass
        END;
        GeClass[i]:=GeClass[i]-OneClass;  GeneratorUnchange:=false
      END
    END
  UNTIL GeneratorUnchange;
{ Delete generators which are contained in others. }
  REPEAT  NoSubGenerator:=true;
   FOR i:=1 TO SizeGeClass DO
    FOR j:=1 TO SizeGeClass DO
     IF (i<>j) AND (GeClass[i]<=GeClass[j]) AND (GeClass[i]<>[]) THEN
      BEGIN  GeClass[i]:=[];
       NoSubGenerator:=false; GeClassUnchange:=false
      END;
{ Delete the null generators. }
   j:=0;
   FOR i:=1 TO SizeGeClass DO
```

```
    IF GeClass[i]<>[] THEN
     BEGIN j:=j+1;  GeClass[j]:=GeClass[i]
     END;
    SizeGeClass:=j
  UNTIL NoSubGenerator OR (SizeGeClass=0)
 UNTIL GeClassUnchange OR (SizeGeClass=0)
END; { of Deco }

{ Decompose generating class. }
PROCEDURE DecompGC(VAR SizeGeClass: IntGC; VAR GeClass: SetArr);
VAR i,j,k,InAGeneratorsNum,InBGeneratorsNum: IntGC; GeClassA: SetArr;
    BoundaryOfAB,VarInA,VarInB: IntSet;
    Researched: ARRAY [1..MaxGenerator] OF Boolean;
    DecomposedOrCannotDecompose,NoVarAddedInA: Boolean;
BEGIN
 Deco(SizeGeClass,GeClass);
 IF SizeGeClass<>0 THEN
  BEGIN  i:=0;  DecomposedOrCannotDecompose:=false;
{ Find variables in A. }
   REPEAT
    IF i=0 THEN BoundaryOfAB:=[] ELSE BoundaryOfAB:=GeClass[i];
    InAGeneratorsNum:=1;
    IF i<>1 THEN k:=1 ELSE k:=2;
    GeClassA[1]:=GeClass[k]; Researched[k]:=true; VarInA:=GeClassA[1];
    FOR j:=1 TO SizeGeClass DO IF j<>k THEN Researched[j]:=false;
    REPEAT  NoVarAddedInA:=true;
     FOR j:=1 TO SizeGeClass DO
      IF (i<>j) AND NOT Researched[j] AND
         (VarInA*GeClass[j]-BoundaryOfAB<>[]) THEN
       BEGIN NoVarAddedInA:=false; InAGeneratorsNum:=InAGeneratorsNum+1;
        GeClassA[InAGeneratorsNum]:=GeClass[j];  Researched[j]:=true;
        VarInA:=VarInA+GeClass[j]
       END
    UNTIL NoVarAddedInA;
    IF i<>0 THEN InBGeneratorsNum:=SizeGeClass-InAGeneratorsNum-1
            ELSE InBGeneratorsNum:=SizeGeClass-InAGeneratorsNum;
    VarInB:=[];
    IF InBGeneratorsNum<>0 THEN
{ The generating class can be decomposed by GeClass[i]. }
     BEGIN  j:=0;  DecomposedOrCannotDecompose:=true;
{ Find variables in B. }
      FOR k:=1 TO SizeGeClass DO
       IF (k<>i) AND NOT Researched[k] THEN
        BEGIN  j:=j+1; VarInB:=VarInB+GeClass[k]; GeClass[j]:=GeClass[k]
        END;
       IF i<>0 THEN
        BEGIN  InAGeneratorsNum:=InAGeneratorsNum+1;
         GeClassA[InAGeneratorsNum]:=BoundaryOfAB;
         InBGeneratorsNum:=InBGeneratorsNum+1;
         GeClass[InBGeneratorsNum]:=BoundaryOfAB*VarInB;
         NumDenom:=NumDenom+1;  Error(NumDenom,MaxGenerator,4,false);
         Denominator[NumDenom]:=GeClass[InBGeneratorsNum]
        END;
{ Decompose recursively A and B. }
      DecompGC(InAGeneratorsNum,GeClassA);
      DecompGC(InBGeneratorsNum,GeClass)
     END
    ELSE
{ The generating class cannot be decomposed by GeClass[i]. }
    IF i=SizeGeClass THEN
{ The generating class cannot be decomposed by any generator. }
```

```
      BEGIN  DecomposedOrCannotDecompose:=true; NumSubGC:=NumSubGC+1;
       IF NumSubGC>MaxSubGeClass THEN
        BEGIN  Fault:=5;  NumSubGC:=MaxSubGeClass
        END;
       SizeSubGC[NumSubGC]:=SizeGeClass;
       FOR k:=1 TO SizeGeClass DO SubGeClass[NumSubGC,k]:=GeClass[k]
      END
     ELSE i:=i+1
    UNTIL DecomposedOrCannotDecompose
   END
END; { of DecompGC }

BEGIN { of CollDeco }
 Error(SizeGeClass,MaxGenerator,1,SizeGeClass<1);  VarSetModel:=[];
 IF Fault=0 THEN
  FOR i:=1 TO SizeGeClass DO
   BEGIN
    IF NOT([1..MaxVar]>=GeClass[i]) OR (GeClass[i]=[]) THEN Fault:=2;
    VarSetModel:=VarSetModel+GeClass[i]
   END;
 NumNumer:=0;  NumDenom:=0;  NumSubGC:=0;
 IF Fault=0 THEN
  BEGIN
   DecompGC(SizeGeClass,GeClass);
   Reduce(NumNumer,NumDenom,Numerator,Denominator)
  END
END; { of CollDeco }

{ Check whether the sub-generating classes in models 1 and 2 are same. }
FUNCTION ClassesNotSame: Boolean;
VAR k: IntGC; m: NonnegInt; NotSame: Boolean;
BEGIN  k:=0;  NotSame:=M1SizeSubGC[i]<>LRSizeDenomGC[j];
 WHILE NOT NotSame AND (k<M1SizeSubGC[i]) DO
  BEGIN  k:=k+1;  m:=1;
   WHILE (m<=LRSizeDenomGC[j]) AND
         (M1SubGeClass[i,k]<>LRGCDenom[j,m]) DO m:=m+1;
   NotSame:=NotSame OR (m>LRSizeDenomGC[j])
  END;
 ClassesNotSame:=NotSame
END; { of ClassesNotSame }

BEGIN { of HiModel }
 M1NumNumer:=0; M1NumDenom:=0; M1NumSubGC:=0; VariableSet:=[];
 M2NumNumer:=0; M2NumDenom:=0; M2NumSubGC:=0; Fault:=0;
 LRNumNumer:=0; LRNumDenom:=0; LRNumGCNumer:=0; LRNumGCDenom:=0;
 IF (NumModels<1) OR (NumModels>2) THEN Fault:=6
 ELSE
  BEGIN
   CollDeco(M1SizeGeClass,M1GeClass,M1NumNumer,M1NumDenom,M1NumSubGC,
            M1Numerator,M1Denominator,M1SubGeClass,M1SizeSubGC);
   VariableSet:=VarSetModel;
   IF NumModels>1 THEN
    BEGIN
     CollDeco(M2SizeGeClass,M2GeClass,M2NumNumer,M2NumDenom,M2NumSubGC,
              M2Numerator,M2Denominator,M2SubGeClass,M2SizeSubGC);
     IF VariableSet<>VarSetModel THEN Fault:=9;
{ Find the likelihood ratio: M1/M2. }
     LRNumNumer:=M1NumNumer+M2NumDenom;
     Error(LRNumNumer,MaxGenerator,7,false);
     FOR i:=1 TO LRNumNumer DO
      IF i<=M1NumNumer THEN LRNumerator[i]:=M1Numerator[i]
```

```
          ELSE LRNumerator[i]:=M2Denominator[i-M1NumNumer];
      LRNumDenom:=M2NumNumer+M1NumDenom;
      Error(LRNumDenom,MaxGenerator,8,false);
      FOR i:=1 TO LRNumDenom DO
       IF i<=M1NumDenom THEN LRDenominator[i]:=M1Denominator[i]
          ELSE LRDenominator[i]:=M2Numerator[i-M1NumDenom];
      Reduce(LRNumNumer,LRNumDenom,LRNumerator,LRDenominator);
  { Reduce sub-generating classes of M1 and M2. }
      LRNumGCDenom:=M2NumSubGC;
      FOR m:=1 TO LRNumGCDenom DO
       BEGIN  LRSizeDenomGC[m]:=M2SizeSubGC[m];
        FOR n:=1 TO LRSizeDenomGC[m] DO LRGCDenom[m,n]:=M2SubGeClass[m,n]
       END;
      k:=0;
      FOR i:=1 TO M1NumSubGC DO
       BEGIN  j:=1;
        WHILE (j<=LRNumGCDenom) AND ClassesNotSame DO j:=j+1;
        IF j>LRNumGCDenom THEN
         BEGIN  k:=k+1;  LRSizeNumerGC[k]:=M1SizeSubGC[i];
          FOR m:=1 TO LRSizeNumerGC[k] DO
           LRGCNumer[k,m]:=M1SubGeClass[i,m]
         END
        ELSE
         BEGIN
          FOR m:=j+1 TO LRNumGCDenom DO
           BEGIN  LRSizeDenomGC[m-1]:=LRSizeDenomGC[m];
            FOR n:=1 TO LRSizeDenomGC[m-1] DO
             LRGCDenom[m-1,n]:=LRGCDenom[m,n]
           END;
          LRNumGCDenom:=LRNumGCDenom-1
         END
       END;
      LRNumGCNumer:=k
     END
   END
  END; { of HiModel }
```

```
c This file contains two algorithms for the logarithm of the gamma function.
c Algorithm AS 245 is the faster (but longer) and gives an accuracy of about
c 10-12 significant decimal digits except for small regions around X = 1 and
c X = 2, where the function goes to zero.
c The second algorithm is not part of the AS algorithms.   It is slower but
c gives 14 or more significant decimal digits accuracy, except around X = 1
c and X = 2.   The Lanczos series from which this algorithm is derived is
c interesting in that it is a convergent series approximation for the gamma
c function, whereas the familiar series due to De Moivre (and usually wrongly
c called Stirling's approximation) is only an asymptotic approximation, as
c is the true and preferable approximation due to Stirling.
c
c
c
      DOUBLE PRECISION FUNCTION ALNGAM(XVALUE, IFAULT)
C
C     ALGORITHM AS245  APPL. STATIST. (1989) VOL. 38, NO. 2
C
C     Calculation of the logarithm of the gamma function
C
      INTEGER IFAULT
      DOUBLE PRECISION ALR2PI, FOUR, HALF, ONE, ONEP5, R1(9), R2(9),
     +          R3(9), R4(5), TWELVE, X, X1, X2, XLGE, XLGST, XVALUE,
     +          Y, ZERO
C
C     Coefficients of rational functions
C
      DATA R1/-2.66685 51149 5D0, -2.44387 53423 7D1,
     +        -2.19698 95892 8D1,  1.11667 54126 2D1,
     +         3.13060 54762 3D0,  6.07771 38777 1D-1,
     +         1.19400 90572 1D1,  3.14690 11574 9D1,
     +         1.52346 87407 0D1/
      DATA R2/-7.83359 29944 9D1, -1.42046 29668 8D2,
     +         1.37519 41641 6D2,  7.86994 92415 4D1,
     +         4.16438 92222 8D0,  4.70668 76606 0D1,
     +         3.13399 21589 4D2,  2.63505 07472 1D2,
     +         4.33400 02251 4D1/
      DATA R3/-2.12159 57232 3D5,  2.30661 51061 6D5,
     +         2.74647 64470 5D4, -4.02621 11997 5D4,
     +        -2.29660 72978 0D3, -1.16328 49500 4D5,
     +        -1.46025 93751 1D5, -2.42357 40962 9D4,
     +        -5.70691 00932 4D2/
      DATA R4/ 2.79195 31791 8525D-1, 4.91731 76105 05968D-1,
     +         6.92910 59929 1889D-2, 3.35034 38150 22304D0,
     +         6.01245 92597 64103D0/
C
C     Fixed constants
C
      DATA ALR2PI/9.18938 53320 4673D-1/, FOUR/4.D0/, HALF/0.5D0/,
     +     ONE/1.D0/, ONEP5/1.5D0/, TWELVE/12.D0/, ZERO/0.D0/
C
C     Machine-dependant constants.
C     A table of values is given at the top of page 399 of the paper.
C     These values are for the IEEE double-precision format for which
C     B = 2, t = 53 and U = 1023 in the notation of the paper.
C
      DATA XLGE/5.10D6/, XLGST/1.D+305/
C
      X = XVALUE
      ALNGAM = ZERO
C
```

```
C     Test for valid function argument
C
      IFAULT = 2
      IF (X .GE. XLGST) RETURN
      IFAULT = 1
      IF (X .LE. ZERO) RETURN
      IFAULT = 0
C
C     Calculation for 0 < X < 0.5 and 0.5 <= X < 1.5 combined
C
      IF (X .LT. ONEP5) THEN
        IF (X .LT. HALF) THEN
          ALNGAM = -LOG(X)
          Y = X + ONE
C
C     Test whether X < machine epsilon
C
          IF (Y .EQ. ONE) RETURN
        ELSE
          ALNGAM = ZERO
          Y = X
          X = (X - HALF) - HALF
        END IF
        ALNGAM = ALNGAM + X * ((((R1(5)*Y + R1(4))*Y + R1(3))*Y
     +                + R1(2))*Y + R1(1)) / ((((Y + R1(9))*Y + R1(8))*Y
     +                + R1(7))*Y + R1(6))
        RETURN
      END IF
C
C     Calculation for 1.5 <= X < 4.0
C
      IF (X .LT. FOUR) THEN
        Y = (X - ONE) - ONE
        ALNGAM = Y * ((((R2(5)*X + R2(4))*X + R2(3))*X + R2(2))*X
     +              + R2(1)) / ((((X + R2(9))*X + R2(8))*X + R2(7))*X
     +              + R2(6))
        RETURN
      END IF
C
C     Calculation for 4.0 <= X < 12.0
C
      IF (X .LT. TWELVE) THEN
        ALNGAM = ((((R3(5)*X + R3(4))*X + R3(3))*X + R3(2))*X + R3(1)) /
     +            ((((X + R3(9))*X + R3(8))*X + R3(7))*X + R3(6))
        RETURN
      END IF
C
C     Calculation for X >= 12.0
C
      Y = LOG(X)
      ALNGAM = X * (Y - ONE) - HALF * Y + ALR2PI
      IF (X .GT. XLGE) RETURN
      X1 = ONE / X
      X2 = X1 * X1
      ALNGAM = ALNGAM + X1 * ((R4(3)*X2 + R4(2))*X2 + R4(1)) /
     +             ((X2 + R4(5))*X2 + R4(4))
      RETURN
      END
C
C
C
```

```
c
      double precision function lngamma(z, ier)
c
c     Uses Lanczos-type approximation to ln(gamma) for z > 0.
c     Reference:
c          Lanczos, C. 'A precision approximation of the gamma
c                  function', J. SIAM Numer. Anal., B, 1, 86-96, 1964.
c     Accuracy: About 14 significant digits except for small regions
c               in the vicinity of 1 and 2.
c
c     Programmer: Alan Miller
c                 CSIRO Division of Mathematics & Statistics
c
c     N.B. It is assumed that the Fortran compiler supports long
c          variable names, including the underline character.   Some
c          compilers will not accept the 'implicit none' statement
c          below.
c
c     Latest revision - 17 April 1988
c
      implicit none
      double precision a(9), z, lnsqrt2pi, tmp
      integer ier, j
      data a/0.9999999999995183d0, 676.5203681218835d0,
     +       -1259.139216722289d0, 771.3234287757674d0,
     +       -176.6150291498386d0, 12.50734324009056d0,
     +       -0.1385710331296526d0, 0.9934937113930748d-05,
     +       0.1659470187408462d-06/
c
      data lnsqrt2pi/0.91893 85332 04672 7d0/
c
      if (z .le. 0.d0) then
        ier = 1
        return
      end if
      ier = 0
c
      lngamma = 0.d0
      tmp = z + 7.d0
      do 10 j = 9, 2, -1
        lngamma = lngamma + a(j)/tmp
        tmp = tmp - 1.d0
   10 continue
      lngamma = lngamma + a(1)
      lngamma = log(lngamma) + lnsqrt2pi - (z+6.5d0) +
     +                              (z-0.5d0)*log(z+6.5d0)
      end
```

```
C---- SUBROUTINE ARAVSS ------------------------------------------------
C
C  Computes sufficient statistics for analysis of variance table when
C  the errors follow AR(1) normal process, given the design matrix X(N,P),
C  observation matrix Y(N,T) and assuming that XXINV(P,P) contains
C  the inverse matrix of Xtranspose*X.
C
C-----------------------------------------------------------------------
      SUBROUTINE ARAVSS(Y,N,T,P,QS,XD1,XD2,YBAR,X,XXINV,IFAULT)
C
C     ALGORITHM AS 246.1  APPL. STATIST. (1989), VOL.38, NO. 2
C
      INTEGER T,P
      REAL Y(N,T),X(N,P),DLIN(2),DQUAD(3),YBAR(T)
      REAL XD1(P),XD2(P),XXINV(P,P),QS(14),ZERO,PROD
      DATA ZERO/0.0/
C
C      Computes admissibility of parameters
C
      IFAULT=0
      IF(P.LT.1) IFAULT=1
      IF(T.LT.3) IFAULT=IFAULT+2
      IF(N.LT.P) IFAULT=IFAULT+4
      J=0
      DO 10 I=1,N
  10    IF(X(I,1).NE.1.0) J=1
      IF(J.NE.0) IFAULT=IFAULT+8
      IF(IFAULT.GT.0) RETURN
C
C      Sets initial values to zero
C
      DO 20 I=1,14
  20    QS(I)=ZERO
      DO 30 I=1,P
        XD1(I)=ZERO
        XD2(I)=ZERO
  30    CONTINUE
C
C      For each individual time series, sufficient
C      statistics are computed and accumulated
C
      DO 60 I=1,N
        DO 40 J=1,T
  40      YBAR(J)=Y(I,J)
        CALL ARSUFF(YBAR,T,1,DLIN,DQUAD,IFAULT)
        QS(1)=QS(1)+DQUAD(1)
        QS(2)=QS(2)+DQUAD(2)
        QS(3)=QS(3)+DQUAD(3)
        QS(4)=QS(4)+DLIN(1)**2
        QS(5)=QS(5)+DLIN(1)*DLIN(2)
        QS(6)=QS(6)+DLIN(2)**2
        DO 50 J=1,P
          XD1(J)=XD1(J)+X(I,J)*DLIN(1)
          XD2(J)=XD2(J)+X(I,J)*DLIN(2)
  50      CONTINUE
  60    CONTINUE
C
C      Computes d(i)'X(X'X)**(-1)X'd(j) for i,j=1,2
C
      QS(7)=PROD(XD1,XXINV,XD1,P)
      QS(8)=PROD(XD1,XXINV,XD2,P)
```

```
      QS(9)=PROD(XD2,XXINV,XD2,P)
C
C      Computing column means and their sufficient statistics
C
      DO 80 J=1,T
        YBAR(J)=ZERO
        DO 70 I=1,N
  70      YBAR(J)=YBAR(J)+Y(I,J)
        YBAR(J)=YBAR(J)/N
  80    CONTINUE
      CALL ARSUFF(YBAR,T,1,DLIN,DQUAD,IFAULT)
      QS(10)=DQUAD(1)
      QS(11)=DQUAD(2)
      QS(12)=DQUAD(3)
      QS(13)=DLIN(1)
      QS(14)=DLIN(2)
      RETURN
      END


C---- SUBROUTINE ARAVLK -------------------------------------------
C
C  Computes the log-likelihood and the components of the analysis of
C  variance table supposing that the errors follow an AR(1) stationary
C  normal process, and that the autoregressive parameter is equal to
C  alpha. The sufficient statistics must have been previously computed
C  using SUBROUTINE ARAVSS.
C
C-----------------------------------------------------------------------
      SUBROUTINE ARAVLK(ALPHA,N,T,P,QS,XD1,XD2,XXINV,LOGLIK,GM,
     +                  BETA,SE,SIGMA2,PSI,SS,DF,RANDOM,TIME,IFAULT)
C
C     ALGORITHM AS 246.2  APPL. STATIST. (1989), VOL.38, NO. 2
C
      INTEGER T,P,DF(6),DFR
      REAL LOGLIK,SS(6),BETA(P),SE(P)
      REAL XD1(P),XD2(P),XXINV(P,P),QS(14),DELTA,RSS,DA,HA,BA
      LOGICAL RANDOM,TIME
      DELTA(B11,B12,B22)=B11-2.0*ALPHA*B12+ALPHA*ALPHA*B22
C
C      Checks whether alpha is between -1 and 1
C
      IF(ALPHA**2.LT.1.0) THEN
        IFAULT=0
      ELSE
        IFAULT=1
        RETURN
        ENDIF
C
      FN=FLOAT(N)
      TAU=T-(T-2)*ALPHA
      CTAU=(1.0-ALPHA)/TAU
C
C      Computes D(alpha), H(alpha), B(alpha)
C
      DA=DELTA(QS(1),QS(2),QS(3))
      HA=DELTA(QS(4),QS(5),QS(6))
      BA=DELTA(QS(7),QS(8),QS(9))
      RSS=DA
      DFR=N*T
C
```

```
C      Subtracts required sum of squares from RSS
C      and computes profile log-likelihood
C
       GM=(QS(13)-ALPHA*QS(14))/TAU
       SS(1)=CTAU*FN*(GM*TAU)**2
       DF(1)=1
       SS(6)=DA
       DF(6)=N*T
       DO 10 I=2,5
         SS(I)=0.0
         DF(I)=0
  10     CONTINUE
       IF(P.GT.1) THEN
         SS(2)=CTAU*BA-SS(1)
         DF(2)=P-1
         ENDIF
C
C      Branch for time effect
C
       IF(TIME) THEN
         SS(4)=FN*DELTA(QS(10),QS(11),QS(12))-SS(1)
         DF(4)=T-1
         RSS=RSS-SS(4)
         DFR=DFR-DF(4)
         ENDIF

       IF(RANDOM) THEN
C
C          Random effect model
C
         SS(3)=(HA-BA)*CTAU
         DF(3)=N-P
         PSI=SS(3)/FN
         RSS=RSS-CTAU*HA
         DFR=DFR-N
         SIGMA2=RSS/((T-1)*FN)
         IF(PSI.LT.SIGMA2) THEN
           SIGMA2=(RSS+CTAU*(HA-BA))/(FN*T)
           PSI=SIGMA2
           ENDIF
         LOGLIK=(T-1)*ALOG(SIGMA2)+ALOG(PSI)-ALOG(1.0-ALPHA**2)
       ELSE
C
C          Fixed effect model
C
         RSS=RSS-BA*CTAU
         DFR=DFR-P
         SIGMA2=RSS/(N*T)
         PSI=SIGMA2
         LOGLIK=T*ALOG(SIGMA2)-ALOG(1.0-ALPHA**2)
         ENDIF

       SS(5)=RSS
       DF(5)=DFR
       LOGLIK=-0.5*FN*LOGLIK
C
C      Computes estimates of BETA()
C
       DO 20 I=1,P
         BETA(I)=0.0
         SE(I)=(XD1(I)-ALPHA*XD2(I))/TAU
```

```
   20    CONTINUE
         DO 40 I=1,P
           DO 30 J=1,P
   30      BETA(I)=BETA(I)+XXINV(I,J)*SE(J)
   40    CONTINUE
C
C       Computes standard errors for BETA()
C
         U=PSI/((1.0-ALPHA)*TAU)
         DO 50 I=1,P
   50    SE(I)=SQRT(U*XXINV(I,I))
         RETURN
         END


C---- SUBROUTINE ARSUFF ------------------------------------------------
C
C  Computes sufficient statistics for stationary autoregressive
C  normal process Y(1),...,Y(T) of order ORDER.
C  The output values are stored in vector DLIN (linear terms) and
C  DQUAD (quadratic terms, packed upper triangular).
C  A. Azzalini (Dipart. di Statistica, Universita` di Padova) Feb.1982
C-----------------------------------------------------------------------
         SUBROUTINE ARSUFF(Y,T,ORDER,DLIN,DQUAD,IFAULT)
C
C     ALGORITHM AS 246.3  APPL. STATIST. (1989), VOL.38, NO. 2
C
         INTEGER T,ORDER,R,S,SMR
         REAL Y(T),DLIN(ORDER+1),DQUAD((ORDER+1)*(ORDER+2)/2)
         IFAULT=0
         IF(ORDER.LT.0) IFAULT=1
         IF(T.LE.2*ORDER) IFAULT=IFAULT+2
         IF(IFAULT.GT.0) RETURN
         M=ORDER+1
         N=M*(ORDER+2)/2
         SUM1=0.
         SUM2=0.
         DO 10 J=M,T-ORDER
           YJ=Y(J)
           SUM1=SUM1+YJ
           SUM2=SUM2+YJ*YJ
   10    CONTINUE
         DLIN(M)=SUM1
         DQUAD(N)=SUM2
         DO 20 K=1,ORDER
           R=ORDER-K+1
           YJ=Y(R)
           SUM1=SUM1+YJ
           SUM2=SUM2+YJ*YJ
           YJ=Y(T-R+1)
           SUM1=SUM1+YJ
           SUM2=SUM2+YJ*YJ
           DLIN(R)=SUM1
           DQUAD(R*(R+1)/2)=SUM2
   20    CONTINUE
         DO 50 SMR=1,ORDER
           SUM2=0.
           DO 30 J=M,T-ORDER+SMR
             SUM2=SUM2+Y(J)*Y(J-SMR)
   30      CONTINUE
           DQUAD(N-SMR)=SUM2
```

```
          DO 40 R=ORDER-SMR,1,-1
            S=R+SMR
            SUM2=SUM2+Y(R)*Y(S)+Y(T-R+1)*Y(T-S+1)
            DQUAD(S*(S-1)/2+R)=SUM2
  40        CONTINUE
  50      CONTINUE
        RETURN
        END

C-------------------------------------------------------------
C
        FUNCTION PROD(XD1,XX,XD2,M)
        REAL A,ZERO,XD1(M),XD2(M),XX(M,M)
        DATA ZERO/0.0/
        PROD=ZERO
        DO 20 I=1,M
          A=ZERO
          DO 10 J=1,M
  10        A=A+XX(I,J)*XD2(J)
  20      PROD=PROD+XD1(I)*A
        RETURN
        END
```

```
FUNCTION EdgeOperation(VAR Graph : GraphType; m, n : NodeType;
                       NodeSet : CliqueType; Add : boolean) : integer;

{Algorithm AS 247.1  Appl. Statist. (1989) Vol.38, No.2}

VAR
    exist      : boolean;
    error      : integer;
    i          : integer;

BEGIN
exist := false;
WITH Graph DO
    FOR i := 1 TO NoOfCliques DO
        IF ([m,n] <= cliques[i]) THEN
            exist := true;
error := 0;

{ Check for invalid inputs.}
IF NOT (m IN NodeSet) OR NOT (n IN NodeSet) THEN
    m := n;
IF m = n THEN
    error := 1
ELSE
    IF exist AND Add THEN
        error := 2
    ELSE
        IF NOT exist AND NOT Add THEN
            error := 3
        ELSE
            {If input ok then perform operation.}
            IF Add THEN
                AddEdge(Graph,m,n)
            ELSE
                DeleteEdge(Graph,m,n);
EdgeOperation := error;
END; { of EdgeOperation }



 PROCEDURE AddEdge (VAR Graph : GraphType;
         m, n : NodeType);

 {Algorithm AS 247.2  Appl. Statist. (1989) Vol.38, No.2}

    { This procedure replaces the cliques of the current graph held in the
      structure cliques of type GraphType by the cliques of the graph
      obtained by adding edge m-n.}

  VAR
   mSubgraph, nSubgraph, mnSubgraph : GraphType;
   ThisClique, mClique, nClique, mnClique : CliqueType;
   i, j, k, NoOfCliques : CliqueIndex;
   mCliqueOK, nCliqueOK, mnCliqueOK : ARRAY[1..Cliquemax] OF Boolean;

 BEGIN
  NoOfCliques := Graph.NoOfCliques;
  Graph.NoOfCliques := 0;
  mSubgraph.NoOfCliques := 0;
  nSubgraph.NoOfCliques := 0;
  mnSubgraph.NoOfCliques := 0;
```

```
    { 1. Scan through the cliques of the current graph in Graph. if a
      clique contains neither node of the edge, copy it as a clique of the
      new graph back to Graph. If it contains one of the nodes copy it as
      one of the temporary cliques to mSubgraph or nSubgraph setting the
      associated flag true.}

  FOR i := 1 TO NoOfCliques DO
   BEGIN
    ThisClique := Graph.Cliques[i];
    IF m IN ThisClique THEN
     WITH mSubgraph DO
      BEGIN
       NoOfCliques := NoOfCliques + 1;
       Cliques[NoOfCliques] := ThisClique;
       mCliqueOK[NoOfCliques] := true
      END
    ELSE IF n IN ThisClique THEN
     WITH nSubgraph DO
      BEGIN
       NoOfCliques := NoOfCliques + 1;
       Cliques[NoOfCliques] := ThisClique;
       nCliqueOK[NoOfCliques] := true
      END
    ELSE
     WITH Graph DO
      BEGIN
       NoOfCliques := NoOfCliques + 1;
       Cliques[NoOfCliques] := ThisClique
      END
   END;

    { 2.  Compare each clique from mSubgraph with each clique from
      nSubgraph in turn, generate the clique created from the intersection
      of the pair combined with the nodes m and n (It will, most often, be
      simply the edge m-n) and put it into mnSubgraph, setting the
      associated flag true. Compare both of the creating cliques with the
      created one and if either is a subset of it, set the associated flag
      false.  Compare the created one with all previous created ones;  if
      it is a subset of any  of them remove it; if any of them is a subset
      of it set their associated flags false. }

  FOR i := 1 TO mSubgraph.NoOfCliques DO
   BEGIN
    mClique := mSubgraph.Cliques[i];
    FOR j := 1 TO nSubgraph.NoOfCliques DO
     BEGIN
      nClique := nSubgraph.Cliques[j];
      WITH mnSubgraph DO
       BEGIN
        mnClique := mClique * nClique + [m, n];
        IF mClique <= mnClique THEN
         mCliqueOK[i] := false;
        IF nClique <= mnClique THEN
         nCliqueOK[j] := false;
        NoOfCliques := NoOfCliques + 1;
        Cliques[NoOfCliques] := mnClique;
        mnCliqueOK[NoOfCliques] := true;
        k := 1;
        WHILE NOT (mnClique <= Cliques[k]) DO
         BEGIN
```

```
          IF mnClique >= Cliques[k] THEN
           mnCliqueOK[k] := false;
          k := k + 1
         END;
        IF k <> NoOfCliques THEN
         NoOfCliques := NoOfCliques - 1
       END
     END
  END;

   { 3.  Scan mnSubgraph, mSubgraph and nSubgraph in turn, copying to
     Graph each clique whose associated flag is still set true.}

 WITH Graph DO
  BEGIN
   FOR i := 1 TO mnSubgraph.NoOfCliques DO
    IF mnCliqueOK[i] THEN
     BEGIN
      NoOfCliques := NoOfCliques + 1;
      Cliques[NoOfCliques] := mnSubgraph.Cliques[i]
     END;
   FOR i := 1 TO mSubgraph.NoOfCliques DO
    IF mCliqueOK[i] THEN
     BEGIN
      NoOfCliques := NoOfCliques + 1;
      Cliques[NoOfCliques] := mSubgraph.Cliques[i]
     END;
   FOR i := 1 TO nSubgraph.NoOfCliques DO
    IF nCliqueOK[i] THEN
     BEGIN
      NoOfCliques := NoOfCliques + 1;
      Cliques[NoOfCliques] := nSubgraph.Cliques[i]
     END
  END
END;{of AddEdge }


PROCEDURE DeleteEdge (VAR Graph : GraphType;
        m, n : NodeType);

{Algorithm AS 247.3  Appl. Statist. (1989) Vol.38, No.2}

   { This procedure replaces the cliques of the current graph held in the
     structure cliques of type GraphType by the cliques of the graph
     obtained by deleting edge m-n.                                    }

 VAR
  Subgraph : GraphType;
  ThisClique : CliqueType;
  i, j, NoOfCliques, buffer : CliqueIndex;

BEGIN
 NoOfCliques := Graph.NoOfCliques;
 Graph.NoOfCliques := 0;
 Subgraph.NoOfCliques := 0;

   { 1. Scan through the cliques of the current graph held in Graph. For
     each clique containing the edge m-n, store in Subgraph the two
     (possibly redundant) cliques obtained by deleting each node of the
     edge. Copy each clique not containing the edge back to Graph as a
     clique  of the new graph.                                         }
```

```
 FOR i := 1 TO NoOfCliques DO
  BEGIN
   ThisClique := Graph.Cliques[i];
   IF [m, n] <= ThisClique THEN
    WITH Subgraph DO
     BEGIN
      NoOfCliques := NoOfCliques + 2;
      Cliques[NoOfCliques - 1] := ThisClique - [m];
      Cliques[NoOfCliques] := ThisClique - [n];
     END
   ELSE
    WITH Graph DO
     BEGIN
      NoOfCliques := NoOfCliques + 1;
      Cliques[NoOfCliques] := ThisClique
     END
  END;

   { 2. Compare each clique in Subgraph with those directly placed in
     Graph.  If it is a subset of any clique in Graph it is ignored,
     otherwise it is added to Graph. The searching process uses a
     variation of the technique of adding the item as a sentinel at the
     end of the sequence being searched to simplify the loop termination.
     Here, since Graph is being extended but the searching takes place
     only over its initial part, a buffer is left at the interface to be
     filled in just before exit.  }

 WITH Graph DO
  BEGIN
   NoOfCliques := NoOfCliques + 1;
   buffer := NoOfCliques;
   FOR i := 1 TO Subgraph.NoOfCliques DO
    BEGIN
     ThisClique := Subgraph.Cliques[i];
     Cliques[buffer] := ThisClique;
     j := 1;
     WHILE NOT (ThisClique <= Cliques[j]) DO
      j := j + 1;
     IF j = buffer THEN
      BEGIN
       NoOfCliques := NoOfCliques + 1;
       Cliques[NoOfCliques] := ThisClique
      END
    END;
   Cliques[buffer] := Cliques[NoOfCliques];
   NoOfCliques := NoOfCliques - 1
  END
END;{of DeleteEdge }
```

```
      SUBROUTINE EDFGOF(X, N, ITEST, Z, XBAR, S2, D, V, W2, W2P, U2,
     +   U2P, A2, A2P, IFAULT)
C
C     ALGORITHM AS 248.1  APPL. STATIST. (1989), VOL.38, NO. 3
C
C     Tests a sample for uniformity, normality or exponentiality
C     using goodness-of-fit statistics based on the empirical
C     distribution function
C
C     Auxiliary routines called: ALNORM from AS 66
C
      INTEGER I, IFAULT, ITEST, J, N
      REAL  X(N), Z(N), XBAR, S2, D, V, W2, W2P, U2, U2P, A2, A2P, RN,
     +      ROOTN, RN2, S, ZERO, PT01, PT05, PT1, PT11, PT12, PT155,
     +      PT16, PT2, PT24, PT26, PT3, PT35, PT4, PT5, PT6, PT75, PT8,
     +      PT82, PT85, ONE, TWOP25, WCUT2(3), WCOEF2(4, 3), WCUT3(3),
     +      WCOEF3(4, 3), UCUT2(3), UCOEF2(4, 3), UCUT3(3),
     +      UCOEF3(4, 3), ACUT2(3), ACOEF2(4, 3), ACUT3(3), ACOEF3(4, 3)
      REAL ALNORM, PVALUE
      DATA ZERO, PT01, PT05, PT1, PT11, PT12, PT155, PT16, PT2, PT24
     +     /0.0, 0.01, 0.05, 0.1, 0.11, 0.12, 0.155, 0.16, 0.2, 0.24/
      DATA PT26, PT3, PT35, PT4, PT5, PT6, PT75, PT8, PT82, PT85, ONE
     +     /0.26, 0.3, 0.35, 0.4, 0.5, 0.6, 0.75, 0.8, 0.82, 0.85, 1.0/
      DATA TWOP25 /2.25/
      DATA WCUT2, WCUT3, UCUT2, UCUT3, ACUT2, ACUT3 /0.0275, 0.051,
     +     0.092, 0.035, 0.074, 0.160, 0.0262, 0.048, 0.094, 0.029,
     +     0.062, 0.120, 0.200, 0.340, 0.600, 0.260, 0.510, 0.950/
      DATA ((WCOEF2(I,J), J=1,3), I=1,4) / -13.953, 775.5, -12542.61,
     +     -5.903, 179.546, -1515.29, 0.886, -31.62, 10.897, 1.111,
     +     -34.242, 12.832/
      DATA ((WCOEF3(I,J), J=1,3), I=1,4) / -11.334, 459.098, -5652.1,
     +     -5.779, 132.89, -866.58, 0.586, -17.87, 7.417, 0.447,
     +     -16.592, 4.849/
      DATA ((UCOEF2(I,J), J=1,3), I=1,4) / -13.642, 766.31, -12432.74,
     +     -6.3328, 214.57, -2022.28, 0.8510, -32.006, -3.45, 1.325,
     +     -38.918, 16.45/
      DATA ((UCOEF3(I,J), J=1,3), I=1,4) / -11.703, 542.5, -7574.59,
     +     -6.3288, 178.1, -1399.49, 0.8071, -25.166, 8.44, 0.7663,
     +     -24.359, 4.539/
      DATA ((ACOEF2(I,J), J=1,3), I=1,4) / -13.436, 101.14, -223.73,
     +     -8.318, 42.796, -59.938, 0.9177, -4.279, -1.38, 1.2937,
     +     -5.709, 0.0186/
      DATA ((ACOEF3(I,J), J=1,3), I=1,4) / -12.2204, 67.459, -110.3,
     +     -6.1327, 20.218, -18.663, 0.9209, -3.353, 0.3, 0.731, -3.009,
     +     0.15/
C
      IFAULT = 1
      IF (ITEST .LT. 1 .OR. ITEST .GT. 3) RETURN
      IFAULT = 2
      IF (N .LT. 2 .OR. (ITEST .EQ. 2 .AND. N .EQ. 2)) RETURN
      IFAULT = 3
      DO 10 I = 2, N
        IF (X(I) .LT. X(I-1)) RETURN
   10 CONTINUE
      RN = N
      ROOTN = SQRT(RN)
      RN2 = RN * RN
      IF (ITEST .EQ. 1) THEN
C
C     Test for uniformity (F(x) is completely specified)
C
```

```
          IFAULT = 0
          CALL STATS(X, N, D, V, W2, U2, A2, IFAULT)
          IF (IFAULT .NE. 0) RETURN
C
C     Modifications when F(x) is completely specified
C
          D = D * (ROOTN + PT12 + PT11 / ROOTN)
          V = V * (ROOTN + PT155 + PT24 / ROOTN)
          W2 = (W2 - PT4 / RN + PT6 / RN2) * (ONE + ONE / RN)
          U2 = (U2 - PT1 / RN + PT1 / RN2) * (ONE + PT8 / RN)
          RETURN
        ELSE
C
C     Estimate the mean XBAR
C
          XBAR = ZERO
          DO 20 I = 1, N
            XBAR = XBAR + X(I)
   20     CONTINUE
          XBAR = XBAR / RN
          IF (ITEST .EQ. 2) THEN
C
C     Test for normality (MU and SIGMA**2 unspecified).
C     First estimate the variance S**2
C
            IFAULT = 5
            S2 = ZERO
            DO 30 I = 1, N
              S2 = S2 + (X(I) - XBAR)**2
   30       CONTINUE
            S2 = S2 / (RN - ONE)
            IF (S2 .LE. ZERO) RETURN
C
C     Compute Z(I) = F(XI - XBAR)/S)
C
            S = SQRT(S2)
            DO 40 I = 1, N
              Z(I) = ALNORM((X(I) - XBAR)/S, .FALSE.)
   40       CONTINUE
            CALL STATS(Z, N, D, V, W2, U2, A2, IFAULT)
            IF (IFAULT .NE. 0) RETURN
C
C     Modifications when F(x) is the normal distribution
C
            D = D * (ROOTN - PT01 + PT85 / ROOTN)
            V = V * (ROOTN + PT05 + PT82 / ROOTN)
            W2 = W2 * (ONE + PT5 / RN)
            W2P = PVALUE(W2, WCUT2, WCOEF2)
            U2 = U2 * (ONE + PT5 / RN)
            U2P = PVALUE(U2, UCUT2, UCOEF2)
            A2 = A2 * (ONE + PT75 / RN + TWOP25 / RN2)
            A2P = PVALUE(A2, ACUT2, ACOEF2)
            RETURN
          ELSE
C
C     Test for exponentiality (scale parameter unspecified)
C
            IFAULT = 6
            IF (XBAR .LE. ZERO) RETURN
            DO 50 I = 1, N
              Z(I) = ONE - EXP(-X(I) / XBAR)
```

```
   50     CONTINUE
          CALL STATS(Z, N, D, V, W2, U2, A2, IFAULT)
          IF (IFAULT .NE. 0) RETURN
C
C     Modifications when F(x) is the exponential distribution
C
          D = (D - PT2 / RN) * (ROOTN + PT26 + PT5 / ROOTN)
          V = (V - PT2 / RN) * (ROOTN + PT24 + PT35 / ROOTN)
          W2 = W2 * (ONE + PT16 / RN)
          W2P = PVALUE(W2, WCUT3, WCOEF3)
          U2 = U2 * (ONE + PT16 / RN)
          U2P = PVALUE(U2, UCUT3, UCOEF3)
          A2 = A2 * (ONE + PT3 / RN)
          A2P = PVALUE(A2, ACUT3, ACOEF3)
          RETURN
        END IF
      END IF
      END
C
C
      SUBROUTINE STATS(Z, N, D, V, W2, U2, A2, IFAULT)
C
C     ALGORITHM AS 248.2  APPL. STATIST. (1989), VOL.38, NO. 3
C
C     Computes the goodness-of-fit statistics D, V, W**2, U**2 and
C     A**2 from the transformed Z values
C
      INTEGER I, IFAULT, N, NI
      REAL Z(N), D, V, W2, U2, A2, RI, RN, DPLUS, DMINUS, D1, D2, SUMZ,
     +   TWON, ZM1, A2SUM, ZERO, SMALL, HALF, ONE, TWO, TWELVE
C
C     Initialize constants
C
      DATA ZERO/0.0/, SMALL/1.E-37/, HALF/0.5/, ONE/1.0/,
     +     TWO/2.0/, TWELVE/12.0/
C
      RN = N
      TWON = TWO * RN
C
C     Calculating the Kolmorogov statistics DPLUS, DMINUS, D and
C     the Kuiper statistic V.
C
      DPLUS = ZERO
      DMINUS = ZERO
      IFAULT = 4
      DO 10 I = 1, N
        IF (Z(I) .LE. ZERO .OR. Z(I) .GE. ONE) RETURN
        RI = I
        D1 = RI / RN - Z(I)
        IF (D1 .GT. DPLUS) DPLUS = D1
        D2 = Z(I) - (RI - ONE) / RN
        IF (D2 .GT. DMINUS) DMINUS = D2
   10 CONTINUE
      IFAULT = 0
      D = DPLUS
      IF (DMINUS .GT. DPLUS) D = DMINUS
      V = DPLUS + DMINUS
C
C     Calculating the Cramer-von Mises statistic W2 and the Watson
C     statistic U2
C
```

```
      W2 = ONE / (TWELVE * RN)
      SUMZ = ZERO
      DO 20 I = 1, N
        W2 = W2 + (Z(I) - (TWO * I - ONE) / TWON)**2
        SUMZ = SUMZ + Z(I)
   20 CONTINUE
      U2 = W2 - RN * (SUMZ / RN - HALF)**2
C
C     Calculating the Anderson-Darling statistic A2
C
      A2SUM = ZERO
      NI = N
      DO 30 I = 1, N
        IF (Z(I) .LT. SMALL) Z(I) = SMALL
        ZM1 = ONE - Z(NI)
        IF (ZM1 .LT. SMALL) ZM1 = SMALL
        A2SUM = A2SUM + (TWO * I - ONE) * LOG(Z(I) * ZM1)
        NI = NI - 1
   30 CONTINUE
      A2 = -A2SUM / RN - RN
C
      RETURN
      END
C
C
      REAL FUNCTION PVALUE(T, CUT, COEF)
C
C     ALGORITHM AS 248.3  APPL. STATIST. (1989), VOL.38, NO. 3
C
C     Computes the approximate significance level for W**2, U**2 or
C     A**2 when testing for normality or exponentiality
C
      INTEGER I, J
      REAL T, CUT(3), COEF(4, 3), ONE
      DATA ONE /1.0/
C
      I = 4
      DO 10 J = 1, 3
        IF (T .LT. CUT(J)) I = I - 1
   10 CONTINUE
      PVALUE = EXP(COEF(I, 1) + T * (COEF(I, 2) + T * COEF(I, 3)))
      IF (I .LT. 3) PVALUE = ONE - PVALUE
C
      RETURN
      END
```

```
c This file contains algorithm AS 249 for the evaluation of the mean and
c covariance of the truncated multinormal distribution, followed by code
c by the same authors for the numerical estimation of the observed
c information matrix (= maximum likelihood covariance estimates).  As
c the latter is not in the published algorithm, a test program is included
c showing its use.   For further details, contact Phil Leppard, Statistics
c Department, University of Adelaide.   E-mail: PLEPPARD@f.ua.oz
c
c
c
      subroutine mvntrc(n,bounds,u,v,ndim,eps,tmean,tvar,ifault)
c
c     ALGORITHM AS 249  APPL. STATIST. (1989) VOL. 38, NO. 3 (?)
c
c     Auxiliary routines required: MULNOR and MATINV from AS 195.
c     MATINV calls LINRG from the IMSL Stat/Library, but AS 7 can
c     be substituted.
c
      dimension bounds(ndim),u(ndim),v(ndim,ndim),tmean(ndim,1),
     +  tvar(ndim,ndim,1),r(5,5),rinv(5,5),rq(5,5),rqr(5,5),
     +  ind(5),iord(5),sd(5),a(5),aq(5),aqr(5),
     +  phiu(5),phib(5,5),phin1(5),ephin1(5),phin2(5,5),ephin2(5,5),
     +  alpha(3),sum(3),w(10),ww(5,5)
        data zero,one,two/0.0, 1.0, 2.0/
        data  twopi/6.283185308/
        data rtwopi/2.506628275/
        data alphas/1.e-3/
        data ind/5*0/
c
      ifault=0
      if(n.gt.5) go to 100
c
c     Calculate normalised truncation points and correlation
c     matrix R
c
      do 1 i=1,n
        iord(i)=i
        if(v(i,i).le.zero) go to 200
1     sd(i)=sqrt(v(i,i))
      do 2 i=1,n
        a(i)=(bounds(i)-u(i))/sd(i)
        do 2 j=1,n
2     r(i,j)=v(i,j)/(sd(i)*sd(j))
c
c     Order variates by decreasing size of range of integration
c     to increase the speed of MULNOR at highest level
c     (Smallest standardised boundary = largest range, etc)
c
      if(n.gt.2) then
      do 3 i=1,n-1
        do 3 j=i+1,n
          if(a(i).gt.a(j)) then
          aa=a(i)
          a(i)=a(j)
          a(j)=aa
          k=iord(i)
          iord(i)=iord(j)
          iord(j)=k
          do 4 k=1,n
            aa=r(i,k)
            r(i,k)=r(j,k)
```

```
4           r(j,k)=aa
            do 5 k=1,n
              aa=r(k,i)
              r(k,i)=r(k,j)
5           r(k,j)=aa
            end if
3       continue
        end if
c
c
c       Evaluate n dimensional integral alpha = Phi(a:R) and error bound
c
c       alpha(1)=alpha+error   alpha(2)=alpha   alpha(3)=alpha-error
c
        l=0
        do 6 i=1,n
          do 6 j=1,i-1
            l=l+1
6       w(l)=r(i,j)
        call mulnor(w,a,w,eps,n,ind,alpha(2),error,i)
        alpha(1)=alpha(2)+error
        alpha(3)=alpha(2)-error
        if(i.ne.0.or.alpha(2).le.alphas) go to 400
c
c
c       Evaluate univariate and bivariate normal densities
c       Initialise multinormals of dimension n-1 and n-2
c
c
        do 7 i=1,n
          phiu(i)=exp(-a(i)*a(i)/two)/rtwopi
          phin1(i)=one
          ephin1(i)=zero
          do 7 j=i+1,n
            phib(i,j)=(a(i)*a(i)-two*r(i,j)*a(i)*a(j)+a(j)*a(j))/
     +      (two*(1-r(i,j)*r(i,j)))
            phib(i,j)=exp(-phib(i,j))/(twopi*sqrt(one-r(i,j)*r(i,j)))
            phib(j,i)=phib(i,j)
            phin2(i,j)=one
            phin2(j,i)=one
            ephin2(i,j)=zero
7       ephin2(j,i)=zero
c
c
c       Calculation of n-1 dimensional integrals Phi(Aq:Rq)
c
c
        if(n.gt.1) then
        do 8 i=1,n
          do 8 j=1,n
8       rinv(i,j)=r(i,j)
        call matinv(rinv,5,n,ifault)
        if(ifault.gt.0) go to 300
        do 9 iq=1,n
          m1=0
c
c       Determine bounds of integration,vector Aq
c
          do 10 i=1,n
            if(i.ne.iq) then
            m1=m1+1
```

```
            aq(m1)=(a(i)-r(i,iq)*a(iq))/sqrt(one-r(i,iq)*r(i,iq))
            m2=0
            do 11 j=1,n
              if(j.ne.iq) then
              m2=m2+1
              ww(m1,m2)=rinv(i,j)
              end if
11          continue
            end if
10        continue
c
c       Determine n-1 x n-1 correlation matrix Rq
c
          call matinv(ww,5,n-1,ifault)
          if(ifault.gt.0) go to 300
          do 13 i=1,m1
            do 13 j=1,m1
13          rq(i,j)=ww(i,j)/sqrt(ww(i,i)*ww(j,j))
c
c       Evaluate n-1 dimensional integral  Phi(Aq:Rq) and error bound
c
            l=0
            do 14 i=1,m1
              do 14 j=1,i-1
                l=l+1
14          w(l)=rq(i,j)
            call mulnor(w,aq,w,eps,n-1,ind,phin1(iq),ephin1(iq),i)
9       if(i.ne.0) go to 500
        end if
c
c       Calculation of n-2 dimensional integrals Phi(Aqr:Rqr)
c
        if(n.gt.2) then
        do 15 iq=1,n
          do 15 ir=iq+1,n
c
c       Determine n-2 x n-2 correlation matrix Rqr
c
          m1=0
          do 16 i=1,n
            if(i.ne.iq.and.i.ne.ir) then
            m1=m1+1
            m2=0
            do 17 j=1,n
              if(j.ne.iq.and.j.ne.ir) then
              m2=m2+1
              ww(m1,m2)=rinv(i,j)
              end if
17          continue
            end if
16        continue
          call matinv(ww,5,n-2,ifault)
          if(ifault.gt.0) go to 300
          do 19 i=1,m1
            do 19 j=1,m1
19        rqr(i,j)=ww(i,j)/sqrt(ww(i,i)*ww(j,j))
c
c       Determine bounds of integration,vector Aqr
c
          m2=0
          do 20 i=1,n
```

```
            if(i.ne.iq.and.i.ne.ir) then
            m2=m2+1
            bsqr=(r(iq,i)-r(iq,ir)*r(ir,i))/(one-r(iq,ir)*r(iq,ir))
            bsrq=(r(ir,i)-r(iq,ir)*r(iq,i))/(one-r(iq,ir)*r(iq,ir))
            rsrq=(one-r(i,iq)*r(i,iq))*(one-r(ir,iq)*r(ir,iq))
            rsrq=(r(i,ir)-r(i,iq)*r(iq,ir))/sqrt(rsrq)
            aqr(m2)=a(i)-bsqr*a(iq)-bsrq*a(ir)
            aqr(m2)=aqr(m2)/sqrt((one-r(i,iq)*r(i,iq))*(one-rsrq*rsrq))
            end if
   20       continue
c
c       Evaluate n-2 dimensional integral Phi(Aqr:Rqr) and error bound
c
            l=0
            do 21 i=1,m2
              do 21 j=1,i-1
                l=l+1
   21       w(l)=rqr(i,j)
            call mulnor(w,aqr,w,eps,n-2,ind,phin2(iq,ir),ephin2(iq,ir),i)
            if(i.ne.0) go to 600
            phin2(ir,iq)=phin2(iq,ir)
   15     ephin2(ir,iq)=ephin2(iq,ir)
          end if
c
c       Calculation of E(Xi) ,with upper and lower bounds
c       Tallis(1961) , equation (3).
c
          do 22 i=1,n
            do 23 j=1,3
   23       sum(j)=zero
            do 24 j=1,n
              aa=r(i,j)*phiu(j)
              do 25 k=1,3
   25         w(k)=aa*(phin1(j)+(k-2)*ephin1(j))
              if(w(1).gt.w(3)) then
              aa=w(1)
              w(1)=w(3)
              w(3)=aa
              end if
              do 24 k=1,3
   24         sum(k)=sum(k)+w(k)
            k=iord(i)
            do 22 j=1,3
   22     tmean(k,j)=sum(j)/alpha(j)
c
c       Calculation of E(Xi,Xj), with upper and lower bounds
c       Tallis(1961), equation (4).
c
          do 26 i=1,n
            do 26 j=i,n
              do 27 k=1,3
   27         sum(k)=zero
              do 28 k=1,n
                aa=r(k,i)*r(k,j)*a(k)*phiu(k)
                do 29 m1=1,3
   29           w(m1)=aa*(phin1(k)+(m1-2)*ephin1(k))
                if(w(1).gt.w(3)) then
                aa=w(1)
                w(1)=w(3)
                w(3)=aa
                end if
```

```
                do 30 m1=1,3
30              sum(m1)=sum(m1)+w(m1)
                do 28 l=1,n
                  if(l.ne.k) then
                  aa=(r(l,j)-r(k,l)*r(k,j))*r(k,i)*phib(k,l)
                  do 31 m1=1,3
31                w(m1)=aa*(phin2(k,l)+(m1-2)*ephin2(k,l))
                  if(w(1).gt.w(3)) then
                  aa=w(1)
                  w(1)=w(3)
                  w(3)=aa
                  end if
                  do 32 m1=1,3
32                sum(m1)=sum(m1)+w(m1)
                  end if
28            continue
              m1=iord(i)
              m2=iord(j)
              do 26 m3=1,3
                tvar(m1,m2,m3)=r(i,j)+sum(m3)/alpha(m3)
26        tvar(m2,m1,m3)=tvar(m1,m2,m3)
c
c         Calculate truncated covariance matrix, with upper and
c         lower bounds. Original scale restored.
c
          do 33 i=1,n
            do 33 j=i,n
              tvar(i,j,2)=(tvar(i,j,2)-tmean(i,2)*tmean(j,2))*sd(i)*sd(j)
              l=0
              do 34 m1=1,3,2
                do 34 m2=1,3,2
                  do 34 m3=1,3,2
                    l=l+1
34            w(l)=(tvar(i,j,m1)-tmean(i,m2)*tmean(j,m3))*sd(i)*sd(j)
              tvar(i,j,1)=min(w(1),w(2),w(3),w(4),w(5),w(6),w(7),w(8))
              tvar(i,j,3)=max(w(1),w(2),w(3),w(4),w(5),w(6),w(7),w(8))
              do 33 k=1,3
33        tvar(j,i,k)=tvar(i,j,k)
c
c         Original location and scale restored to truncated mean
c
          do 35 i=1,n
            do 35 j=1,3
35        tmean(i,j)=u(i)+sd(i)*tmean(i,j)
c
c         Normal termination of subroutine
c
          return
c
c         Error conditions
c
c
100       ifault=100
c         Dimension n greater than 5
          return
c
200       ifault=200+i
c         i-th variance less than or equal to zero
          return
c
300       ifault=300+i
```

```
c       Error in inversion of i x i matrix
        return
c
400     ifault=400+i
c       Error in Phi(n)...MULNOR error number i
c               or
c       truncated region, ie alpha, "too small" (then ifault=400)
c
        return
c
500     ifault=500+i
c       Error in Phi(n-1)...MULNOR error number i
        return
c
600     ifault=600+i
c       Error in Phi(n-2)...MULNOR error number i
        return
c
        end
c
c  **   End of AS 249
c
c-----------------------------------------------------------------------
c
        program covtest
        implicit double precision (a-h,o-z)
        dimension cov(5,5),a(5),b(5),x(5)
        common r(1000),nn,y(1000,3)
        external fnorm,fmult
        data a,b/10*777.d0/

C       TEST RUN FROM N(0,4)

        nn=100
        x(1)=0.
        x(2)=0.
        do 1 i=1,nn
          r(i)=dnorin((i-.5)/nn)*2
          x(1)=x(1)+r(i)/nn
1       x(2)=x(2)+r(i)*r(i)
        x(2)=(x(2)-nn*x(1)*x(1))/nn
        print 5,x(1),x(2)
5       format(//5x'Mle = ',5f10.3)
        call mlecov(fnorm,x,2,a,b,cov,5,10,1.d-06,1,ifault)
        print 3,ifault
3       format(//5x'Final estimate   ifault='i4)
        do 4 i=1,2
4       print 2,(cov(i,j),j=1,2)
2       format(5x,3f10.5)
        do 20 i=1,2
20      cov(i,i)=sqrt(cov(i,i))
        cov(1,2)=cov(1,2)/(cov(1,1)*cov(2,2))
        cov(2,1)=cov(1,2)
        print 26
26      format(40x'Standard errors and correlations')
        do 21 i=1,2
21      print 22,(cov(i,j),j=1,2)
22      format(40x,3f10.5)

C       TEST RUN FROM MULTINOMIAL P=(.5,.2,.2,.1)
```

```
      nn=4
      r(1)=500
      r(2)=200
      r(3)=200
      r(4)=100
      x(1)=.5
      x(2)=.2
      x(3)=.2
      x(4)=.1
      print 5,(x(i),i=1,3)
      call mlecov(fmult,x,3,a,b,cov,5,20,1.d-10,1,ifault)
      print 3,ifault
      do 8 i=1,3
8     print 2,(cov(i,j),j=1,3)
      do 23 i=1,3
23    cov(i,i)=sqrt(cov(i,i))
      do 25 i=1,3
      do 25 j=1,3
25    if(i.ne.j) cov(i,j)=cov(i,j)/(cov(i,i)*cov(j,j))
      print 26
      do 24 i=1,3
24    print 22,(cov(i,j),j=1,3)
      end
c
c
c
      subroutine matinv(cov,nd,n,ifault)
      implicit double precision (a-h,o-z)
      dimension cov(nd,1)
c
      call dlinrg(n,cov,nd,cov,nd)
      ifault=0
      return
      end
c
c
c
      double precision function fnorm(x)
      implicit double precision (a-h,o-z)
      common r(1000),nn
      dimension x(1)
c
      fnorm=0.
      do 73 i=1,nn
73    fnorm=fnorm-(r(i)-x(1))*(r(i)-x(1))
      fnorm=fnorm/(2*x(2))-nn*log(x(2))/2
      return
      end
c
c
c
      double precision function fmult(x)
      implicit double precision (a-h,o-z)
      dimension x(1),p(5)
      common r(1000),nn
c
      fmult=0.
      p(nn)=1.
      do 1 i=1,nn-1
      p(i)=x(i)
1     p(nn)=p(nn)-p(i)
```

```
        do 2 i=1,nn
2       fmult=fmult+r(i)*log(p(i))
        return
        end
c
c
c  **   End of test program and functions
c--------------------------------------
c
        subroutine mlecov(f,theta,n,a,b,cov,nd,maxh,eps,iprint,ifault)
        implicit double precision (a-h,o-z)
        dimension cov(nd,1),theta(*),a(*),b(*),x(5),h(5),store(5,5)
        external f
        data zero,small,two,hun/0.d0, .001d0, 2.d0, 100.d0/
c
        ifault=0
        if(n.gt.5.or.n.gt.nd) go to 100
c
c       Determine initial values for h(i), i=1,..n
c
        do 1 i=1,n
          h(i)=small
          h(i)=max(h(i),abs(theta(i))/hun)
          if(a(i).eq.b(i)) go to 1
          if(theta(i).lt.a(i).or.theta(i).gt.b(i)) go to 200
          h(i)=min(h(i),abs(a(i)-theta(i)),abs(b(i)-theta(i)))
1       continue
        f00=f(theta)
        ncy=0
2       ncy=ncy+1
        if(ncy.gt.maxh) go to 300
c
c       Evaluation of diagonal elements of -I(theta)
c
        do 3 i=1,n
          cov(i,i)=-two*f00
          do 4 j=1,n
4         x(j)=theta(j)
          x(i)=x(i)-h(i)
          do 3 j=1,2
            cov(i,i)=cov(i,i)+f(x)
3       x(i)=x(i)+two*h(i)
c
c       Evaluation of off-diagonal elements of -I(theta)
c
        do 5 i=1,n
          jj=i+1
          if(jj.gt.n) go to 5
          do 7 j=jj,n
            cov(i,j)=cov(i,i)+cov(j,j)+two*f00
            do 6 k=1,n
6           x(k)=theta(k)
            x(i)=x(i)-h(i)
            x(j)=x(j)-h(j)
            do 8 k=1,2
              cov(i,j)=cov(i,j)-f(x)
              x(i)=x(i)+two*h(i)
8           x(j)=x(j)+two*h(j)
            cov(i,j)=cov(i,j)/(two*h(i)*h(j))
7         cov(j,i)=cov(i,j)
5       cov(i,i)=-cov(i,i)/(h(i)*h(i))
```

```
c
c       In-place inversion of COV
c
        call matinv(cov,nd,n,ifault)
        if(ifault.ne.0) go to 400
c
c       Check for convergence
c
        s=zero
        if(ncy.eq.1) go to 9
        do 10 i=1,n
          do 10 j=i,n
10      s=max(s,abs(cov(i,j)-store(i,j)))
c
c       Display option if requested
c
9       if(iprint.eq.0) go to 11
        print 12,ncy-1,s
12      format(//5x'Covariance estimate at halving ',i3/
     +  5x'Maximum absolute difference from previous estimate = ',e14.7)
        do 13 i=1,n
13      print 14,(cov(i,j),j=1,n)
14      format(5x,5(e14.7,2x))
c
c       Vector h halved and process repeated
c
11      if(s.le.eps.and.ncy.gt.1) return
        do 15 i=1,n
          h(i)=h(i)/two
          do 15 j=i,n
15      store(i,j)=cov(i,j)
        go to 2
c
c       Error conditions
c
100     ifault=100
c       Dimension or problem size misspecified
        return
200     ifault=200+i
c       Maximum likelihood estimate for i-th parameter outside bounds
        return
300     ifault=300
c       Convergence not achieved in maxh iterations
        return
400     ifault=400+ncy
c       Matrix inversion difficulties at iteration ncy
        return
        end
```

```
      SUBROUTINE DMAT(KSET, NP, ND, NM, NG, X, XSIGMA, MDF, CHI, DETMUL,
     *               W2, IFAULT)
C
C        ALGORITHM AS 250 APPL. STATIST. (1989), VOL. 38, NO. 3
C
C        Test of equality of dispersion matrices using
C        Sen & Puri's non-parametric and Anderson's parametric methods.
C
C     Auxiliary routine required: CHOL from AS 6
C
      INTEGER IA, IB, IC, ID, IE
      PARAMETER (IA=9, IB=10, IC=IA * (IA + 1)/2, ID=1000,
     *          IE=IC * (IC + 1)/2)
C
C        Arguments
      INTEGER NG( * ), IFAULT, KSET, MDF, ND, NM, NP
      REAL X(IA, * ), XSIGMA(IC, * ), CHI, DETMUL, W2
C
C        Local variables
      INTEGER I, I1, II, IM, J, J1, JJ, K, L, M, MP, MR, MS, N, NPT, NT,
     *        NULLTY
      REAL XMEAN(IA, IB), XMALL(IA), DET(IB), XSALL(IC), Y(ID), VRS(IE),
     *     W(IE), DE, DETALL, F48, F6, HALF, ONE, R1, R2, RHO, S12,
     *     TEMP, TWO
      DATA HALF, ONE, TWO, F6, S12, F48 / 0.5, 1.0, 2.0, 6.0,
     *     3.4641016151377, 48.0 /
C
      NT = 0
      DO 10 K = 1, KSET
         NT = NT + NG(K)
   10 CONTINUE
      MP = NP * (NP + 1) / 2
      MDF = 0
      CHI = 0.0
      W2 = 0.0
      DETMUL = 0.0
C
C        Check input data for failure.
C
      IFAULT = 0
      IF (KSET .LT. 1 .OR. NP .LT. 1 .OR.
     *    (ND .NE. 1 .AND. ND .NE. 2) .OR.
     *    (NM .NE. 1 .AND. NM .NE. 2)) IFAULT = 1
      IF (ND .EQ. 2 .AND. NM .NE. 2) IFAULT = 2
      DO 20 K = 1, KSET
         IF (NG(K) .LT. 1) IFAULT = 3
   20 CONTINUE
C
C        Partial check on dimensions.
C
      IF (IA .LT. NP .OR. IB .LT. KSET .OR. ID .LT. NT) IFAULT = 4
      IF (IFAULT .NE. 0) RETURN
C
C        Check the `type' of data input
C
      IF (ND .EQ. 2) GO TO 290
C
C        Manipulate raw data according to flag given by NM and
C        calculate MEAN and SIGMA.  Note that `raw' data is `ordered'
C        if Sen & Puri's method(NM=1) is used.
C
```

```
C        Rank order the raw data if NM=1 and load it in X(L,N).
C        Ordered data for each variable are temporarily stored in Y(N).
C        Map the ranked data into its general score such that variate
C        means = 0 and variate std = 1 across all samples.
C
      IF (NM .EQ. 1) THEN
         DO 70 L = 1, NP
            DO 30 N = 1, NT
               Y(N) = ONE
   30       CONTINUE
            DO 50 I = 1, NT
               DO 40 N = 1, I - 1
C
                  IF (X(L, I) .LT. X(L, N)) THEN
                     Y(N) = Y(N) + ONE
                  ELSE IF (X(L, I) .EQ. X(L, N)) THEN
                     Y(N) = Y(N) + HALF
                     Y(I) = Y(I) + HALF
                  ELSE
                     Y(I) = Y(I) + ONE
                  END IF
C
   40          CONTINUE
   50       CONTINUE
C
            DO 60 N = 1, NT
               X(L, N) = S12 * (AINT(Y(N)) / (NT + 1) - HALF)
   60       CONTINUE
   70    CONTINUE
      END IF
C
C        The `raw' or `ordered' data are now in X(L,N).  Compute mean
C        of each variate for each sample and store in XMEAN(L,K).
C
C        Compute mean of each variate over all samples, store in
C        XMALL(L).  Note that when NM=1, this step is known and is
C        equal to zero since the ranks are scaled.  When NM=2, XMALL(L)
C        is not needed.
C
      DO 100 L = 1, NP
         NPT = 0
         XMALL(L) = 0.0
         DO 90 K = 1, KSET
            XMEAN(L, K) = 0.0
            DO 80 N = 1, NG(K)
               NPT = NPT + 1
               XMEAN(L, K) = XMEAN(L, K) + X(L, NPT)
   80       CONTINUE
            XMEAN(L, K) = XMEAN(L, K) / NG(K)
   90    CONTINUE
  100 CONTINUE
C
C        Compute sigma matrix for each sample XSIGMA(M,K) where M takes
C        values 1 through NP*(NP+1)/2.  Compute sigma of all samples
C        XSALL(M).  Calculation of these matrices is controlled by NM.
C
      M = 0
      DO 140 I = 1, NP
         DO 130 J = 1, NP
            M = M + 1
            NPT = 0
```

```
            XSALL(M) = 0.0
            DO 120 K = 1, KSET
              XSIGMA(M, K) = 0.0
              DO 110 N = 1, NG(K)
                 NPT = NPT + 1
                 XSIGMA(M, K) = XSIGMA(M, K) +
     *                          (X(I, NPT) - XMEAN(I, K)) *
     *                          (X(J, NPT) - XMEAN(J, K))
  110            CONTINUE
C
                 IF (NM .EQ. 1) THEN
                    XSALL(M) = XSALL(M) + XSIGMA(M, K) +
     *                         XMEAN(I, K) * XMEAN(J, K) * NG(K)
                    XSIGMA(M, K) = XSIGMA(M, K) / (NG(K) - 1)
                 ELSE
                    XSALL(M) = XSALL(M) + XSIGMA(M, K)
                 END IF
C
  120         CONTINUE
              IF (NM .EQ. 1) XSALL(M) = XSALL(M) / (NT - 1)
  130       CONTINUE
  140    CONTINUE
C
C          Check the method chosen: Sen and Puri (NM=1) or Anderson (NM=2)
C
        IF (NM .EQ. 2) GO TO 330
C
C          Sen and Puri's method
C
C          Compute the lower triangle of 4th moment matrix VRS of order
C          (MP,MP), where MP=MP*(NP+1)/2, in the one dimensional array
C          VRS.
C
       L = 0
       MR = 0
       DO 200 I = 1, NP
          DO 190 J = I, NP
             MR = MR + 1
             MS = 0
             DO 180 I1 = 1, I
                DO 170 J1 = I1, NP
                   MS = MS + 1
                   IF (MR .GE. MS) THEN
                      L = L + 1
                      NPT = 0
                      VRS(L) = 0.0
                      DO 160 K = 1, KSET
                         DO 150 N = 1, NG(K)
                            NPT = NPT + 1
                            VRS(L) = VRS(L) + X(I, NPT) * X(J, NPT) *
     *                                 X(I1, NPT) * X(J1, NPT)
  150                    CONTINUE
  160                 CONTINUE
                      VRS(L) = VRS(L) / NT - XSALL(MR) * XSALL(MS)
                   END IF
  170           CONTINUE
  180        CONTINUE
  190     CONTINUE
  200  CONTINUE
C
C         Compute XSIGMA (M,K)-XSALL(M) for all M,K.  Sotre the results
```

```
C        in XSIGMA(M,K).
C
      DO 220 K = 1, KSET
         DO 210 M = 1, MP
            XSIGMA(M, K) = XSIGMA(M, K) - XSALL(M)
  210    CONTINUE
  220 CONTINUE
C
C        Compure the quadratic form:
C        CHI = (XSIGMA transpose * VRS inverse * XSIGMA)
C        by first getting the square root W of VRS (see sub. CHOL), then
C        finding W inverse * XSIGMA, and finally CHI.
C
      CALL CHOL(VRS, MP, MP * (MP + 1) / 2, W, NULLTY, IFAULT)
C
C        Invert root of VRS (=W).  Note that W is an upper diagonal
C        matrix and is inverted `in place'.
C
      IF (NULLTY .GT. 0 .OR. IFAULT .GT. 0) THEN
         IFAULT = 5
         RETURN
      END IF
      DO 250 J = MP, 1, -1
         JJ = J * (J + 1) / 2
         W(JJ) = 1.0 / W(JJ)
         DO 240 I = J - 1, 1, -1
            II = I * (I + 1) / 2
            IM = II
            TEMP = 0.0
            DO 230 M = I + 1, J
               IM = IM + M - 1
               TEMP = TEMP + W(IM) * W(JJ + M - J)
  230       CONTINUE
            W(IM) = -TEMP / W(II)
  240    CONTINUE
  250 CONTINUE
C
C        Compute CHI
C
      CHI = 0.0
      DO 280 K = 1, KSET
         DO 270 I = 1, MP
            N = I * (I - 1) / 2
            TEMP = 0.0
            DO 260 J = 1, I
               TEMP = TEMP + W(N + J) * XSIGMA(J, K)
  260       CONTINUE
            CHI = CHI + TEMP ** 2
  270    CONTINUE
  280 CONTINUE
      MDF = (KSET - 1) * MP
C
      RETURN
C
C        Anderson's method, XSIGMA input
C
C        Calculate the total matrix for all samples and store it in
C        XSALL(M).
C
  290 CONTINUE
      DO 300 M = 1, MP
```

```
            XSALL(M) = 0.0
  300    CONTINUE
         DO 320 K = 1, KSET
            DO 310 M = 1, MP
               XSALL(M) = XSALL(M) + XSIGMA(M, K)
  310       CONTINUE
  320    CONTINUE
C
C        Anderson's method, raw data or XSIGMA inpur.
C        From the diagonal XSIGMA matrix and XSALL generate the lower
C        triangle of respective covariance matrices.  Store these lower
C        triangles in the one dimensional array VRS.  Get the
C        log(determinant) for each matrix by calling subroutine `CHOL'.
C        store results in DET(K) and DETALL.
C
  330    DO 370 K = 1, KSET
            M = 0
            DO 350 I = 1, NP
               N = I * (I - 1) /2 + 1
               DO 340 J = I, NP
                  M = M + 1
                  N = N + J - 1
                  VRS(N) = XSIGMA(M, K) / (NG(K) - 1)
  340          CONTINUE
  350       CONTINUE
            CALL CHOL(VRS, NP, NP * (NP + 1) / 2, W, NULLTY, IFAULT)
            IF (NULLTY .GT. 0 .OR. IFAULT .GT. 0) THEN
               IFAULT = 5
               RETURN
            END IF
            DE = 1.0
            N = 0
            DO 360 I = 1, NP
               N = N + I
               DE = DE * W(N)
  360       CONTINUE
            DET(K) = TWO * LOG(DE)
  370    CONTINUE
C
         M = 0
         DO 390 I = 1, NP
            N = I * (I - 1) / 2 + 1
            DO 380 J = I, NP
               M = M + 1
               N = N + J - 1
               VRS(N) = XSALL(M) / (NT - KSET)
  380       CONTINUE
  390    CONTINUE
         CALL CHOL(VRS, NP, NP * (NP + 1) / 2, W, NULLTY, IFAULT)
         IF (NULLTY .GT. 0 .OR. IFAULT .GT. 0) THEN
            IFAULT = 5
            RETURN
         END IF
         DE = 1.0
         N = 0
         DO 400 I = 1, NP
            N = N + I
            DE = DE * W(N)
  400    CONTINUE
         DETALL = TWO * LOG(DE)
C
```

```
C        Compute log(V), store in DETMUL.  Compute CHI.
C
      DETMUL = 0.0
      DO 410 K = 1, KSET
         DETMUL = DETMUL + (DET(K) - DETALL) * (NG(K) - 1) / TWO
  410 CONTINUE
C
C        Calculate RHO and W2
C
      M = NT - KSET
      R1 = 0.0
      R2 = 0.0
      DO 420 K = 1, KSET
         R1 = R1 + ONE / (NG(K) - 1)
         R2 = R2 + ONE / (NG(K) - 1) ** 2
  420 CONTINUE
      RHO = ONE - (R1 - ONE / M) * (2 * (NP ** 2) + 3 * NP - 1) /
     *      (F6 * (NP + 1) * (KSET - 1))
      W2 = NP * (NP + 1) * ((NP - 1) * (NP + 2) * (R2 - ONE / M ** 2) -
     *     F6 * (KSET - 1) * (1 - RHO) * * 2) / (F48 * RHO ** 2)
C
      MDF = (KSET - 1) * MP
      CHI = -TWO * RHO * DETMUL
      RETURN
      END
```

This file contains:
1. A Readme file provided by Charles Dunnett, the author of AS 251.
2. The published algorithm AS 251 together with the two other AS algorithms which it calls (AS 66 and AS 241).
3. A driver program (MVTIN) for either multivariate normal or t.
4. MVSTUD for calculating multivariate t probabilities.

****

Date: Mon, 10 Apr 1995 16:49:10 +0059 (EDT)
From: "Charles W. Dunnett" <dunnett@mcmail.cis.mcmaster.ca>
Subject: Readme for AS251 (extended version incl. multivariate t)

MVTIN is a driver program for computing multivariate normal or t probability integrals over arbitrary rectangular regions. The correlation structure is assumed to be of product form, rho_ij = b_i x b_j, where -1 < b_i < +1.

It requires the following:-

1. MVNPRD (published as algorithm AS 251 in Applied Statistics (1989), 38: 564-579; see also the correction note in Applied Statistics (1993), 42: 709),

2. ALNORM and PPND7 (published as algorithms AS 66 and AS 241, respectively, in Applied Statistics, and

3. MVSTUD (which Studentizes MVNPRD).

Special cases of correlations which are of product form include the following:-

a) Equal correlations, rho >= 0. Then b_i = sqrt(rho), all i.

b) Multiple comparisons between treatments and a specified treatment. Then b_i = 1 / sqrt(1 + n_0/n_i), where n_0 and n_i are the sample sizes for the specified treatment and i-th treatment, respectively.

MVTIN computes

PROB = P{ B(I) < T_I < A(I), for I = 1,...,N }

where (T_1,...,T_N) is a multivariate normal or t random variable with product correlation structure. The B(I) may be -infinity or finite and the A(I) may be finite or +infinity.

The program can be used on any mainframe computer (such as a VAX) which has a Fortran compiler. A PC can also be used, but the execution times will be much longer, especially for multivariate t integrals (unless your PC uses a 486 or faster chip). Typical computing times for a multivariate t p-value range from about three minutes on an IBM/XT with math co-processor to 2.4 seconds on a 486DX2/66 PC.

The first time you use the program, it is recommended that you specify NDF = 0 (which denotes the multivariate normal case), since computing times are shorter than for multivariate t.

The accuracy is determined by the value of the parameter EPS, which has been pre-set to EPS = 0.0001. For increased accuracy, you may change the value of EPS in the DATA statement of MVTIN to

EPS = 0.00001 (Smaller values than this are not recommended).

EXAMPLES:-

(1) Suppose the problem is to determine g* such that

$$P\{\min(Z1,...,Zk) > g^*\}$$

$$= P\{Z1 > g^*, \ldots , Zk > g^*\} = \text{alpha}$$

where Z1,...,Zk are N(0,1) with equal correlations, rho_ij = 0.5. The first prompt you receive is for N and NDF. You enter the value of k for N, and 0 for NDF (indicating d.f.= infinity). (At the end of each problem, the program returns to this prompt for the next problem. If there is no further problem, enter 0 0 to exit from the program.)

The next prompt is to specify the correlation structure. You are given three choices: 1) equal correlations, 2) correlations defined from sample sizes as in b) above, and 3) unequal correlations of the form rho_ij = b_i x b_j. For this example, enter 1, followed by 0.50 when you receive a prompt for rho.

Next you are asked whether you want equal or unequal limits. Since all your limits are to be g* to +infinity, you specify equal limits. You are also asked whether you want the central or non-central case: if the latter, you are asked to enter the value(s) of the non-centrality parameters.

Then you are prompted to enter the values of INF and the limit(s). You enter 0 (to indicate upper 1-sided) followed by a trial value for g*.

After computing PROB, you have the option of entering new limits to obtain a new value of PROB, or returning to the beginning for a new problem.

For example, for k = 5 and rho = 0.50, I found that g* = 0.564 achieved the value alpha = 0.05.

(2) For the next example, consider a mc/control problem with sample sizes 11 for the control and 10, 10, 12, 10 and 9 for the treatment groups. Here, k = 5 and d.f. = 56. Suppose the largest of the t-statistics comparing treatments with the control has the value 2.50. You want to determine its two-sided p-value, which is given by

$$p = 1 - P\{ |T_1| < 2.5, \ldots , |T_5| < 2.5 \}.$$

At the prompt for N and NDF, enter: 5 56.

At the prompt for correlation structure: enter 2 to denote it is to be determined from the sample sizes, followed by the sample sizes at the next prompt.

At the prompt for equal or unequal limits, enter: 1.

Enter 0 for the central case.

Then for the value of INF and the limits, enter: 2 -2.50 2.50

You should obtain the value PROB = .9375. This is the probability for -2.50 < T_i < 2.50 (all i); then p-value = 1-.9375 = .0625.

Let me know if you encounter problems. Also, please let me know if it does not meet your needs in some way, or if you think of any improvements.

Good luck!

With best wishes, Charles Dunnett.

* *
Professor Charles W. Dunnett
Dept. of Mathematics and Statistics
McMaster University
Hamilton, Ontario L8S 4K1
Canada.
E-mail: dunnett@mcmaster.ca
TEL: (905) 525-9140 (Extension 27104)
FAX: (905) 522-0935
* *

```
      SUBROUTINE MVNPRD(A, B, BPD, EPS, N, INF, IERC, HINC, PROB, BOUND,
     *  IFAULT)
C
C        ALGORITHM AS 251.1  APPL.STATIST. (1989), VOL.38, NO.3
C
C        FOR A MULTIVARIATE NORMAL VECTOR WITH CORRELATION STRUCTURE
C        DEFINED BY RHO(I,J) = BPD(I) * BPD(J), COMPUTES THE PROBABILITY
C        THAT THE VECTOR FALLS IN A RECTANGLE IN N-SPACE WITH ERROR
C        LESS THAN EPS.
C
      INTEGER NN
      PARAMETER (NN = 50)
      REAL A(*), B(*), BPD(*), ESTT(22), FV(5), FD(5), F1T(22),
     *  F2T(22), F3T(22), G1T(22), G3T(22), PSUM(22), H(NN), HL(NN),
     *  BB(NN)
      INTEGER INF(*), INFT(NN), LDIR(22)
      REAL ZERO, HALF, ONE, TWO, FOUR, SIX, PT1, PT24, ONEP5,
     *  X2880, SMALL, DXMIN, SQRT2, PROB, ERRL, BI, START,
     *  Z, HINC, ADDN, EPS2, EPS1, EPS, ZU, Z2, Z3, Z4, Z5, ZZ,
     *  ERFAC, EL, EL1, BOUND, PART0, PART2, PART3, FUNC0, FUNC2,
     *  FUNCN, WT, CONTRB, DLG, DX, DA, ESTL, ESTR, SUM, EXCESS, ERROR,
     *  PROB1, SAFE
      INTEGER N, IERC, IFAULT, I, NTM, NMAX, LVL, NR, NDIM
      REAL ALNORM, PPND7
      EXTERNAL ALNORM, PPND7
      DATA ZERO, HALF, ONE, TWO, FOUR, SIX /0.0, 0.5, 1.0, 2.0,
     *  4.0, 6.0/
      DATA PT1, PT24, ONEP5, X2880 /0.1, 0.24, 1.5, 2880.0/
      DATA SMALL, DXMIN, SQRT2 /1.0E-10, 0.0000001, 1.41421356237310/
C
C        CHECK FOR INPUT VALUES OUT OF RANGE.
C
      PROB = ZERO
      BOUND = ZERO
      IFAULT = 1
      IF (N .LT. 1 .OR. N .GT. NN) RETURN
```

```
      DO 10 I = 1, N
         BI = ABS(BPD(I))
         IFAULT = 2
         IF (BI .GE. ONE) RETURN
         IFAULT = 3
         IF (INF(I) .LT. 0 .OR. INF(I) .GT. 2) RETURN
         IFAULT = 4
         IF (INF(I) .EQ. 2 .AND. A(I) .LE. B(I)) RETURN
   10 CONTINUE
      IFAULT = 0
      PROB = ONE
C
C        CHECK WHETHER ANY BPD(I) = 0.
C
      NDIM = 0
      DO 20 I = 1, N
         IF (BPD(I) .NE. ZERO) THEN
            NDIM = NDIM + 1
            H(NDIM) = A(I)
            HL(NDIM) = B(I)
            BB(NDIM) = BPD(I)
            INFT(NDIM) = INF(I)
         ELSE
C
C           IF ANY BPD(I) = 0, THE CONTRIBUTION TO PROB FOR THAT
C           VARIABLE IS COMPUTED FROM A UNIVARIATE NORMAL.
C
            IF (INF(I) .LT. 1) THEN
               PROB = PROB * (ONE - ALNORM(B(I), .FALSE.))
            ELSE IF (INF(I) .EQ. 1) THEN
               PROB = PROB * ALNORM(A(I), .FALSE.)
            ELSE
               PROB = PROB * (ALNORM(A(I), .FALSE.) -
     *                 ALNORM(B(I), .FALSE.))
            END IF
            IF (PROB .LE. SMALL) PROB = ZERO
         END IF
   20 CONTINUE
      IF (NDIM .EQ. 0 .OR. PROB .EQ. ZERO) RETURN
C
C        IF NOT ALL BPD(I) = 0, PROB IS COMPUTED BY SIMPSON'S RULE.
C        BUT FIRST, INITIALIZE THE VARIABLES.
C
      Z = ZERO
      IF (HINC .LE. ZERO) HINC = PT24
      ADDN = -ONE
      DO 30 I = 1, NDIM
         IF (INFT(I) .EQ. 2 .OR.
     *   (INFT(I) .NE. INFT(1) .AND. BB(I) * BB(1) .GT. ZERO) .OR.
     *   (INFT(I) .EQ. INFT(1) .AND. BB(I) * BB(1) .LT. ZERO))
     *       ADDN = ZERO
   30 CONTINUE
C
C        THE VALUE OF ADDN IS TO BE ADDED TO THE PRODUCT EXPRESSIONS IN
C        THE INTEGRAND TO INSURE THAT THE LIMITING VALUE IS ZERO.
C
      PROB1 = ZERO
      NTM = 0
      NMAX = 400
      IF (IERC .EQ. 0) NMAX = NMAX * 2
      CALL PFUNC (Z, H, HL, BB, NDIM, INFT, ADDN, SAFE, FUNC0, NTM,
```

```
     *  IERC, PART0)
        EPS2 = EPS * PT1 * HALF
C
C          SET UPPER BOUND ON Z AND APPORTION EPS.
C
        ZU = -PPND7(EPS2, IFAULT) / SQRT2
        IF (IFAULT .NE. 0) THEN
           IFAULT = 6
           RETURN
        END IF
        NR = IFIX(ZU / HINC) + 1
        ERFAC = ONE
        IF (IERC .NE. 0) ERFAC = X2880 / HINC ** 5
        EL = (EPS - EPS2) / FLOAT(NR) * ERFAC
        EL1 = EL
C
C          START COMPUTATIONS FOR THE INTERVAL (Z, Z + HINC).
C
   40   ERROR = ZERO
        LVL = 0
        FV(1) = PART0
        FD(1) = SAFE
        START = Z
        DA = HINC
        Z3 = START + HALF * DA
        CALL PFUNC(Z3, H, HL, BB, NDIM, INFT, ADDN, FD(3), FUNCN, NTM,
     *  IERC, FV(3))
        Z5 = START + DA
        CALL PFUNC(Z5, H, HL, BB, NDIM, INFT, ADDN, FD(5), FUNC2, NTM,
     *  IERC, FV(5))
        PART2 = FV(5)
        SAFE = FD(5)
        WT = DA / SIX
        CONTRB = WT * (FV(1) + FOUR * FV(3) + FV(5))
        DLG = ZERO
        IF (IERC .NE. 0) THEN
           CALL WMAX(FD(1), FD(3), FD(5), DLG)
           IF (DLG .LE. EL) GO TO 90
           DX = DA
           GO TO 60
        END IF
        LVL = 1
        LDIR(LVL) = 2
        PSUM(LVL) = ZERO
C
C          BISECT INTERVAL.  IF IERC = 1, COMPUTE ESTIMATE ON LEFT
C          HALF; IF IERC = 0, ON BOTH HALVES.
C
   50   DX = HALF * DA
        WT = DX / SIX
        Z2 = START + HALF * DX
        CALL PFUNC(Z2, H, HL, BB, NDIM, INFT, ADDN, FD(2), FUNCN, NTM,
     *  IERC,FV(2))
        ESTL = WT * (FV(1) + FOUR * FV(2) + FV(3))
        IF (IERC .EQ. 0) THEN
           Z4 = START + ONEP5 * DX
           CALL PFUNC(Z4, H, HL, BB, NDIM, INFT, ADDN, FD(4), FUNCN,
     *     NTM, IERC, FV(4))
           ESTR = WT * (FV(3) + FOUR * FV(4) + FV(5))
           SUM = ESTL + ESTR
           DLG = ABS(CONTRB - SUM)
```

```
            EPS1 = EL / TWO ** (LVL - 1)
            ERRL = DLG
         ELSE
            FV(3) = FV(2)
            FD(3) = FD(2)
            CALL WMAX(FD(1), FD(3), FD(5), DLG)
            ERRL = DLG / TWO ** (5 * LVL)
            SUM = ESTL
            EPS1 = EL * (TWO ** LVL) ** 4
         END IF
C
C        STOP SUBDIVIDING INTERVAL WHEN ACCURACY IS SUFFICIENT,
C        OR IF INTERVAL TOO NARROW OR SUBDIVIDED TOO OFTEN.
C
         IF (DLG .LE. EPS1 .OR. DLG .LT. SMALL) GO TO 70
         IF (IFAULT .EQ. 0 .AND. NTM .GE. NMAX) IFAULT = 5
         IF (ABS(DX) .LE. DXMIN .OR. LVL .GT. 21) IFAULT = 7
         IF (IFAULT .NE. 0) GO TO 70
C
C        RAISE LEVEL.  STORE INFORMATION FOR RIGHT HALF AND APPLY
C        SIMPSON'S RULE TO LEFT HALF.
C
      60 LVL = LVL + 1
         LDIR(LVL) = 1
         F1T(LVL) = FV(3)
         F3T(LVL) = FV(5)
         DA = DX
         FV(5) = FV(3)
         IF (IERC .EQ. 0) THEN
            F2T(LVL) = FV(4)
            ESTT(LVL) = ESTR
            CONTRB = ESTL
            FV(3) = FV(2)
         ELSE
            G1T(LVL) = FD(3)
            G3T(LVL) = FD(5)
            FD(5) = FD(3)
         END IF
         GO TO 50
C
C        ACCEPT APPROXIMATE VALUE FOR INTERVAL.
C        RESTORE SAVED INFORMATION TO PROCESS
C        RIGHT HALF INTERVAL.
C
      70 ERROR = ERROR + ERRL
      80 IF (LDIR(LVL) .EQ. 1) THEN
            PSUM(LVL) = SUM
            LDIR(LVL) = 2
            IF (IERC .EQ. 0) DX = DX * TWO
            START = START + DX
            DA = HINC / TWO ** (LVL - 1)
            FV(1) = F1T(LVL)
            IF (IERC .EQ. 0) THEN
               FV(3) = F2T(LVL)
               CONTRB = ESTT(LVL)
            ELSE
               FV(3) = F3T(LVL)
               FD(1) = G1T(LVL)
               FD(5) = G3T(LVL)
            END IF
            FV(5) = F3T(LVL)
```

```
            GO TO 50
          END IF
          SUM = SUM + PSUM(LVL)
          LVL = LVL - 1
          IF (LVL .GT. 0) GO TO 80
          CONTRB = SUM
          LVL = 1
          DLG = ERROR
       90 PROB1 = PROB1 + CONTRB
          BOUND = BOUND + DLG
          EXCESS = EL - DLG
          EL = EL1
          IF (EXCESS .GT. ZERO) EL = EL1 + EXCESS
          IF ((FUNC0 .GT. ZERO .AND. FUNC2 .LE. FUNC0) .OR.
         *    (FUNC0 .LT. ZERO .AND. FUNC2 .GE. FUNC0)) THEN
            ZZ = -SQRT2 * Z5
            PART3 = ABS(FUNC2) * ALNORM(ZZ, .FALSE.) + BOUND / ERFAC
            IF (PART3 .LE. EPS .OR. NTM .GE. NMAX .OR. Z5 .GE. ZU) GOTO 100
          END IF
          Z = Z5
          PART0 = PART2
          FUNC0 = FUNC2
          IF (Z .LT. ZU .AND. NTM .LT. NMAX) GO TO 40
      100 PROB = (PROB1 - ADDN * HALF) * PROB
          BOUND = PART3
          IF (NTM .GE. NMAX .AND. IFAULT .EQ. 0) IFAULT = 5
          IF (BOUND .GT. EPS .AND. IFAULT .EQ. 0) IFAULT = 8
          RETURN
          END
          SUBROUTINE PFUNC(Z, A, B, BPD, N, INF, ADDN, DERIV, FUNCN, NTM,
         *  IERC, RESULT)
      C
      C        ALGORITHM AS 251.2  APPL.STATIST. (1989), VOL.38, NO.3
      C
      C
      C        COMPUTE FUNCTION IN INTEGRAND AND ITS 4TH DERIVATIVE.
      C
          INTEGER NN
          PARAMETER (NN = 50)
          REAL A(*), B(*), BPD(*), FOU(NN), FOU1(4, NN), TMP(4), GOU(NN),
         *  GOU1(4, NN), FF(4), GF(4), TERM(4), GERM(4)
          INTEGER INF(*)
          REAL ZERO, ONE, TWO, THREE, FOUR, SIX, EIGHT, TWELVE, SIXTN,
         *  SMALL, Z, U, U1, U2, BI, HI, HLI, BP, ADDN, DERIV, FUNCN,
         *  RESULT, RSLT1, RSLT2, DEN, SQRT2, SQRTPI, PHI, PHI1, PHI2,
         *  PHI3, PHI4, FRM, GRM
          INTEGER N, NTM, IERC, INFI, I, J, K, M, L, IK
          REAL ALNORM
          EXTERNAL ALNORM
          DATA ZERO, ONE, TWO, THREE, FOUR, SIX, EIGHT, TWELVE, SIXTN,
         *  SMALL /0.0, 1.0, 2.0, 3.0, 4.0, 6.0, 8.0, 12.0, 16.0, 0.1E-12/
          DATA SQRT2, SQRTPI /1.41421356237310, 1.77245385090552/
          DERIV = ZERO
          NTM = NTM + 1
          RSLT1 = ONE
          RSLT2 = ONE
          BI = ONE
          HI = A(1) + ONE
          HLI = B(1) + ONE
          INFI = -1
          DO 60 I = 1, N
```

```
            IF (BPD(I) .EQ. BI .AND. A(I) .EQ. HI .AND. B(I) .EQ. HLI .AND.
     *      INF(I) .EQ. INFI) THEN
               FOU(I) = FOU(I - 1)
               GOU(I) = GOU(I - 1)
               DO 10 IK = 1, 4
                  FOU1(IK, I) = FOU1(IK, I - 1)
                  GOU1(IK, I) = GOU1(IK, I - 1)
10             CONTINUE
         ELSE
            BI = BPD(I)
            HI = A(I)
            HLI = B(I)
            INFI = INF(I)
            IF (BI .EQ. ZERO) THEN
               IF (INFI .LT. 1) THEN
                  FOU(I) = ONE - ALNORM(HLI, .FALSE.)
               ELSE IF (INFI .EQ. 1) THEN
                  FOU(I) = ALNORM(HI, .FALSE.)
               ELSE
                  FOU(I) = ALNORM(HI, .FALSE.) - ALNORM(HLI, .FALSE.)
               END IF
               GOU(I) = FOU(I)
               DO 20 IK = 1, 4
                  FOU1(IK, I) = ZERO
                  GOU1(IK, I) = ZERO
20             CONTINUE
            ELSE
               DEN = SQRT(ONE - BI * BI)
               BP = BI * SQRT2 / DEN
               IF (INFI .LT. 1) THEN
                  U = -HLI / DEN + Z * BP
                  FOU(I) = ALNORM(U, .FALSE.)
                  CALL ASSIGN (U, BP, FOU1(1, I))
                  BP = -BP
                  U = -HLI / DEN + Z * BP
                  GOU(I) = ALNORM(U, .FALSE.)
                  CALL ASSIGN (U, BP, GOU1(1, I))
               ELSE IF (INFI .EQ. 1) THEN
                  U = HI / DEN + Z * BP
                  GOU(I) = ALNORM(U, .FALSE.)
                  CALL ASSIGN (U, BP, GOU1(1, I))
                  BP = -BP
                  U = HI / DEN + Z * BP
                  FOU(I) = ALNORM(U, .FALSE.)
                  CALL ASSIGN (U, BP, FOU1(1, I))
               ELSE
                  U2 = -HLI / DEN + Z * BP
                  CALL ASSIGN (U2, BP, FOU1(1, I))
                  BP = -BP
                  U1 = HI / DEN + Z * BP
                  CALL ASSIGN (U1, BP, TMP(1))
                  FOU(I) = ALNORM(U1, .FALSE.) + ALNORM(U2, .FALSE.) - ONE
                  DO 30 IK = 1, 4
                     FOU1(IK, I) = FOU1(IK, I) + TMP(IK)
30                CONTINUE
                  IF (-HLI .EQ. HI) THEN
                     GOU(I) = FOU(I)
                     DO 40 IK = 1, 4
                        GOU1(IK, I) = FOU1(IK, I)
40                   CONTINUE
                  ELSE
```

```
                  U2 = -HLI / DEN + Z * BP
                  CALL ASSIGN (U2, BP, GOU1(1, I))
                  BP = -BP
                  U1 = HI / DEN + Z * BP
                  GOU(I) = ALNORM(U1, .FALSE.) + ALNORM(U2, .FALSE.)-ONE
                  CALL ASSIGN (U1, BP, TMP(1))
                  DO 50 IK = 1, 4
                     GOU1(IK, I) = GOU1(IK, I) + TMP(IK)
   50             CONTINUE
               END IF
            END IF
         END IF
      END IF
      RSLT1 = RSLT1 * FOU(I)
      RSLT2 = RSLT2 * GOU(I)
      IF (RSLT1 .LE. SMALL) RSLT1 = ZERO
      IF (RSLT2 .LE. SMALL) RSLT2 = ZERO
   60 CONTINUE
      FUNCN = RSLT1 + RSLT2 + ADDN
      RESULT = FUNCN * EXP(-Z * Z) / SQRTPI
C
C        IF 4TH DERIVATIVE IS NOT WANTED, STOP HERE.
C        OTHERWISE, PROCEED TO COMPUTE 4TH DERIVATIVE.
C
      IF (IERC .EQ. 0) RETURN
      DO 70 IK = 1, 4
         FF(IK) = ZERO
         GF(IK) = ZERO
   70 CONTINUE
      DO 100 I = 1, N
         FRM = ONE
         GRM = ONE
         DO 80 J = 1, N
            IF (J .EQ. 1) GO TO 80
            FRM = FRM * FOU(J)
            GRM = GRM * GOU(J)
            IF (FRM .LE. SMALL) FRM = ZERO
            IF (GRM .LE. SMALL) GRM = ZERO
   80    CONTINUE
         DO 90 IK = 1, 4
            FF(IK) = FF(IK) + FRM * FOU1(IK, I)
            GF(IK) = GF(IK) + GRM * GOU1(IK, I)
   90    CONTINUE
  100 CONTINUE
      IF (N .LE. 2) GO TO 230
      DO 130 I = 1, N
         DO 120 J = I + 1, N
            TERM(2) = FOU1(1, I) * FOU1(1, J)
            GERM(2) = GOU1(1, I) * GOU1(1, J)
            TERM(3) = FOU1(2, I) * FOU1(1, J)
            GERM(3) = GOU1(2, I) * GOU1(1, J)
            TERM(4) = FOU1(3, I) * FOU1(1, J)
            GERM(4) = GOU1(3, I) * GOU1(1, J)
            TERM(1) = FOU1(2, I) * FOU1(2, J)
            GERM(1) = GOU1(2, I) * GOU1(2, J)
            DO 110 K = 1, N
               IF (K .EQ. I .OR. K .EQ. J) GO TO 110
               CALL TOOSML (1, TERM, FOU(K))
               CALL TOOSML (1, GERM, GOU(K))
  110       CONTINUE
            FF(2) = FF(2) + TWO * TERM(2)
```

```
            FF(3) = FF(3) + TWO * TERM(3) * THREE
            FF(4) = FF(4) + TWO * (TERM(4) * FOUR + TERM(1) * THREE)
            GF(2) = GF(2) + TWO * GERM(2)
            GF(3) = GF(3) + TWO * GERM(3) * THREE
            GF(4) = GF(4) + TWO * (GERM(4) * FOUR + GERM(1) * THREE)
  120    CONTINUE
  130 CONTINUE
      DO 170 I = 1, N
         DO 160 J = I + 1, N
            DO 150 K = J + 1, N
               TERM(3) = FOU1(1, I) * FOU1(1, J) * FOU1(1, K)
               TERM(4) = FOU1(2, I) * FOU1(1, J) * FOU1(1, K)
               GERM(3) = GOU1(1, I) * GOU1(1, J) * GOU1(1, K)
               GERM(4) = GOU1(2, I) * GOU1(1, J) * GOU1(1, K)
               IF (N .GT. 3) THEN
                  DO 140 M = 1, N
                  IF (M .EQ. I .OR. M .EQ. J .OR. M .EQ. K) GO TO  140
                  CALL TOOSML (3, TERM, FOU(M))
                  CALL TOOSML (3, GERM, GOU(M))
  140             CONTINUE
            END IF
            FF(3) = FF(3) + SIX * TERM(3)
            FF(4) = FF(4) + SIX * TERM(4) * SIX
            GF(3) = GF(3) + SIX * GERM(3)
            GF(4) = GF(4) + SIX * GERM(4) * SIX
  150       CONTINUE
  160    CONTINUE
  170 CONTINUE
      IF (N .LE. 3) GO TO 230
      DO 220 I = 1, N
         DO 210 J = I + 1, N
            DO 200 K = J + 1, N
               DO 190 M = K + 1, N
      TERM(4) = FOU1(1, I) * FOU1(1, J) * FOU1(1, K) * FOU1(1, M)
      GERM(4) = GOU1(1, I) * GOU1(1, J) * GOU1(1, K) * GOU1(1, M)
               IF (N .GT. 4) THEN
                  DO 180 L = 1, N
         IF (L .EQ. I .OR. L .EQ. J .OR. L .EQ. K .OR. L .EQ. M)GOTO 180
                     CALL TOOSML (4, TERM, FOU(L))
                     CALL TOOSML (4, GERM, GOU(L))
  180             CONTINUE
               END IF
               FF(4) = FF(4) + FOUR * SIX * TERM(4)
               GF(4) = GF(4) + FOUR * SIX * GERM(4)
  190          CONTINUE
  200       CONTINUE
  210    CONTINUE
  220 CONTINUE
C
  230 CONTINUE
      PHI = EXP(-Z * Z) / SQRTPI
      PHI1 = -TWO * Z * PHI
      PHI2 = (FOUR * Z ** 2 - TWO) * PHI
      PHI3 = (-EIGHT * Z ** 3 + TWELVE * Z) * PHI
      PHI4 = (SIXTN * Z ** 2 * (Z ** 2 - THREE) + TWELVE) * PHI
      DERIV = PHI * (FF(4) + GF(4)) + FOUR * PHI1 * (FF(3) + GF(3))
     *  + SIX * PHI2 * (FF(2) + GF(2)) + FOUR * PHI3 * (FF(1) + GF(1))
     *  + PHI4 * FUNCN
      RETURN
      END
      SUBROUTINE ASSIGN (U, BP, FF)
```

```
C
C       ALGORITHM AS 251.3  APPL.STATIST. (1989), VOL.38, NO.3
C
C
C       COMPUTE DERIVATIVES OF NORMAL CDF'S.
C
      REAL FF(4)
      REAL U, U2, BP, HALF, ONE, THREE, SQ2PI, T1, T2, T3
      INTEGER I
      DATA HALF, ONE, THREE, SQ2PI /0.5, 1.0, 3.0, 2.50662827463100/
      DATA ZERO, UMAX, SMALL /0.0,  8.0, 0.1E-07/
      IF (ABS(U) .GT. UMAX) THEN
         DO 10 I = 1, 4
            FF(I) = ZERO
   10    CONTINUE
      ELSE
         U2 = U * U
         T1 = BP * EXP(-HALF * U2) / SQ2PI
         T2 = BP * T1
         T3 = BP * T2
         FF(1) = T1
         FF(2) = -U * T2
         FF(3) = (U2 - ONE) * T3
         FF(4) = (THREE - U2) * U * BP * T3
         DO 20 I = 1, 4
            IF(ABS(FF(I)) .LT. SMALL) FF(I) = ZERO
   20    CONTINUE
      END IF
      RETURN
      END
      SUBROUTINE WMAX(W1, W2, W3, DLG)
C
C       ALGORITHM AS 251.4  APPL.STATIST. (1989), VOL.38, NO.3
C
C
C       LARGEST ABSOLUTE VALUE OF QUADRATIC FUNCTION FITTED
C       TO THREE POINTS.
C
      REAL W1, W2, W3, DLG, QUAD, QLIM, QMIN, ONE, TWO, B2C
      DATA ONE, TWO, QMIN /1.0, 2.0, 0.00001/
      DLG = MAX( ABS(W1), ABS(W3) )
      QUAD = W1 - W2 * TWO + W3
      QLIM = MAX( ABS(W1 - W3) / TWO , QMIN)
      IF (ABS(QUAD) .LE. QLIM) RETURN
      B2C = (W1 - W3) / QUAD / TWO
      IF (ABS(B2C) .GE. ONE) RETURN
      DLG = MAX( DLG, ABS(W2 - B2C * QUAD * B2C / TWO) )
      RETURN
      END
      SUBROUTINE TOOSML (N, FF, F)
C
C       ALGORITHM AS 251.5  APPL.STATIST. (1989), VOL.38, NO.3
C
C
C       MULTIPLY FF(I) BY F FOR I = N TO 4.  SET TO ZERO IF TOO SMALL.
C
      REAL FF(4), F, ZERO, SMALL
      INTEGER N, I
      DATA ZERO, SMALL /0.0, 0.1E-12/
      DO 10 I = N, 4
         FF(I) = FF(I) * F
```

```
            IF (ABS(FF(I)) .LE. SMALL) FF(I) = ZERO
      10 CONTINUE
         RETURN
         END
         REAL FUNCTION ALNORM(X, UPPER)
C
C        ALGORITHM AS 66  APPL. STATIST. (1973) VOL.22, P.424
C
C        EVALUATES THE TAIL AREA OF THE STANDARDIZED NORMAL CURVE
C        FROM X TO INFINITY IF UPPER IS .TRUE. OR
C        FROM MINUS INFINITY TO X IF UPPER IS .FALSE.
C
         REAL LTONE, UTZERO, ZERO, HALF, ONE, CON, A1, A2, A3,
        $  A4, A5, A6, A7, B1, B2, B3, B4, B5, B6, B7, B8, B9,
        $  B10, B11, B12, X, Y, Z, ZEXP
         LOGICAL UPPER, UP
C
C        LTONE AND UTZERO MUST BE SET TO SUIT THE PARTICULAR COMPUTER
C        (SEE INTRODUCTORY TEXT)
C
         DATA LTONE, UTZERO /7.0, 18.66/
         DATA ZERO, HALF, ONE, CON /0.0, 0.5, 1.0, 1.28/
         DATA           A1,             A2,            A3,
        $               A4,             A5,            A6,
        $               A7
        $  /0.398942280444, 0.399903438504, 5.75885480458,
        $    29.8213557808,  2.62433121679, 48.6959930692,
        $    5.92885724438/
         DATA           B1,            B2,             B3,
        $               B4,            B5,             B6,
        $               B7,            B8,             B9,
        $              B10,           B11,            B12
        $  /0.398942280385,     3.8052E-8,  1.00000615302,
        $    3.98064794E-4, 1.98615381364, 0.151679116635,
        $    5.29330324926,  4.8385912808,  15.1508972451,
        $   0.742380924027,  30.789933034,  3.99019417011/
C
         ZEXP(Z) = EXP(Z)
C
         UP = UPPER
         Z = X
         IF (Z .GE. ZERO) GOTO 10
         UP = .NOT. UP
         Z = -Z
      10 IF (Z .LE. LTONE .OR. UP .AND. Z .LE. UTZERO) GOTO 20
         ALNORM = ZERO
         GOTO 40
      20 Y = HALF * Z * Z
         IF (Z .GT. CON) GOTO 30
C
         ALNORM = HALF - Z * (A1 - A2 * Y / (Y + A3 - A4 / (Y + A5 +
        $  A6 / (Y + A7))))
         GOTO 40
C
      30 ALNORM = B1 * ZEXP(-Y) / (Z - B2 + B3 / (Z + B4 + B5 / (Z -
        $  B6 + B7 / (Z + B8 - B9 / (Z + B10 + B11 / (Z + B12))))))
C
      40 IF (.NOT. UP) ALNORM = ONE - ALNORM
         RETURN
         END
         REAL FUNCTION PPND7 (P, IFAULT)
```

```
C
C        ALGORITHM AS241  APPL. STATIST. (1988) VOL. 37, NO. 3
C
C       PRODUCES THE NORMAL DEVIATE  Z  CORRESPONDING TO A GIVEN LOWER
C       TAIL AREA OF  P;  Z  IS ACCURATE TO ABOUT  1  PART IN 10**7.
C
C       THE HASH SUMS BELOW ARE THE SUMS OF THE MANTISSAS OF THE
C       COEFFICIENTS. THEY ARE INCLUDED FOR USE IN CHECKING
C       TRANSCRIPTION.
C
      INTEGER IFAULT
      REAL ZERO, ONE, HALF, SPLIT1, SPLIT2, CONST1, CONST2,
     *  A0, A1, A2, A3, B1, B2, B3, C0, C1, C2, C3, D1, D2,
     *  E0, E1, E2, E3, F1, F2, P, Q, R
      PARAMETER (ZERO = 0.0E0, ONE = 1.0E0, HALF = 0.5E0,
     *  SPLIT1 = 0.425E0,    SPLIT2 = 5.0E0,
     *  CONST1 = 0.180625E0, CONST2 = 1.6E0)
C
C     COEFFICIENTS FOR  P  CLOSE TO  1/2
      PARAMETER (A0 = 3.38713 27179E0,
     *           A1 = 5.04342 71938E1,
     *           A2 = 1.59291 13202E2,
     *           A3 = 5.91093 74720E1,
     *           B1 = 1.78951 69469E1,
     *           B2 = 7.87577 57664E1,
     *           B3 = 6.71875 63600E1)
C     HASH SUM AB    32.31845 77772
C
C     COEFFICIENTS FOR  P  NEITHER CLOSE TO  1/2  NOR  0 OR 1
      PARAMETER (C0 = 1.42343 72777E0,
     *           C1 = 2.75681 53900E0,
     *           C2 = 1.30672 84816E0,
     *           C3 = 1.70238 21103E-1,
     *           D1 = 7.37001 64250E-1,
     *           D2 = 1.20211 32975E-1)
C     HASH SUM CD    15.76149 29821
C
C     COEFFICIENTS FOR  P  NEAR  0 OR 1
      PARAMETER (E0 = 6.65790 51150E0,
     *           E1 = 3.08122 63860E0,
     *           E2 = 4.28682 94337E-1,
     *           E3 = 1.73372 03997E-2,
     *           F1 = 2.41978 94225E-1,
     *           F2 = 1.22582 02635E-2)
C     HASH SUM EF    19.40529 10204
C
      IFAULT = 0
      Q = P - HALF
      IF (ABS(Q) .LE. SPLIT1) THEN
         R = CONST1 - Q * Q
         PPND7 = Q * (((A3 * R + A2) * R + A1) * R + A0) /
     *               (((B3 * R + B2) * R + B1) * R + ONE)
         RETURN
      ELSE
         IF (Q .LT. 0) THEN
            R = P
         ELSE
            R = ONE - P
         ENDIF
         IF (R .LE. ZERO) THEN
            IFAULT = 1
```

```
            PPND7 = ZERO
            RETURN
         ENDIF
         R = SQRT(-LOG(R))
         IF (R .LE. SPLIT2) THEN
            R = R - CONST2
            PPND7 = (((C3 * R + C2) * R + C1) * R + C0) /
     *               ((D2 * R + D1) * R + ONE)
         ELSE
            R = R - SPLIT2
            PPND7 = (((E3 * R + E2) * R + E1) * R + E0) /
     *               ((F2 * R + F1) * R + ONE)
         ENDIF
         IF (Q .LT. 0) PPND7 = -PPND7
         RETURN
       ENDIF
       END
       PROGRAM MVTIN
C
C       COMPUTE PROBABILITY INTEGRAL FOR A MULTIVARIATE
C       NORMAL OR A MULTIVARIATE T WITH PRODUCT CORRELATION
C       STRUCTURE USING MVSTUD AND MVNPRD.
C
       DIMENSION A(50),B(50),BPD(50),INF(50),D(50),TMP(2)
       DATA  ZERO, ONE, EPS / 0.0, 1.0, 0.0001 /
       IERC = 0
       HNC = ZERO
C
C       START A NEW PROBLEM:-
C       ENTER N = NO. DIMENSIONS, AND NDF = DEG. OF FREEDOM
C       (FOR INFINITE D.F., ENTER NDF = 0)
C
    10 WRITE (*, 200)
       READ (*, *) N, NDF
       IF (N .LE. 0 .OR. NDF .LT. 0) STOP
       IF (N .GT. 50) THEN
          WRITE (*, 300)
          STOP
       ELSE
          XN = FLOAT(N)
       ENDIF
C
C       SPECIFY THE CORRELATION STRUCTURE:-
C       ENTER IX = 1 IF THERE IS A FIXED RHO
C          OR IX = 2 IF RHO IS DETERMINED FROM SAMPLE SIZES
C          OR IX = 3 IF RHO(I,J) = BPD(I)*BPD(J)
C          OR IX = 4 IF RHO(I,J) = 1 / (1 + SQRT(N))
C
    20 WRITE (*, 210)
       READ (*,*) IX
       IF (IX .LT. 1 .OR. IX .GT. 4) GO TO 10
       IF (IX .EQ. 1 .OR. IX .EQ. 4) THEN
          IF (IX .EQ. 4) THEN
              RHO = ONE / (ONE + SQRT(XN))
          ELSE
    30       WRITE (*, 220)
             READ (*, *) RHO
             IF (RHO .LT. ZERO .OR. RHO .GE. ONE) GO TO 30
          ENDIF
          SQTRHO = ZERO
          IF (RHO .GT. ZERO) SQTRHO = SQRT(RHO)
```

```
            DO 40 I = 1, N
               BPD(I) = SQTRHO
     40     CONTINUE
         ELSE
            IF (IX .EQ. 2) THEN
               WRITE (*, 310)
               READ (*, *) X0, (BPD(I), I = 1, N)
               DO 50 I = 1, N
                  BPD(I) = ONE / SQRT(ONE + X0 / BPD(I))
     50        CONTINUE
            ELSE
                  WRITE (*, 230) N
                  READ (*, *) (BPD(I), I = 1, N)
            ENDIF
         ENDIF
C
C       SPECIFY MEANS
C
     80 WRITE (*, 205)
         READ (*, *) IDEL
         IF (IDEL .EQ. 0) THEN
            DO 90 I = 1, N
               D(I) = ZERO
     90     CONTINUE
         ELSE
            NCM = 1
            IF (IDEL .NE. 1) NCM = N
            WRITE (*, 270) NCM
            READ (*, *) (D(I), I = 1, NCM)
            IF (NCM .EQ. N) GO TO 120
            DO 110 I = 1, N
               D(I) = D(1)
   110     CONTINUE
         ENDIF
   120 CONTINUE
         WRITE (*, 240)
         READ (*, *) NCM
            IF (NCM .NE. 1) NCM = N
         NSTRT = 1
   125 CONTINUE
C
C       SPECIFY THE VALUES FOR THE ENDPOINTS
C
               WRITE (*, 245)
               DO 130 I = NSTRT, NCM
   132         IF (NCM .EQ. 1) THEN
                   WRITE (*, 250)
               ELSE
                   WRITE (*, 260) I
               END IF
               READ (*, *) INF1, TMP(1), (TMP(J), J = 2, INF1)
               IF (INF1 .LT. 0 .OR. INF1 .GT. 2) GO TO 132
               INF(I) = INF1
               IF (INF1 .EQ. 0) THEN
                  B(I) = TMP(1)
                  A(I) = ZERO
               ELSE
                  IF (INF1 .EQ. 1) THEN
                     B(I) = ZERO
                     A(I) = TMP(1)
                  ELSE
```

```
                  B(I) = TMP(1)
                  A(I) = TMP(2)
               ENDIF
            ENDIF
            IF (NCM .EQ. 1) GO TO 131
  130    CONTINUE
  131    IF (NCM .EQ. 1) THEN
            DO 140 I = 2, N
               A(I) = A(1)
               B(I) = B(1)
               INF(I) = INF(1)
  140       CONTINUE
         ELSE
            CONTINUE
         ENDIF
      NX = N
      IF (N .GT. 4) NX = 4
      WRITE (*, 180) NX
      DO 150 I = 1, NX
         WRITE (*, 190) I, B(I), A(I), INF(I), BPD(I), D(I)
  150 CONTINUE
C
C        COMPUTE PROB
C
      WRITE (*, 290) N, NDF
      CALL MVSTUD(NDF,A,B,BPD,EPS,N,INF,D,IERC,HNC,PROB,BND,IFLT)
      WRITE (*, 280) PROB, BND, IFLT
  160 WRITE (*, 170)
      READ (*, *) NSTRT
      IF (NSTRT .GT. N) GO TO 160
      IF (NSTRT .GE. 1) GO TO 125
      GO TO 10
  170 FORMAT(1X,'Enter 1 to repeat with new values for limits',
     *    /1X,'   or J for new values for J-th to N-th limits, incl.',
     *    /1X '   or 0 for new problem')
  180 FORMAT(1X,'First',I2,' parameter sets:-',
     *       /,1X,' i      L_i      U_i   Type     b_i    \mu_i')
  190 FORMAT(1X, I2, 2F9.3, I6, 2F9.3)
  200 FORMAT(1X, 'Enter No. Dimensions, and D.o.F. (0 for MVN)',
     *  /8X, '(or enter 0, 0 to terminate)')
  205 FORMAT(1X, 'Enter 0 for central case,'/
     *       1X, '   or 1 for vector (D,...,D),'/
     *       1X, '   or 2 for vector (D_1,...,D_N)')
  210 FORMAT(1X, 'Enter 1 for equal-rho case,'/
     *       1X, '   or 2 to compute \rho_ij from sample sizes,'/
     *       1X, '   or 3 if \rho_ij = b_i*b_j,'/
     *       1X, '   or 4 for \rho = 1 / (1+sqrt(N))')
  220 FORMAT(1X, 'Enter common \rho')
  230 FORMAT(1X, 'Enter b_i, i = 1, ',I3)
  240 FORMAT(1X, 'Enter 1 for a common set of limits or 0 if not')
  245 FORMAT(1X,'For each variable enter type of interval and limits,'/
     *       1X,'Type = 0 iff [a,\infty], = 1 iff [-\infty,b]',
     *', = 2 iff [a,b]'/)
  250 FORMAT(1X, 'Enter common values of TYPE and limit(s)')
  260 FORMAT(1X, 'Enter TYPE and limit(s) for I =',I3)
  270 FORMAT(1X, 'Enter',I3,' mean value(s)')
  280 FORMAT(1X, 'PROB  =',F10.6,/,1X,'BOUND =',F10.6,/,1X,
     *  'FAULT =', I10, /)
  290 FORMAT(/,1X,'Computed results for N =',I3,', NDF =',I4,':-')
  300 FORMAT(1X, 'DIMENSION LIMITS EXCEEDED',/)
  310 FORMAT(1X, 'Enter sample sizes (control first)',/)
```

```
      END
      SUBROUTINE MVSTUD(NDF,A,B,BPD,ERRB,N,INF,D,IERC,HNC,PROB,
     *    BND,IFLT)
C
C        COMPUTE MULTIVARIATE STUDENT INTEGRAL,
C        USING MVNPRD (DUNNETT, APPL. STAT., 1989)
C        IF RHO(I,J) = BPD(I)*BPD(J).
C
C        IF RHO(I,J) HAS GENERAL STRUCTURE, USE
C        MULNOR (SCHERVISH, APPL. STAT., 1984) AND REPLACE
C        CALL MVNPRD(A,B,BPD,EPS,N,INF,IERC,HNC,PROB,BND,IFLT)
C        BY CALL MULNOR(A,B,SIG,EPS,N,INF,PROB,BND,IFLT).
C
C        AUTHOR: C.W. DUNNETT, MCMASTER UNVERSITY
C
C        BASED ON ADAPTIVE SIMPSON'S RULE ALGORITHM
C        DESCRIBED IN SHAMPINE & ALLEN: "NUMERICAL
C        COMPUTING", (1974), PAGE 240.
C
C        PARAMETERS ARE SAME AS IN ALGORITHM AS 251
C        IN APPL. STAT. (1989), VOL. 38: 564-579
C        WITH THE FOLLOWING ADDITIONS:
C             NDF   INTEGER      INPUT  DEGREES OF FREEDOM
C             D     REAL ARRAY   INPUT  NON-CENTRALITY VECTOR
C        (PUT NDF = 0 FOR INFINITE D.F.)
C
      INTEGER NN
      PARAMETER (NN=50)
      DIMENSION A(*),B(*),BPD(*),INF(*),D(*),F(3),AA(NN),BB(NN)
      DATA ZERO,HALF,TWO,THREE,FOUR / 0.0, 0.5, 2.0, 3.0, 4.0 /
      DO 10 I = 1, N
         AA(I) = A(I) - D(I)
         BB(I) = B(I) - D(I)
   10 CONTINUE
      IF (NDF .LE. 0) THEN
         CALL MVNPRD(AA,BB,BPD,ERRB,N,INF,IERC,HNC,PROB,BND,IFLT)
         RETURN
      ENDIF
      BND   = ZERO
      IFLT  =  0
      MAXDF = 150
      ERB2  = ERRB
C
C        CHECK IF D.F. EXCEED MAXDF; IF YES, THEN PROB
C        IS COMPUTED BY QUADRATIC INTERPOLATION ON 1./D.F.
C
      IF (NDF .LE. MAXDF) GO TO 20
      CALL MVNPRD(AA,BB,BPD,ERB2,N,INF,IERC,HNC,F(1),BND,IFLT)
      NF    =  MAXDF / 2
      CALL SIMPSN(NF,A,B,BPD,ERB2,N,INF,D,IERC,HNC,F(3),BND,IFLT)
      NF    =  NF * 2
      CALL SIMPSN(NF,A,B,BPD,ERB2,N,INF,D,IERC,HNC,F(2),BND,IFLT)
      XX    =  FLOAT(NF) / FLOAT(NDF)
      AX    =  F(3) - F(2)*TWO + F(1)
      BX    =  F(2)*FOUR - F(3) - F(1)*THREE
      PROB  =  F(1) + XX * (AX * XX + BX) * HALF
      RETURN
   20 CALL SIMPSN (NDF,A,B,BPD,ERB2,N,INF,D,IERC,HNC,PROB,BND,IFLT)
      RETURN
      END
      SUBROUTINE SIMPSN (NDF,A,B,BPD,ERRB,N,INF,D,IERC,HNC,PROB,
```

```
     *   BND,IFLT)
C
C        STUDENTIZES A MULTIVARIATE INTEGRAL USING SIMPSON'S RULE.
C
      DIMENSION A(*),B(*),BPD(*),INF(*),D(*),
     *   FV(5),F1T(30),F2T(30),F3T(30),
     *   LDIR(30),PSUM(30),ESTT(30),ERRR(30),GV(5),G1T(30),G2T(30),
     *   G3T(30),GSUM(30)
      DATA ZERO,HALF,ONE,ONEP5,TWO,FOUR,SIX,DXMIN /0.0,0.5,1.0,1.5,
     *   2.0,4.0,6.0,0.000004/
      PROB    =  ZERO
      BOUNDA  =  ZERO
      BOUNDG  =  ZERO
      IFLAG   =  0
      IER     =  0
      START   =  -ONE
      DAX     =   ONE
      ERB2    =   ERRB * HALF
      EPS1    =   ERB2 * HALF
      CALL FUN (ZERO,NDF,A,B,BPD,ERB2,N,INF,D,F0,G0,IERC,HNC,IER)
   10 FV(1)   =  ZERO
      GV(1)   =  ZERO
      ERROR   =  ZERO
      DA      =  DAX
      LVL     =  1
      Z3      =  START + HALF*DA
      CALL FUN(Z3,NDF,A,B,BPD,ERB2,N,INF,D,FV(3),GV(3),IERC,HNC,IER)
      FV(5)   =  F0
      GV(5)   =  G0
      WT      =  ABS(DA) / SIX
      CONTRB  =  WT * (FV(1) + FOUR * FV(3) + FV(5))
      CONTRG  =  WT * (GV(1) + FOUR * GV(3) + GV(5))
      LDIR(LVL)  = 2
      PSUM(LVL)  = ZERO
      GSUM(LVL)  = ZERO
C
C        BISECT INTERVAL; COMPUTE ESTIMATES FOR EACH HALF.
C
   20 DX      =  HALF * DA
      WT      =  ABS(DX) / SIX
      Z2      =  START + HALF * DX
      CALL FUN(Z2,NDF,A,B,BPD,ERB2,N,INF,D,FV(2),GV(2),IERC,HNC,IER)
      Z4      =  START + ONEP5 * DX
      CALL FUN(Z4,NDF,A,B,BPD,ERB2,N,INF,D,FV(4),GV(4),IERC,HNC,IER)
      ESTL    =  WT * (FV(1) + FOUR * FV(2) + FV(3))
      ESTR    =  WT * (FV(3) + FOUR * FV(4) + FV(5))
      ESTGL   =  WT * (GV(1) + FOUR * GV(2) + GV(3))
      ESTGR   =  WT * (GV(3) + FOUR * GV(4) + GV(5))
      SUM     =  ESTL  +  ESTR
      SUMG    =  ESTGL +  ESTGR
      DLG     =  ABS(CONTRB - SUM)
      ERRL    =  DLG
C
C        STOP BISECTING WHEN ACCURACY SUFFICIENT, OR IF
C        INTERVAL TOO NARROW OR BISECTED TOO OFTEN.
C
   30 IF (DLG .LE. EPS1) GO TO 50
      IF (ABS(DX) .LE. DXMIN .OR. LVL .GE. 30) GO TO 40
C
C        RAISE LEVEL.  STORE INFORMATION FOR RIGHT HALF
C        AND APPLY SIMPSON'S RULE TO LEFT HALF.
```

```
C
      LVL     =  LVL + 1
      LDIR(LVL)  =  1
      F1T(LVL)   =  FV(3)
      F2T(LVL)   =  FV(4)
      F3T(LVL)   =  FV(5)
      G1T(LVL)   =  GV(3)
      G2T(LVL)   =  GV(4)
      G3T(LVL)   =  GV(5)
      DA      =  DX
      FV(5)   =  FV(3)
      FV(3)   =  FV(2)
      GV(5)   =  GV(3)
      GV(3)   =  GV(2)
      ESTT(LVL)  =  ESTR
      CONTRB  =  ESTL
      CONTRG  =  ESTGL
      EPS1    =  EPS1 * HALF
      ERRR(LVL)  =  EPS1
      GO TO 20
C
C        ACCEPT APPROXIMATE VALUE FOR INTERVAL.
C
   40 IFLAG   =  11
   50 ERROR   =  ERROR + ERRL
   60 IF (LDIR(LVL) .EQ. 1) GO TO 70
      SUM     =  SUM + PSUM(LVL)
      SUMG    =  SUMG + GSUM(LVL)
      LVL     =  LVL - 1
      IF (LVL .GT. 0) GO TO 60
      CONTRB  =  SUM
      CONTRG  =  SUMG
      LVL     =  1
      DLG     =  ERROR
      GO TO 80
C
C        RESTORE SAVED INFORMATION TO PROCESS RIGHT HALF.
C
   70 PSUM(LVL)  =  SUM
      GSUM(LVL)  =  SUMG
      LDIR(LVL)  =  2
      DA      =  DAX / TWO**(LVL-1)
      START      =  START + DX * TWO
      FV(1)      =  F1T(LVL)
      FV(3)      =  F2T(LVL)
      FV(5)      =  F3T(LVL)
      GV(1)      =  G1T(LVL)
      GV(3)      =  G2T(LVL)
      GV(5)      =  G3T(LVL)
      CONTRB     =  ESTT(LVL)
      EXCESS     =  EPS1 - DLG
      EPS1       =  ERRR(LVL)
      IF (EXCESS .GT. ZERO) EPS1 = EPS1 + EXCESS
      GO TO 20
   80 PROB       =  PROB + CONTRB
      BOUNDG     =  BOUNDG + CONTRG
      BOUNDA     =  BOUNDA + DLG
      IF (Z4 .LE. ZERO) GO TO 90
      IF (IFLT .EQ. 0) IFLT = IER
      IF (IFLT .EQ. 0) IFLT = IFLAG
      BOUNDA     =  BOUNDA + BOUNDG
```

```
      IF (BND .LT. BOUNDA) BND = BOUNDA
      RETURN
   90 EPS1       =  ERB2 * HALF
      EXCESS     =  EPS1 - BND
      IF (EXCESS .GT. ZERO) EPS1 = EPS1 + EXCESS
      START      =  ONE
      DAX        = -ONE
      GO TO 10
      END
      FUNCTION SDIST(Y,N)
C
C        COMPUTE Y**(N/2 - 1) EXP(-Y) / GAMMA(N/2)
C
C                (Revised: 1994-01-19)
C
      DATA ZERO, HALF, ONE, X23 / 0.0, 0.5, 1.0, -23.0 /
      DATA SQRTPI / 1.77245385090552 /
      SDIST      =  ZERO
      IF (Y .LE. ZERO) RETURN
      JJ         =  N/2 - 1
      JK         =  2 * JJ - N + 2
      JKP        =  JJ - JK
      SDIST      =  ONE
      IF (JK .LT. 0) SDIST = SDIST / SQRT(Y) / SQRTPI
      IF (JKP .EQ. 0) GO TO 20
      XN         =  FLOAT(N) * HALF
      TEST       =  ALOG(Y) - Y / FLOAT(JKP)
      IF ( TEST .LT. X23 ) THEN
         SDIST = ZERO
         RETURN
      ENDIF
      SDIST = ALOG ( SDIST )
      DO 10 J = 1, JKP
         XN    =   XN - ONE
         SDIST   =  SDIST + TEST - ALOG(XN)
   10 CONTINUE
      IF ( SDIST .LT. X23 ) THEN
          SDIST  =  ZERO
      ELSE
          SDIST  =  EXP( SDIST )
      ENDIF
      RETURN
   20 SDIST      =  SDIST * EXP(-Y)
      RETURN
      END
      SUBROUTINE FUN (Z,NDF,H,HL,BPD,ERB2,N,INF,D,F0,G0,IERC
     *   ,HNC,IER)
      INTEGER NN
      PARAMETER (NN=50)
      DIMENSION A(NN),B(NN),H(*),HL(*),BPD(*),INF(*),D(*)
      DATA  ZERO, ONE, TWO, SMALL / 0.0, 1.0, 2.0, 1.0E-08 /
      F0    =  ZERO
      G0    =  ZERO
      IF (Z .LE. -ONE .OR. Z .GE. ONE) RETURN
      DF    =  FLOAT(NDF)
      ARG   =  (ONE + Z) / (ONE - Z)
      TERM = ARG * DF * TWO / (ONE-Z)**2 * SDIST(DF/TWO*ARG*ARG,NDF)
      IF (TERM .LE. SMALL) RETURN
      DO 10 I = 1, N
         A(I) = ARG * H(I)   - D(I)
         B(I) = ARG * HL(I)  - D(I)
```

```
      10 CONTINUE
         CALL MVNPRD (A,B,BPD,ERB2,N,INF,IERC,HNC,PROB,BND,IFLT)
         IF (IER .EQ. 0) IER = IFLT
         G0  =  TERM * BND
         F0  =  TERM * PROB
         RETURN
         END

C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C Charles Dunnett
C Dept. of Mathematics and Statistics
C McMaster University
C Hamilton, Ontario L8S 4K1
C Canada
C E-mail: dunnett@mcmaster.ca
C Tel.: (905) 525-9140 (Ext. 27104)
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
      SUBROUTINE GESAT(NVAR, NP, ISET, NUM, SATMOD, IFAULT)
C
C        ALGORITHM AS 252.1  APPL.STATIST. (1990), VOL.39, NO.1
C
C        Obtains the description of all interactions
C        in the saturated model
C
C     Auxiliary routine required: COMBO from AS 160 (included here)
C
      INTEGER NVAR, NP, IFAULT, ISET(NVAR), NUM(NVAR), SATMOD(NVAR, NP)
      INTEGER NVMAX, I, J, K, M, N
      LOGICAL LAST
      DATA NVMAX / 15 /
C
      IFAULT = 1
      IF (NVAR .LT. 1 .OR. NVAR .GT. NVMAX) RETURN
      IFAULT = 2
      I = 2 ** NVAR - 1
      IF (NP .LT. I) RETURN
      IFAULT = 0
      DO 20 I = 1, NVAR
         DO 10 J = 1, NP
            SATMOD(I, J) = 0
   10    CONTINUE
   20 CONTINUE
      M = 0
      N = 0
      DO 50 K = 1, NVAR
         I = NVAR - K + 1
         LAST = .TRUE.
   30    CALL COMBO(ISET, NVAR, I, LAST)
         IF ( .NOT. LAST) THEN
            N = N + 1
            DO 40 J = 1, I
               SATMOD(J, N) = ISET(J)
   40       CONTINUE
            GO TO 30
         END IF
         M = M + 1
         NUM(M) = N
   50 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE CLAGEN(NVAR, NP, NUM, SATMOD, IPAR, DES, NG, INTG,
     *                  IGEN, ISET, JSET, IDES, JDES, NCON, CONFIG,
     *                  IFAULT)
C
C        ALGORITHM AS 252.2  APPL.STATIST. (1990), VOL.39, NO.1
C
C        Obtains the generating class or dual generating class
C        of the model defined in terms of interactions which
C        must be present/absent or which must be added/omitted
C        to/from given model
C
      INTEGER NVAR, NP, IPAR, NG, IGEN, NCON, IFAULT, NUM(NVAR),
     *        SATMOD(NVAR, NP), ISET(NVAR), JSET(NVAR), INTG(NVAR, NG),
     *        CONFIG(NVAR, NP)
      LOGICAL DES(NP), IDES(NP), JDES(NP)
      INTEGER I, IP, J, J1, J2, K, KZ, LF, M, M1, M2, M3, M4, N, NI,
```

```
     *        NR, NR1, NR2, NST, NVMAX
      LOGICAL IND, LG, LW, STER
      EXTERNAL INSET
      LOGICAL INSET
C
C        NVMAX:  Set to 15 for 16 bit integers - see text.
C
      DATA NVMAX / 15 /
C
C        Check input parameters
C
      IFAULT = 1
      IF (NVAR .LT. 1 .OR. NVAR .GT. NVMAX) RETURN
      IFAULT = 2
      I = 2 ** NVAR - 1
      IF (NP .LT. I) RETURN
      IFAULT = 3
      IF (IPAR .LT. 1 .OR. IPAR .GT. 4) RETURN
      STER = IPAR .EQ. 1 .OR. IPAR .EQ. 3
      IFAULT = 4
      IF (NG .LT. 1) RETURN
      IFAULT = 5
      DO 20 I = 1, NG
         DO 10 J = 1, NVAR
            K = INTG(J, I)
            IF (K .LT. 0 .OR. K .GT. NVAR) RETURN
   10    CONTINUE
   20 CONTINUE
         IFAULT = 6
         IF (IGEN .NE. 1 .AND. IGEN .NE. 2) RETURN
         IFAULT = 0
C
C        Initialize workspaces and variables
C
      LG = IPAR .EQ. 3 .OR. IPAR .EQ. 4
      IF ( .NOT. LG) THEN
         DO 30 I = 1, NP
            DES(I) = .NOT. STER
   30    CONTINUE
      END IF
      DO 40 I = 1, NP
         LW = DES(I)
         IDES(I) = LW
         JDES(I) = LW
   40 CONTINUE
      IF (STER) THEN
         K = -1
         KZ = 1
      ELSE
         K = 1
         KZ = NVAR
      END IF
C
C        Find full description of the model
C
      DO 140 NI = 1, NG
         DO 50 I = 1, NVAR
            JSET(I) = 0
   50    CONTINUE
         NR = 0
         DO 60 I = 1, NVAR
```

```
               J = INTG(I, NI)
               IF (J .EQ. 0) GO TO 70
               NR = NR + 1
               JSET(NR) = J
      60    CONTINUE
      70    N = NR
            IF (N .EQ. 0) THEN
               IF (STER) GO TO 140
               DO 80 I = 1, NP
                  IDES(I) = .FALSE.
      80       CONTINUE
               GO TO 150
            END IF
      90    IF (N .EQ. NVAR) THEN
               M3 = 1
               M4 = 1
            ELSE
               J = NVAR - N
               M3 = NUM(J) + 1
               M4 = NUM(J + 1)
            END IF
            DO 120 I = M3, M4
               IDES(I) = STER
               IF (JDES(I) .NEQV. STER) THEN
                  IP = 0
                  DO 100 J = 1, NVAR
                     IF (SATMOD(J, I) .EQ. 0) GO TO 110
                     IP = IP + 1
                     ISET(IP) = SATMOD(J, I)
     100             CONTINUE
     110             IF (STER) THEN
                        IND = INSET(NR, JSET, IP, ISET)
                     ELSE
                        IND = INSET(IP, ISET, NR, JSET)
                     END IF
                     IDES(I) = JDES(I)
                     IF (IND) IDES(I) = STER
               END IF
     120    CONTINUE
            N = N + K
            IF (STER) THEN
               IF (N .GE. KZ) GO TO 90
            ELSE
               IF (N .LE. KZ) GO TO 90
            END IF
            DO 130 J = 1, NP
               JDES(J) = IDES(J)
     130    CONTINUE
     140 CONTINUE
     150 IF (LG) GO TO 170
C
C           If the model is defined by interactions which must be
C           present/absent, store its full description in DES
C
         DO 160 I = 1, NP
            DES(I) = IDES(I)
     160 CONTINUE
C
C           Find the (dual) generating class if the model is saturated
C
     170 STER = IGEN .EQ. 1
```

```
      NCON = 1
      LW = IDES(1)
      LG = LW .EQV. STER
      DO 180 I = 1, NVAR
         K = 0
         IF (LG) K = I
         CONFIG(I, 1) = K
  180 CONTINUE
      IF (LW) RETURN
C
C        Find the (dual) generating class if the model is unsaturated
C
      LW = .TRUE.
      NCON = 0
      NR1 = NVAR - 1
      IF (STER) THEN
         NR2 = NR1
         NR1 = 1
         NST = 1
      ELSE
         NR2 = 1
         NST = -1
      END IF
      DO 240 M = NR1, NR2, NST
         I = NVAR - M
         LF = 0
         M1 = NUM(M) + 1
         M2 = NUM(M + 1)
         DO 230 J = M1, M2
            IF (STER .EQV. IDES(J)) THEN
               IF ( .NOT. LW) THEN
                  DO 190 J1 = 1, I
                     ISET(J1) = SATMOD(J1, J)
  190             CONTINUE
                  IF (I .NE. NR2) THEN
                     M3 = NUM(M - NST) + 1
                     M4 = NUM(M - NST + 1)
                     DO 210 J1 = M3, M4
                        IF (IDES(J1) .EQV. STER) THEN
                           IP = I + NST
                           DO 200 J2 = 1, IP
                              JSET(J2) = SATMOD(J2, J1)
  200                      CONTINUE
                           IF ( .NOT. STER) THEN
                              LG = INSET(I, ISET, IP, JSET)
                           ELSE
                              LG = INSET(IP, JSET, I, ISET)
                           END IF
                           IF (LG) GO TO 230
                        END IF
  210                CONTINUE
                  END IF
               END IF
               NCON = NCON + 1
               DO 220 J1 = 1, NVAR
                  CONFIG(J1, NCON) = SATMOD(J1, J)
  220          CONTINUE
            ELSE
               LF = LF + 1
            END IF
  230    CONTINUE
```

```
            IF (LF .EQ. 0) RETURN
            J = M2 - M1 + 1
            LW = LF .EQ. J
      240 CONTINUE
          IF (NCON .EQ. 0) NCON = 1
          RETURN
          END
C
C
          LOGICAL FUNCTION INSET(N, ISET, M, JSET)
C
C           ALGORITHM AS 252.3  APPL.STATIST. (1990), VOL.39, NO.1
C
          INTEGER N, M, ISET(N), JSET(M)
          INTEGER I, J
C
          INSET = .FALSE.
          DO 20 I = 1, M
             DO 10 J = 1, N
                IF (JSET(I) .EQ. ISET(J)) GO TO 20
       10    CONTINUE
             RETURN
       20 CONTINUE
          INSET = .TRUE.
          RETURN
          END
C
C
          SUBROUTINE COMBO(ISET, N, M, LAST)
C
C     ALGORITHM AS 160.2  APPL. STATIST. (1981) VOL.30, NO.1
C
C     Subroutine to generate all possible combinations of M of the
C     integers from 1 to N in a stepwise fashion.   Prior to the first
C     call, LAST should be set to .FALSE.   Thereafter, as long as LAST
C     is returned .FALSE., a new valid combination has been generated.
C     When LAST goes .TRUE., there are no more combinations.
C
          LOGICAL LAST
          INTEGER N, M, ISET(M)
C
          IF (LAST) GO TO 110
C
C     Get next element to increment
C
          K = M
      100 L = ISET(K) + 1
          IF (L + M - K .LE. N) GO TO 150
          K = K - 1
C
C     See if we are done
C
          IF (K .LE. 0) GO TO 130
          GO TO 100
C
C     Initialize first combination
C
      110 DO 120 I = 1, M
      120 ISET(I) = I
      130 LAST = .NOT. LAST
          RETURN
```

```
C
C     Fill in remainder of combination.
C
150   DO 160 I = K, M
        ISET(I) = L
        L = L + 1
160   CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE RCM(TAB, FIT, FF, L, LR, LC, MU, NU, PHI, DETAIL, TOL,
     *               RWT, CWT, WRK, WRK2, ITER, NTAB, NR, NC, NM, NW,
     *               NW2, MAXIT, IFAULT)
C
C        ALGORITHM AS 253.1  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Maximum likelihood estimation of the RC(M) association model
C
      INTEGER FF(NTAB), ITER, NTAB, NR, NC, NM, NW, NW2, MAXIT, IFAULT
      REAL TAB(NTAB), FIT(NTAB), L, LR(NR), LC(NC), MU( * ), NU( * ),
     *     PHI( * ), DETAIL(MAXIT), TOL, RWT(NR), CWT(NC), WRK(NW),
     *     WRK2(NW2)
      INTEGER IPT(8), I, ICON, II, J, MNRC, JOB
      REAL SALE, TOT, ZERO
      DATA ZERO / 0.0 /
C
C        Check for errors
C
      IFAULT = 0
      IF (NM .GT. (NR - 1) .OR. NM .GT. (NC - 1)) IFAULT = 1
      IF (NM .LT. 0) IFAULT = 1
      IF (TOL .LE. ZERO) IFAULT = 2
      IF (NR .LT. 2 .OR. NC .LT. 2) IFAULT = 3
      DO 10 J = 1, NTAB
         IF (TAB(J) .LT. ZERO) IFAULT = 4
   10 CONTINUE
      IF (NTAB .LT. (NR * NC)) IFAULT = 6
      IF (NW .LT. (2 * (NR + NC) + (NR * NC))) IFAULT = 7
      IF (NW2 .LT. ((MAX(NR, NC)) + (NC + 1) + (NC ** 2) + NR +
     *    NC)) IFAULT = 7
      DO 20 I = 1, NTAB
         IF (FF(I) .NE. 1 .AND. FF(I) .NE. 0) IFAULT = 8
   20 CONTINUE
      IF (IFAULT .NE. 0) RETURN
C
C        Set pointers to locations in work space
C
      MNRC = MAX(NR, NC)
C
      IPT(1) = NR + 1
      IPT(2) = NR + NC + 1
      IPT(3) = 2 * NR + NC + 1
      IPT(4) = 2 * (NR + NC) + 1
      IPT(5) = MNRC * NC + 1
      IPT(6) = (MNRC * NC) + NC + 2
      IPT(7) = (MNRC * NC) + (NC * NC) + NC + 2
      IPT(8) = (MNRC * NC) + (NC * NC) + NR + NC + 2
C
C        Get the marginal totals for the observed table
C
      CALL MARGIN(TAB, WRK(1), WRK(IPT(1)), TOT, NR, NC)
C
C        Compute intial parameter estimates
C
      JOB = 21
      CALL START(TAB, L, LR, LC, MU, NU, PHI, RWT, CWT, WRK2(1),
     *           WRK2(IPT(5)), WRK2(IPT(6)), WRK2(IPT(7)), WRK2(IPT(8)),
     *           WRK(IPT(4)), WRK(IPT(2)), WRK(IPT(3)), NR, NC, NM,
     *           MNRC, JOB, IFAULT)
      IF (IFAULT .NE. 0) THEN
         IFAULT = 5
```

```
            RETURN
         END IF
C
C           *****  Iterations  *****
C
         ITER = MAXIT
         DO 30 II = 1, MAXIT
C
C           Update the LAMBDAs
C
            CALL LAMFIT(1, TAB, FIT, FF, L, LR, LC, MU, NU, PHI, WRK(1),
     *                  WRK(IPT(1)), WRK(IPT(2)), WRK(IPT(3)), TOT, NR, NC,
     *                  NR, NC, NM)
            CALL LAMFIT(2, TAB, FIT, FF, L, LC, LR, NU, MU, PHI, WRK(1),
     *                  WRK(IPT(1)), WRK(IPT(2)), WRK(IPT(3)), TOT, NR, NC,
     *                  NC, NR, NM)
C
            IF (NM .GT. 0) THEN
C
C            Update the MU's and NU's
C
               CALL SCRFIT(1, TAB, FIT, FF, L, LR, LC, MU, NU, PHI, NR, NC,
     *                     NM, NR, NC)
               CALL SCRFIT(2, TAB, FIT, FF, L, LC, LR, NU, MU, PHI, NR, NC,
     *                     NM, NC, NR)
C
C            Update the PHI's
C
               CALL PHIFIT(TAB, FIT, FF, L, LR, LC, MU, NU, PHI, NR, NC,
     *                     NM)
C
C            Centre the MU's and NU's about zero
C
               CALL CENTRE(L, LR, LC, MU, NU, PHI, RWT, CWT, NR, NC, NM)
C
            END IF
C
C           Check for convergence
C
            CALL CONV(TAB, FIT, FF, WRK, L, LR, LC, MU, NU, PHI, IPT, TOT,
     *                TOL, SALE, NR, NC, NM, NW, ICON)
C
            DETAIL(II) = SALE
            IF (ICON .EQ. 1 .AND. NM .GT. 0) THEN
C
C            Rescale the MU's and NU's so that they satisfy the
C            identifying restrictions
C
               CALL SCALE(MU, NU, PHI, RWT, CWT, WRK2(1), WRK2(IPT(5)),
     *                    WRK2(IPT(6)), WRK2(IPT(7)), WRK2(IPT(8)), NR, NC,
     *                    NM, MNRC, JOB, IFAULT)
               IF (IFAULT .NE. 0) IFAULT = 5
C
               ITER = II
               RETURN
            ELSE IF (ICON .EQ. 1) THEN
               ITER = II
               RETURN
            END IF
   30    CONTINUE
C
```

```
      IFAULT = 9
      RETURN
      END
      SUBROUTINE MARGIN(T, R, C, N, NR, NC)
C
C        ALGORITHM AS 253.2  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Compute total sample size, and row and column marginal totals
C
      INTEGER NR, NC
      REAL T(NR, NC), R(NR), C(NC), N
      INTEGER I, J
      REAL ZERO
      DATA ZERO / 0.0 /
C
      DO 10 I = 1, NR
         R(I) = ZERO
   10 CONTINUE
      DO 20 J = 1, NC
         C(J) = ZERO
   20 CONTINUE
      N = ZERO
C
      DO 40 J = 1, NC
         DO 30 I = 1, NR
            R(I) = R(I) + T(I, J)
            C(J) = C(J) + T(I, J)
            N = N + T(I, J)
   30    CONTINUE
   40 CONTINUE
C
      RETURN
      END
      SUBROUTINE EXPTAB(TAB, FIT, FF, LAMBDA, LR, LC, MU, NU, PHI, NR,
     *                  NC, NM)
C
C        ALGORITHM AS 253.3  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Compute expected frequencies
C
      INTEGER FF(NR, NC), NR, NC, NM
      REAL TAB(NR, NC), FIT(NR, NC), LAMBDA, LR(NR), LC(NC), MU(NR, * ),
     *     NU(NC, * ), PHI( * )
      INTEGER I, J, L
      REAL ONE, X, ZERO
      DATA ZERO / 0.0 / ONE / 1.0 /
C
      DO 30 J = 1, NC
         DO 20 I = 1, NR
            X = ZERO
            IF (NM .GT. 0) THEN
               DO 10 L = 1, NM
                  X = X + MU(I, L) * NU(J, L) * PHI(L)
   10          CONTINUE
            END IF
            FIT(I, J) = (EXP(LAMBDA + LR(I) + LC(J) + X) * FF(I, J)) +
     *                  ((ONE - FF(I, J)) * TAB(I, J))
   20    CONTINUE
   30 CONTINUE
C
      RETURN
```

```
      END
      SUBROUTINE START(TAB, LAMBDA, LR, LC, M, N, P, RWT, CWT, A, W, V,
     *                 WORK, E, LTAB, LRM, LCM, NR, NC, NM, MNRC, JOB,
     *                 IFAULT)
C
C        ALGORITHM AS 253.4  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Compute initial parameter estimates
C
      INTEGER NR, NC, NM, MNRC, JOB, IFAULT
      REAL TAB(NR, NC), LAMBDA, LR(NR), LC(NC), M(NR, * ), N(NC, * ),
     *     P( * ), RWT(NR), CWT(NC), A(MNRC, NC), W( * ), V(NC, NC),
     *     WORK(NR), E(NC), LTAB(NR, NC), LRM(NR), LCM(NC)
      INTEGER I, IA, J, L
      REAL EPS, ZERO
      DATA EPS / 0.1 / ZERO / 0.0 /
C
      DO 20 J = 1, NC
         DO 10 I = 1, NR
            IF (TAB(I, J) .LE. EPS) THEN
               LTAB(I, J) = LOG(EPS)
            ELSE
               LTAB(I, J) = LOG(TAB(I, J))
            END IF
   10    CONTINUE
   20 CONTINUE
C
      CALL MARGIN(LTAB, LRM, LCM, LAMBDA, NR, NC)
      LAMBDA = LAMBDA / (NR * NC)
C
      DO 30 I = 1, NR
         LR(I) = LRM(I) / NC - LAMBDA
   30 CONTINUE
      DO 40 J = 1, NC
         LC(J) = LCM(J) / NR - LAMBDA
   40 CONTINUE
C
      IF (NM .GE. 1) THEN
         DO 60 J = 1, NC
            DO 50 I = 1, NR
               A(I, J) = LTAB(I, J) - (LAMBDA + LR(I) + LC(J))
               A(I, J) = A(I, J) * SQRT(RWT(I)) * SQRT(CWT(J))
   50       CONTINUE
   60    CONTINUE
C
         IF (MNRC .GT. NR) THEN
            IA = NR + 1
            DO 80 J = 1, NC
               DO 70 I = IA, MNRC
                  A(I, J) = ZERO
   70          CONTINUE
   80       CONTINUE
         END IF
C
C        Call LINPACK SVD subroutine
C
         CALL SSVDC(A, MNRC, NR, NC, W, E, A, MNRC, V, NC, WORK, JOB,
     *              IFAULT)
C
         DO 110 L = 1, NM
            P(L) = W(L)
```

```
            DO 90 I = 1, NR
               M(I, L) = A(I, L) / SQRT(RWT(I))
   90       CONTINUE
C
            DO 100 J = 1, NC
               N(J, L) = V(J, L) / SQRT(CWT(J))
  100       CONTINUE
  110    CONTINUE
      END IF
C
      RETURN
      END
      SUBROUTINE LAMFIT(IFLAG, TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2,
     *                  P, RM, CM, ERM, ECM, TOT, NR, NC, N1, N2, NM)
C
C        ALGORITHM AS 253.5  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Update the estimates of the lambdas
C        IFLAG =1 => Update row parameters, else do column parameters
C
      INTEGER FF(NR, NC), IFLAG, NR, NC, NM, N1, N2
      REAL TAB(NR, NC), FIT(NR, NC), LAMBDA, LAM1(N1), LAM2(N2),
     *     M1(N1, * ), M2(N2, * ), P( * ), RM(NR), CM(NC), ERM(NR),
     *     ECM(NC), TOT
      INTEGER II, NN
      REAL ETOT
C
      IF (IFLAG .EQ. 1) THEN
         CALL EXPTAB(TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2, P, NR,
     *               NC, NM)
         NN = NR
      ELSE
         NN = NC
         CALL EXPTAB(TAB, FIT, FF, LAMBDA, LAM2, LAM1, M2, M1, P, NR,
     *               NC, NM)
      END IF
C
      CALL MARGIN(FIT, ERM, ECM, ETOT, NR, NC)
C
      DO 10 II = 1, NN
         IF (IFLAG .EQ. 1) THEN
            LAM1(II) = LAM1(II) + (RM(II) - ERM(II)) / ERM(II)
         ELSE
            LAM1(II) = LAM1(II) + (CM(II) - ECM(II)) / ECM(II)
         END IF
   10 CONTINUE
C
      RETURN
      END
      SUBROUTINE SCRFIT(IFLAG, TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2,
     *                  P, NR, NC, NM, N1, N2)
C
C        ALGORITHM AS 253.6  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Update the estimates of the MU's and NU's
C        IFLAG = 1 => Update MU's, else do NU's
C
      INTEGER FF(NR, NC), IFLAG, NR, NC, NM, N1, N2
      REAL TAB(NR, NC), FIT(NR, NC), LAMBDA, LAM1(N1), LAM2(N2),
     *     M1(N1, * ), M2(N2, * ), P( * )
      INTEGER I1, I2, L
```

```
      REAL C1, C2, X, XX, ZERO
      DATA ZERO / 0.0 /
C
      DO 30 L = 1, NM
         IF (IFLAG .EQ. 1) THEN
            N1 = NR
            N2 = NC
            CALL EXPTAB(TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2, P, NR,
     *                  NC, NM)
         ELSE
            N1 = NC
            N2 = NR
            CALL EXPTAB(TAB, FIT, FF, LAMBDA, LAM2, LAM1, M2, M1, P, NR,
     *                  NC, NM)
         END IF
C
         DO 20 I1 = 1, N1
            X = ZERO
            XX = ZERO
            DO 10 I2 = 1, N2
               IF (IFLAG .EQ. 1) THEN
                  C1 = TAB(I1, I2) - FIT(I1, I2)
                  C2 = FIT(I1, I2)
               ELSE
                  C1 = TAB(I2, I1) - FIT(I2, I1)
                  C2 = FIT(I2, I1)
               END IF
               X = X + M2(I2, L) * C1
               XX = XX + M2(I2, L) * M2(I2, L) * C2
   10       CONTINUE
            M1(I1, L) = M1(I1, L) + X / (XX * P(L))
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
      SUBROUTINE PHIFIT(TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2, P, NR,
     *                  NC, NM)
C
C        ALGORITHM AS 253.7  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Update the estimates of the PHI's
C
      INTEGER FF(NR, NC), NR, NC, NM
      REAL TAB(NR, NC), FIT(NR, NC), LAMBDA, LAM1(NR), LAM2(NC),
     *     M1(NR, * ), M2(NC, * ), P( * )
      INTEGER I, J, L
      REAL X, XX, ZERO
      DATA ZERO / 0.0 /
C
      DO 30 L = 1, NM
         CALL EXPTAB(TAB, FIT, FF, LAMBDA, LAM1, LAM2, M1, M2, P, NR,
     *               NC, NM)
         X = ZERO
         XX = ZERO
         DO 20 J = 1, NC
            DO 10 I = 1, NR
               X = X + M1(I, L) * M2(J, L) * (TAB(I, J) - FIT(I, J))
               XX = XX + M1(I, L) * M1(I, L) * M2(J, L) * M2(J, L) *
     *              FIT(I, J)
   10       CONTINUE
   20    CONTINUE
```

```
            P(L) = P(L) + X / XX
   30 CONTINUE
C
      RETURN
      END
      SUBROUTINE CENTRE(LAMBDA, LR, LC, MU, NU, P, RWT, CWT, NR, NC, NM)
C
C        ALGORITHM AS 253.8  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Centre (with respect to the row/column weights) the MUs/NU's
C        about zero
C
      INTEGER NR, NC, NM
      REAL LAMBDA, LR(NR), LC(NC), MU(NR, * ), NU(NC, * ), P( * ),
     *     RWT(NR), CWT(NC)
      INTEGER I, J, L
      REAL C1, C2, X, XX, ZERO
      DATA ZERO / 0.0 /
C
      C1 = ZERO
      C2 = ZERO
C
      DO 10 I = 1, NR
         C1 = C1 + RWT(I)
   10 CONTINUE
      DO 20 J = 1, NC
         C2 = C2 + CWT(J)
   20 CONTINUE
C
      DO 70 L = 1, NM
         X = ZERO
         XX = ZERO
         DO 30 I = 1, NR
            X = X + MU(I, L) * RWT(I)
   30    CONTINUE
         DO 40 J = 1, NC
            XX = XX + NU(J, L) * CWT(J)
   40    CONTINUE
         DO 50 I = 1, NR
            LR(I) = LR(I) + (XX / C2 * P(L) * MU(I, L))
            MU(I, L) = MU(I, L) - X / C1
   50    CONTINUE
         DO 60 J = 1, NC
            LC(J) = LC(J) + (X / C1 * P(L) * NU(J, L))
            NU(J, L) = NU(J, L) - XX / C2
   60    CONTINUE
         LAMBDA = LAMBDA - (P(L) * X / C1 * XX / C2)
   70 CONTINUE
      RETURN
      END
      SUBROUTINE SCALE(M, N, P, RWT, CWT, A, W, V, WORK, E, NR, NC, NM,
     *                 MNRC, JOB, IFAULT)
C
C        ALGORITHM AS 253.9  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Scale the MU's/NU's so that they satisfy the
C        identifying restrictions
C
      INTEGER NR, NC, NM, MNRC, JOB, IFAULT
      REAL M(NR, * ), N(NC, * ), P( * ), RWT(NR), CWT(NC), A(MNRC, NC),
     *     W( * ), V(NC, NC), WORK(NR), E(NC)
```

```
      INTEGER I, IA, J, L
      REAL ZERO
      DATA ZERO / 0.0 /
C
      DO 30 J = 1, NC
         DO 20 I = 1, NR
            A(I, J) = ZERO
            DO 10 L = 1, NM
               A(I, J) = A(I, J) + P(L) * M(I, L) * N(J, L)
10          CONTINUE
            A(I, J) = A(I, J) * SQRT(RWT(I)) * SQRT(CWT(J))
20       CONTINUE
30    CONTINUE
C
      IF (MNRC .GT. NR) THEN
         IA = NR + 1
         DO 50 J = 1, NC
            DO 40 I = IA, MNRC
               A(I, J) = ZERO
40          CONTINUE
50       CONTINUE
      END IF
C
C        Call LINPACK SVD subroutine
C
      CALL SSVDC(A, MNRC, NR, NC, W, E, A, MNRC, V, NC, WORK, JOB,
     *           IFAULT)
      IF (IFAULT .NE. 0) RETURN
C
      DO 80 L = 1, NM
         P(L) = W(L)
         DO 60 I = 1, NR
            M(I, L) = A(I, L) / SQRT(RWT(I))
60       CONTINUE
C
         DO 70 J = 1, NC
            N(J, L) = V(J, L) / SQRT(CWT(J))
70       CONTINUE
80    CONTINUE
C
      RETURN
      END
      SUBROUTINE CONV(TAB, FIT, FF, WRK, LAMBDA, LR, LC, MU, NU, PHI,
     *                IPT, TOT, TOL, SALE, NR, NC, NM, NW, ICON)
C
C        ALGORITHM AS 253.10  APPL. STATIST. (1990) VOL.39, NO. 1
C
C        Check for convergence of the algorithm to a solution
C                of the likelihood equations
C
      INTEGER FF(NR, NC), IPT(6), NR, NC, NM, NW, ICON
      REAL TAB(NR, NC), FIT(NR, NC), WRK(NW), LAMBDA, LR(NR), LC(NC),
     *     MU(NR, * ), NU(NC, * ), PHI( * ), TOT, TOL, SALE
      INTEGER I, J, L
      REAL ETOT, X, XX, ZERO
      DATA ZERO / 0.0 /
C
      CALL EXPTAB(TAB, FIT, FF, LAMBDA, LR, LC, MU, NU, PHI, NR, NC, NM)
C
      CALL MARGIN(FIT, WRK(IPT(2)), WRK(IPT(3)), ETOT, NR, NC)
C
```

```
      X = ZERO
C
      DO 10 I = 1, NR
         X = X + ABS(WRK(I) - WRK(IPT(2) + I - 1))
   10 CONTINUE
C
      DO 20 J = 1, NC
         X = X + ABS(WRK(IPT(1) + J - 1) - WRK(IPT(3) + J - 1))
   20 CONTINUE
C
      IF (NM .GT. 0) THEN
         DO 50 L = 1, NM
            XX = ZERO
            DO 40 J = 1, NC
               DO 30 I = 1, NR
                  XX = XX + MU(I, L) * NU(J, L) *
     *                 (TAB(I, J) - FIT(I, J))
   30          CONTINUE
   40       CONTINUE
            X = X + ABS(XX)
   50    CONTINUE
C
         DO 80 L = 1, NM
            DO 70 I = 1, NR
               XX = ZERO
               DO 60 J = 1, NC
                  XX = XX + NU(J, L) * (TAB(I, J) - FIT(I, J))
   60          CONTINUE
               X = X + ABS(XX * PHI(L))
   70       CONTINUE
   80    CONTINUE
C
         DO 110 L = 1, NM
            DO 100 J = 1, NC
               XX = ZERO
               DO 90 I = 1, NR
                  XX = XX + MU(I, L) * (TAB(I, J) - FIT(I, J))
   90          CONTINUE
               X = X + ABS(XX * PHI(L))
  100       CONTINUE
  110    CONTINUE
      END IF
C
      SALE = X
      IF (SALE .LT. TOL) THEN
         ICON = 1
      ELSE
         ICON = 0
      END IF
C
      RETURN
      END
```

```
C This file starts with a test driver program.
C It includes all auxiliary routines required by the algorithm
C
      PROGRAM TESTMGT
      DIMENSION X(19),N(19),XBAR(10),PI(10),VAR(10),PSE(10),XSE(10),
     *          VSE(10),WK(500)
      DATA N/10,21,56,79,114,122,110,85,85,
     *       61,47,49,47,44,31,20,11,4,4/
C
C        TEST DATA FOR MGT OBTAINED FROM:
C
C        EVERITT B.S. AND HAND D.J. (1981) FINITE MIXTURE DISTRIBUTIONS.
C        CHAPMAN AND HALL. P.46.
C
C      PUBLISHED PARAMETER ESTIMATES
C
C      MEANS:       19.96  ,26.16
C      VARIANCES:    4.6225, 7.6176
C      PROPORTIONS:  0.65  , 0.35
C
C
      LOUT = 6
      NG = 2
      NR = 19
      X0 = 14.5
      DO 10 I = 1,19
        FI = FLOAT(I)
        X(I) = X0+FI
   10 CONTINUE
      KW = 500
      PI(1) = .5
      PI(2) = .5
      XBAR(1) = 20.0
      XBAR(2) = 26.0
      VAR(1) = 4.0
      VAR(2) = 8.0
      TOL = 1.0E-8
      ITRUNC = 1
      ITER = 200
      CALL MGT(NG,NR,ITRUNC,X0,X,NL,NU,N,WK,KW,
     *  PI,XBAR,VAR,PSE,XSE,VSE,TOL,XLOGL,ITER,IFAULT)
      WRITE(LOUT,35)
      WRITE(LOUT,20)IFAULT
      WRITE(LOUT,20)ITER
   20 FORMAT(I10,//)
   30 FORMAT(2F14.7)
   40 FORMAT(E14.7)
   35 FORMAT(//)
      WRITE(LOUT,35)
      WRITE(LOUT,30)(XBAR(I),XSE(I),I=1,2)
      WRITE(LOUT,35)
      WRITE(LOUT,30)(VAR(I),VSE(I),I=1,2)
      WRITE(LOUT,35)
      WRITE(LOUT,30)(PI(I),PSE(I),I=1,2)
      WRITE(LOUT,35)
      WRITE(LOUT,40)XLOGL
      WRITE(LOUT,35)
      STOP
      END
C
      SUBROUTINE MGT(NG, NR, ITRUNC, X0, X, NL, NU, N, WK, KW, PI, XBAR,
```

```
     *                VAR, PSE, XSE, VSE, TOL, XLOGL, ITER, IFAULT)
C
C     ALGORITHM AS 254 APPL. STATIST. (1990), VOL. 39, NO. 2
C
C     Subroutine for fitting a mixture of normal distributions to
C     grouped and truncated data.
C
C     Auxiliary routines required: ALNORM (= AS66), CHOL (= AS6) and
C     SYMINV (= AS7)
C
      INTEGER NR, N(NR), NG, ITRUNC, NL, NU, KW, ITER, IFAULT
      REAL X(NR), WK(KW), PI(NG), XBAR(NG), VAR(NG), PSE(NG), XSE(NG),
     *     VSE(NG), X0, TOL, XLOGL
      INTEGER I1, I2, I3, IFA, IG, IG1, II1, IL, IP, IQ, IR, IR1,
     *        IT, IU, J1, J2, J3, JG, JG1, K1, K2, KG, KG1, LOC1, LOC2,
     *        LOC3, LOC4, LOC5, LOC6, MAXITR, NG1, NG3, NN, NPAR, NR1,
     *        NR2, NROW, NTOT, NULL
      REAL FNT(2), AA, AB, ARG, BA, BB, CA, CB, CONST, FN, ONE, R1, R2,
     *     RATIO, SMALL, SMALLM, SUMPI, T, TOT, TOTN, TRUNC, TWO, TXBAR,
     *     TXLOGL, UFLO, WW,XS,ZERO
      EXTERNAL ALNORM
      REAL ALNORM
      DATA ZERO, ONE, TWO / 0.0, 1.0, 2.0 /
      DATA CONST / 0.398942280401432 /
      DATA UFLO / -87.0 /
      DATA SMALL / -1.0E30 /
      DATA SMALLM / 1.0E-37 /
C
C         Check on initial parameter estimates
C
      IFAULT = 0
      IF(NG.GT.NR)IFAULT = 1
      IF(IFAULT.NE.0)RETURN
      SUMPI = ZERO
      DO 1 IG = 1,NG
        IF(PI(IG).LT.ZERO.OR.PI(IG).GT.ONE)IFAULT = 2
        IF(IFAULT.NE.0)RETURN
        IF(VAR(IG).LE.ZERO)IFAULT = 4
        IF(IFAULT.NE.0)RETURN
        SUMPI = SUMPI+PI(IG)
    1 CONTINUE
      IF(SUMPI.NE.ONE)IFAULT = 3
      IF(IFAULT.NE.0)RETURN
C
C       Set storage location pointers
C
      NG3 = 3*NG
      NPAR = NG3-1
      NPAR = 3*NG-1
      NG1 = NG-1
      NR1 = NR+1
      NR2 = NR+2
      LOC1 = NG*NR2
      LOC2 = 2*LOC1
      LOC3 = 3*LOC1
      LOC4 = LOC3+3*NG
      LOC5 = LOC4+NR2
      LOC6 = LOC5+NPAR+1
      IF(ITRUNC.EQ.1)GOTO 20
      FNT(1) = NL
      FNT(2) = NU
```

```
   20 CONTINUE
      NTOT = 0
      DO 30 IR = 1,NR
   30 NTOT = NTOT+N(IR)
      FN = NTOT
C
C        Begin iteration loop
C
      XLOGL = SMALL
      MAXITR = ITER
      ITER = 1
C
C        Calculate the A, B and C's
C
   40 DO 60 IG = 1,NG
        I1 = LOC3+IG
        I2 = I1+NG
        I3 = I2+NG
        WK(I1) = ZERO
        WK(I2) = ZERO
        WK(I3) = ZERO
        AA = ZERO
        BA = ZERO
        CA = ZERO
        DO 50 IR = 1,NR1
          IF(IR.EQ.1)THEN
            XS = X0
          ELSE
            IR1 = IR-1
            XS = X(IR1)
          ENDIF
          WW = SQRT(VAR(IG))
          ARG = (XS-XBAR(IG))/WW
          IF(ARG.LE.ZERO)THEN
            AB = ALNORM(ARG,.FALSE.)
          ELSE
            AB = ONE-ALNORM(ARG,.TRUE.)
          ENDIF
          ARG = -ARG*ARG/TWO
          IF(ARG.GT.UFLO)THEN
            ARG = EXP(ARG)
          ELSE
            ARG = ZERO
          ENDIF
          BB = ARG*CONST/WW
          CB = XS*BB
          J1 = (IG-1)*NR2+IR
          J2 = LOC1+J1
          J3 = LOC2+J1
          WK(J1) = AB-AA
          WK(J2) = BB-BA
          WK(J3) = CB-CA
          AA = AB
          BA = BB
          CA = CB
   50   CONTINUE
        J1 = (IG-1)*NR2+NR2
        J2 = LOC1+J1
        J3 = LOC2+J1
        WK(J1) = ONE-AA
        WK(J2) = -BA
```

```
          WK(J3) = -CA
       60 CONTINUE
C
C           Calculate the P's
C
          TRUNC = ZERO
          DO 80 IR = 1,NR2
            K1 = LOC4+IR
            WK(K1) = ZERO
            DO 70 IG = 1,NG
              J1 = (IG-1)*NR2+IR
              WK(K1) = WK(K1)+PI(IG)*WK(J1)
       70     CONTINUE
            IF(WK(K1).LE.ZERO)IFAULT = 6
            IF(IFAULT.NE.0)RETURN
            TRUNC = TRUNC+WK(K1)
       80 CONTINUE
C
C          Calculate the sums of the A, B and C's
C
          K1 = LOC4+1
          K2 = LOC4+NR2
          TRUNC = TRUNC-WK(K1)-WK(K2)
          TOT = ZERO
          TXLOGL = ZERO
          DO 110 IR = 1,NR2
            K1 = LOC4+IR
            IF(IR.EQ.1.OR.IR.EQ.NR2)THEN
              K2 = IR/NR2+1
              IF(ITRUNC.EQ.0)THEN
                  T = FNT(K2)
              ELSE
                T = FN*WK(K1)/TRUNC
                FNT(K2) = T
              ENDIF
            ELSE
              IR1 = IR-1
              T = FLOAT(N(IR1))
            ENDIF
            IF(ITRUNC.EQ.1.AND.(IR.EQ.1.OR.IR.EQ.NR2))GOTO 90
            TXLOGL = TXLOGL+T*LOG(WK(K1))
       90   CONTINUE
            TOT = TOT+T
            T = T/WK(K1)
            DO 100 IG = 1,NG
              I1 = LOC3+IG
              I2 = I1+NG
              I3 = I2+NG
              J1 = (IG-1)*NR2+IR
              J2 = LOC1+J1
              J3 = LOC2+J1
              WK(I1) = WK(I1)+T*WK(J1)
              WK(I2) = WK(I2)+T*WK(J2)
              WK(I3) = WK(I3)+T*WK(J3)
      100   CONTINUE
      110 CONTINUE
          IF(ITRUNC.NE.0) TXLOGL = TXLOGL-FN*LOG(TRUNC)
C
C          Calculate the new estimates
C
          DO 120 IG = 1,NG
```

```
          I1 = LOC3+IG
          I2 = I1+NG
          I3 = I2+NG
          WW = WK(I1)
          R1 = WK(I2)/WW
          R2 = WK(I3)/WW
          ARG = VAR(IG)
          TXBAR = XBAR(IG)-R1*ARG
          VAR(IG) = ARG*(ONE-R2+
     *          (TWO*TXBAR-XBAR(IG))*R1)+
     *          (TXBAR-XBAR(IG))**2
          XBAR(IG) = TXBAR
          PI(IG) = PI(IG)*WW/TOT
  120 CONTINUE
C
C        If convergence criterion is not satisfied,
C        perform another iteration
C
      ARG = ABS((TXLOGL-XLOGL)/XLOGL)
      XLOGL = TXLOGL
      IF(ARG.LE.TOL)GOTO 130
      ITER = ITER+1
      IF(ITER.LE.MAXITR)GOTO 40
      IFAULT = 5
      RETURN
C
C        Calculate the standard errors of the estimates
C
  130 IT = 0
      DO 150 IP = 1,NG3
        I1 = LOC5+IP
        WK(I1) = ZERO
        DO 140 IQ = 1,IP
          IT = IT+1
          I2 = LOC6+IT
          WK(I2) = ZERO
  140   CONTINUE
  150 CONTINUE
      IF(ITRUNC.EQ.0)THEN
        IL = 1
        IU = NR2
        TOTN = FN+FNT(1)+FNT(2)
      ELSE
        IL = 2
        IU = NR1
        TOTN = FN
      ENDIF
C
      DO 190 IR = IL,IU
        IR1 = IR-1
        IF(IR.EQ.1)NN = NL
        IF(IR.EQ.NR2)NN = NU
        IF(IR.NE.1.AND.IR.NE.NR2)NN = N(IR1)
        K1 = LOC4+IR
        K2 = NG1*NR2+IR
        WW = WK(K2)
C
        DO 160 IG = 1,NG
          J1 = (IG-1)*NR2+IR
          J2 = LOC1+J1
          J3 = LOC2+J1
```

```
          RATIO = PI(IG)/WK(K1)
          WK(J1) = (WK(J1)-WW)/WK(K1)
          WK(J2) = -RATIO*WK(J2)
          WK(J3) = -(WK(J2)*XBAR(IG)+RATIO*WK(J3))/(TWO*VAR(IG))
  160   CONTINUE
C
        IT = 0
        DO 180 IG = 1,NG3
          IG1 = IG-1
          I1 = IG1*NR2+IR
          II1 = LOC5+IG
          WK(II1) = WK(II1)+NN*WK(I1)
          DO 170 JG = 1,IG
            IT = IT+1
            I2 = (JG-1)*NR2+IR
            I3 = LOC6+IT
            IF(LOG(ABS(WK(I1))+SMALLM)+
     *         LOG(ABS(WK(I2))+SMALLM).LT.
     *         LOG(SMALLM))GOTO 170
            WK(I3) = WK(I3)+NN*WK(I1)*WK(I2)
  170     CONTINUE
  180   CONTINUE
  190 CONTINUE
C
      IF(ITRUNC.EQ.0)GOTO 220
C
      NL = FNT(1)
      NU = FNT(2)
      IT = 0
      DO 210 IG = 1,NG3
        I1 = LOC5+IG
        DO 200 JG = 1,IG
          IT = IT+1
          I2 = LOC5+JG
          I3 = LOC6+IT
          WK(I3) = WK(I3)-WK(I1)*WK(I2)/TOTN
  200   CONTINUE
  210 CONTINUE
C
  220 IT = 0
      IP = 0
      DO 240 IG = 1,NG3
        DO 230 JG = 1,IG
          IT = IT+1
          IF(IG.EQ.NG.OR.JG.EQ.NG)GO TO 230
          IP = IP+1
          I1 = LOC6+IT
          WK(IP) = WK(I1)
  230   CONTINUE
  240 CONTINUE
C
      NROW = NPAR
      NN = NPAR*(NPAR+1)/2
      CALL SYMINV(WK,NROW,NN,WK,WK(LOC3),NULL,IFA)
      IF(NG.EQ.1)THEN
        PSE(1) = ZERO
      ELSE
        IT = 0
        PSE(NG) = ZERO
        DO 260 IG = 1,NG1
          KG = IG*(IG+1)/2
```

```
          PSE(IG) = WK(KG)
          PSE(NG) = PSE(NG)-PSE(IG)
          PSE(IG) = SQRT(PSE(IG))
          DO 250 JG = 1,IG
            IT = IT+1
            PSE(NG) = PSE(NG)+TWO*WK(IT)
  250     CONTINUE
  260   CONTINUE
        PSE(NG) = SQRT(PSE(NG))
      ENDIF
C
      DO 270 IG = 1,NG
        JG = IG+NG-1
        JG1 = (JG+1)*JG/2
        KG = JG+NG
        KG1 = (KG+1)*KG/2
        XSE(IG) = SQRT(WK(JG1))
        VSE(IG) = SQRT(WK(KG1))
  270 CONTINUE
      RETURN
      END
C
      REAL FUNCTION ALNORM(X,UPPER)
C
C         ALGORITHM AS66 APPLIED STATISTICS (1973) VOL22 NO.3
C
C       EVALUATES THE TAIL AREA OF THE STANDARDISED NORMAL CURVE
C       FROM X TO INFINITY IF UPPER IS .TRUE. OR
C       FROM MINUS INFINITY TO X IF UPPER IS .FALSE.
C
      REAL LTONE, UTZERO, ZERO, HALF, ONE, CON,
     $   A1, A2, A3, A4, A5, A6, A7, B1, B2, B3, B4, B5, B6,
     $   B7, B8, B9, B10, B11, B12, X, Y, Z, ZEXP
      LOGICAL UPPER,UP
C
C         LTONE AND UTZERO MUST BE SET TO SUIT THE PARTICULAR COMPUTER
C         (SEE INTRODUCTORY TEXT)
C
      DATA LTONE, UTZERO /7.0, 12.5/
      DATA ZERO, HALF, ONE, CON /0.0, 0.5, 1.0, 1.28/
      DATA           A1,               A2,                A3,
     $               A4,               A5,                A6,
     $               A7
     $   /0.398942280444, 0.399903438504, 5.75885480458,
     $     29.8213557808,  2.62433121679, 48.6959930692,
     $     5.92885724438/
      DATA           B1,               B2,                B3,
     $               B4,               B5,                B6,
     $               B7,               B8,                B9,
     $              B10,              B11,               B12
     $   /0.398942280385,       3.8052E-8,  1.00000615302,
     $       3.98064794E-4, 1.98615381364, 0.151679116635,
     $     5.29330324926,  4.8385912808,  15.1508972451,
     $    0.742380924027,  30.789933034,  3.99019417011/
C
      ZEXP(Z) = EXP(Z)
C
      UP=UPPER
      Z=X
      IF(Z.GE.ZERO)GOTO 10
      UP=.NOT.UP
```

```
      Z=-Z
   10 IF(Z.LE.LTONE.OR.UP.AND.Z.LE.UTZERO)GOTO 20
      ALNORM=ZERO
      GOTO 40
   20 Y=HALF*Z*Z
      IF(Z.GT.CON) GOTO 30
C
      ALNORM=HALF-Z*(A1-A2*Y/(Y+A3-A4/(Y+A5+A6/(Y+A7))))
      GOTO 40
   30 ALNORM=B1*ZEXP(-Y)/(Z-B2+B3/(Z+B4+B5/(Z-B6+B7/(Z+B8-B9/
     $   (Z+B10+B11/(Z+B12))))))
   40 IF(.NOT.UP)ALNORM=ONE-ALNORM
      RETURN
      END
C
        SUBROUTINE SYMINV(A, N, NN, C, W, NULLTY, IFAULT)
C
C       ALGORITHM AS7, APPLIED STATISTICS, VOL.17, 1968, P.198.
C
C       FORMS IN C( ) AS LOWER TRIANGLE, A GENERALISED INVERSE
C       OF THE POSITIVE SEMI-DEFINITE SYMMETRIC MATRIX A( )
C       ORDER N, STORED AS LOWER TRIANGLE.
C
        REAL A(NN), C(NN), W(N), X, ZERO, ONE
C
        DATA ZERO, ONE /0.0, 1.0/
C
C       CHOLESKY FACTORIZATION OF A, RESULT IN C
C
        CALL CHOL(A, N, NN, C, NULLTY, IFAULT)
        IF(IFAULT.NE.0) RETURN
C
C       INVERT C & FORM THE PRODUCT (CINV)'*CINV, WHERE CINV IS THE INVERSE
C       OF C, ROW BY ROW STARTING WITH THE LAST ROW.
C       IROW = THE ROW NUMBER, NDIAG = LOCATION OF LAST ELEMENT IN THE ROW.
C
        IROW=N
        NDIAG=NN
   10   L=NDIAG
        IF (C(NDIAG) .EQ. ZERO) GOTO 60
        DO 20 I=IROW,N
        W(I)=C(L)
        L=L+I
   20   CONTINUE
        ICOL=N
        JCOL=NN
        MDIAG=NN
   30   L=JCOL
        X=ZERO
        IF(ICOL.EQ.IROW) X=ONE/W(IROW)
        K=N
   40   IF(K.EQ.IROW) GO TO 50
        X=X-W(K)*C(L)
        K=K-1
        L=L-1
        IF(L.GT.MDIAG) L=L-K+1
        GO TO 40
   50   C(L)=X/W(IROW)
        IF(ICOL.EQ.IROW) GO TO 80
        MDIAG=MDIAG-ICOL
        ICOL=ICOL-1
```

```
           JCOL=JCOL-1
           GO TO 30
      60   DO 70 J=IROW,N
           C(L)=ZERO
           L=L+J
      70   CONTINUE
      80   NDIAG=NDIAG-IROW
           IROW=IROW-1
           IF(IROW.NE.0) GO TO 10
           RETURN
           END
C
           SUBROUTINE CHOL(A, N, NN, U, NULLTY, IFAULT)
C
C          ALGORITHM AS6, APPLIED STATISTICS, VOL.17, (1968)
C
C          GIVEN A SYMMETRIC MATRIX ORDER N AS LOWER TRIANGLE IN A( )
C          CALCULATES AN UPPER TRIANGLE, U( ), SUCH THAT UPRIME * U = A.
C          A MUST BE POSITIVE SEMI-DEFINITE.  ETA IS SET TO MULTIPLYING
C          FACTOR DETERMINING EFFECTIVE ZERO FOR PIVOT.
C
           REAL A(NN), U(NN), ETA, ETA2, X, W, ZERO, ZABS, ZSQRT
C
C          THE VALUE OF ETA WILL DEPEND ON THE WORD-LENGTH OF THE
C          COMPUTER BEING USED.  SEE INTRODUCTORY TEXT.
C
           DATA ETA, ZERO /1.E-5, 0.0/
C
           ZABS(X) = ABS(X)
           ZSQRT(X) = SQRT(X)
C
           IFAULT=1
           IF(N.LE.0) GO TO 100
           IFAULT=3
           IF (NN .NE. N*(N+1)/2) RETURN
           IFAULT=2
           NULLTY=0
           J=1
           K=0
           ETA2 = ETA*ETA
           II=0
C
C          FACTORIZE COLUMN BY COLUMN, ICOL = COLUMN NO.
C
           DO 80 ICOL=1,N
           II=II+ICOL
           X=ETA2*A(II)
           L=0
           KK=0
C
C          IROW = ROW NUMBER WITHIN COLUMN ICOL
C
           DO 40 IROW=1,ICOL
           KK=KK+IROW
           K=K+1
           W=A(K)
           M=J
           DO 10 I=1,IROW
           L=L+1
           IF(I.EQ.IROW) GO TO 20
           W=W-U(L)*U(M)
```

```
      M=M+1
   10 CONTINUE
   20 IF(IROW.EQ.ICOL) GO TO 50
      IF(U(L).EQ.ZERO) GO TO 30
      U(K)=W/U(L)
      GO TO 40
   30 IF (W*W .GT. ZABS(X*A(KK))) RETURN
      U(K)=ZERO
   40 CONTINUE
   50 IF (ZABS(W) .LE. ZABS(ETA*A(K))) GOTO 60
      IF(W.LT.ZERO) RETURN
      U(K)=ZSQRT(W)
      GO TO 70
   60 U(K)=ZERO
      NULLTY=NULLTY+1
   70 J=J+ICOL
   80 CONTINUE
      IFAULT=0
  100 RETURN
      END
```

```
'procedure' twoway(nrows, ncols, include, resval, baseval, rowval,
    colval, rxcval, init, rcfit, xfit, test, nsteps, evalue, ifault);
'comment' Algorithm AS 255.1  Appl. Statist. (1990) Vol.39, No.2;
'value' nrows, ncols, init, rcfit, xfit, evalue;
'integer' nrows, ncols, nsteps, ifault; 'Boolean' init, rcfit, xfit;
'real' baseval, rxcval, evalue; 'Boolean' 'array' include;
'real' 'array' resval, rowval, colval; 'Boolean' 'procedure' test;
'comment' Fits a two-dimensional array of numerical values to an
    additive model incorporating a row term, a column term, and
    an optional nonadditivity term;
    'begin' 'integer' row, col, s0;
    'real' sx, sy, sx2, sy2, sxy, rxctemp, xm, ym, newres, ssval, changeval;

    'procedure' faultexit(i);
    'value' i; 'integer' i;
    'comment' Local procedure to exit on fault;
        'begin'
        ifault := i; 'goto' exit
        'end' faultexit;

    'procedure' initcalc;
    'comment' Local procedure to initialise the twoway model;
        'begin' 'integer' firstrow, lastrow, firstcol, lastcol, localrow, ndata;
        ndata := 0;
        'for' row := 1 'step' 1 'until' nrows 'do'
            'begin'
            firstcol := 0;
            'for' col := 1 'step' 1 'until' ncols 'do'
            'if' include[row, col] 'then'
                'begin'
                ndata := ndata + 1;
                'if' firstcol = 0 'then' firstcol := col 'else' lastcol := col
                'end';
            'if' firstcol = 0 'then' faultexit(4) 'else'
            'if' firstcol = lastcol 'then'
                'begin'
                firstrow := 0;
                'for' localrow := 1 'step' 1 'until' nrows 'do'
                'if' include[localrow, col] 'then'
                    'begin'
                    'if' firstrow = 0 'then' firstrow := localrow
                        'else' lastrow := localrow
                    'end';
                'if' firstrow = 0 'then' faultexit(4) 'else'
                'if' firstrow = lastrow 'then' faultexit(5)
                'end'
            'end' row;
        'if' ndata < nrows + ncols + ('if' xfit 'then' 1 'else' 0)
            'then' faultexit(3);
        'for' col := 1 'step' 1 'until' ncols 'do'
            'begin'
            firstrow := 0;
            'for' row := 1 'step' 1 'until' nrows 'do'
            'if' include[row, col] 'and' firstrow = 0 'then' firstrow := row;
            'if' firstrow = 0 'then' faultexit(4)
            'end' col;
        nsteps := 0; baseval := rxcval := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do' rowval[row] := 0.0;
        'for' col := 1 'step' 1 'until' ncols 'do' colval[col] := 0.0;
        s0 := 0; sy := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do'
```

```
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
            'begin'
            s0 := s0 + 1; sy := sy + resval[row, col]
            'end';
        changeval := sy / s0; baseval := baseval + changeval;
        ssval := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
            'begin'
            newres := resval[row, col] := resval[row, col] - changeval;
            ssval := ssval + newres * newres
            'end';
        ifault := 0;
        test('true', nsteps, ssval, evalue, ifault);
        'if' ifault > 5 'then' faultexit(ifault)
        'end' initcalc;

    'procedure' addstep(i1, i2, ni1, ni2, i1levels);
    'value' ni1, ni2; 'integer' i1, i2, ni1, ni2; 'real' 'array' i1levels;
    'comment' Local procedure to perform a row or column step. Actual
        substitutions for i1, i2 are row, col or col, row. Note that
        parameters i1, i2 must be called by name;
        'begin'
        ssval := 0.0;
        'for' i1 := 1 'step' 1 'until' ni1 'do'
            'begin'
            s0 := 0; sy := 0.0;
            'for' i2 := 1 'step' 1 'until' ni2 'do'
            'if' include[row, col] 'then'
                'begin'
                s0 := s0 + 1; sy := sy + resval[row, col]
                'end' i2;
            'if' s0 > 0 'then'
                'begin'
                changeval := sy / s0;
                i1levels[i1] := i1levels[i1] + changeval;
                'for' i2 := 1 'step' 1 'until' ni2 'do'
                'if' include[row, col] 'then'
                    'begin'
                    newres := resval[row, col] := resval[row, col] - changeval;
                    ssval := ssval + newres * newres
                    'end' i2
                'end'
            'end' i1;
        nsteps := nsteps + 1;
        'if' test('false', nsteps, ssval, evalue, ifault) 'then' 'goto' rcdone;
        'if' ifault > 5 'then' 'goto' exit
        'end' addstep;

    'procedure' rxccalc;
    'comment' Local procedure to perform nonadditivity calculation;
        'begin'
        s0 := 0; sx := sy := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
            'begin'
            s0 := s0 + 1;
            sx := sx + rowval[row] * colval[col];
```

```
            sy := sy + resval[row, col]
            'end';
        xm := sx / s0; ym := sy / s0;
        sx2 := sxy := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
            'begin'
            rxctemp := rowval[row] * colval[col] - xm;
            sx2 := sx2 + rxctemp * rxctemp;
            sxy := sxy + rxctemp * (resval[row, col] - ym)
            'end';
        changeval := ym - xm * sxy / sx2
        'end' rxccalc;

    'comment' Start of action of procedure twoway;
    'if' nrows < 1 'or' ncols < 1 'then' faultexit(1);
    'if' init 'and' xfit 'and' 'not' rcfit 'then' faultexit(2);
    'if' init 'then' initcalc 'else'
    'if' rcfit 'or' xfit 'then'
        'begin'
        'comment' Undo effect of any existing nonadditivity adjustment;
        s0 := 0; sx := sy := 0.0;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
        resval[row, col] := resval[row, col] +
            rxcval * rowval[row] * colval[col];
        rxccalc;
        baseval := baseval - changeval;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
        resval[row, col] := resval[row, col] + changeval
        'end';
    'if' rcfit 'then'
        'begin'
        'comment' Fit for rows and columns alternately until fit
            criterion is satisfied;
rcloop:
        addstep(row, col, nrows, ncols, rowval);
        addstep(col, row, ncols, nrows, colval);
        'goto' rcloop
        'end' row + col fit;
rc done:
    'if' xfit 'then'
        'begin'
        'comment' Apply nonadditive step;
        rxccalc;
        rxcval := sxy / sx2; ssval := 0.0;
        baseval := baseval + changeval;
        'for' row := 1 'step' 1 'until' nrows 'do'
        'for' col := 1 'step' 1 'until' ncols 'do'
        'if' include[row, col] 'then'
            'begin'
            resval[row, col] := newres := resval[row, col] - changeval
                 - rxcval * rowval[row] * colval[col];
            ssval := ssval + newres * newres
            'end';
        nsteps := nsteps + 1;
        test('false', nsteps, ssval, evalue, ifault)
```

```
        'end' row * col fit;
exit:
    'end' twoway;

'procedure' ddplot(nrows, ncols, include, resval, baseval, rowval,
    colval, rxcval, rowxfit, colxfit, minx, maxx, miny, maxy, plotval);
'comment' Algorithm AS 255.2  Appl. Statist. (1990) Vol.39, No.2;
'value' nrows, ncols, baseval, rxcval, rowxfit, colxfit;
'integer' nrows, ncols; 'real' baseval, rxcval, minx, maxx, miny, maxy;
'Boolean' rowxfit, colxfit; 'Boolean' 'array' include;
'real' 'array' resval, rowval, colval, plotval;
    'comment' To produce a diagonal plot of the results of a
        two-way table analysis;
    'begin' 'integer' row, col; 'real' temp, rc;
    minx := maxx := 0.0; miny := maxy := baseval;
    'for' row := 1 'step' 1 'until' nrows 'do'
    'for' col := 1 'step' 1 'until' ncols 'do'
        'begin'
        rc := rxcval * rowval[row] * colval[col];
        temp := colval[col] - rowval[row];
        'if' rowxfit 'then'
            'begin'
            'if' 'not' colxfit 'then' temp := temp - rc
            'end'
        'else' 'if' colxfit 'then' temp := temp + rc;
        'if' temp < minx 'then' minx := temp
             'else' 'if' temp > maxx 'then' maxx := temp;
        plotval[row, col, 1] := temp;
        temp := baseval + rowval[row] + colval[col];
        'if' rowxfit 'or' colxfit 'then' temp := temp + rc;
        'if' temp < miny 'then' miny := temp
             'else' 'if' temp > maxy 'then' maxy := temp;
        plotval[row, col, 2] := temp;
        'if' include[row, col] 'then' temp := temp + resval[row, col];
        'if' temp < miny 'then' miny := temp
             'else' 'if' temp > maxy 'then' maxy := temp;
        plotval[row, col, 3] := temp
        'end'
    'end' ddplot;

'Boolean' 'procedure' testex(init, nsteps, ssval, evalue, ifault);
'comment' Algorithm AS 255.3  Appl. Statist. (1990) Vol.39, No.2;
'value' init, nsteps, sscal, evalue;
'integer' nsteps, ifault; 'Boolean' init; 'real' ssval, evalue;
    'comment' This particular implementation of the test procedure relies
        upon the ratio of the largest to the smallest of the 3 most
        recent values of ssvals[ ]. The stack is implemented as an own
        array variable. No use is made of the nsteps parameter;
'if' ifault > 0 'then' testex := 'false' 'else'
    'begin' 'integer' i; 'own' 'real' 'array' ssvals[1:3];
    'if' init 'then'
        'begin'
        'for' i := 1, 2, 3 'do' ssvals[i] := -1; testex := 'false'
        'end'
     'else' 'if' ssval < 0.0 'then'
        'begin'
        ifault := 6; testex := 'false'
        'end'
     'else'
        'begin' 'real' ssvalue, ssmax, ssmin;
        'for' i := 3, 2 'do' ssvals[i] := ssvals[i - 1];
```

```
            ssvals[1] := ssval;
            ssmax := -1; ssmin := ssval;
            'for' i := 1, 2, 3 'do'
                'begin'
                ssvalue := ssvals[i];
                'if' ssvalue < 0.0 'then'
                    'begin'
                    testex := 'false' ; 'goto' exit
                    'end'
                'else' 'if' ssvalue < ssmin 'then' ssmin := ssvalue
                'else' 'if' ssvalue > ssmax 'then' ssmax := ssvalue
                'end' i;
            testex := ssmax < ssmin * (1.0 + abs(evalue))
            'end';
    exit:
        'end' testex;
```

```
FUNCTION imhofbd(lambda:realarray; mult:posintarray; delta:realarray;
         noterms:units; eps1,eps2:real; VAR ifault:nonnegint):real;

{Algorithm AS 256.1  Appl. Statist. (1990) Vol.39, No.2}

{Pascal translation of Koerts and Abrahamse's procedure
 for evaluating Imhof's upper bound}

CONST pi=3.14159265;
VAR   i:units;
      count,hold,range,sum1,sum2:real;

BEGIN
IF (noterms < 1) OR (eps1 <= 0.0) OR (eps2 <= 0.0) THEN
   BEGIN
   ifault := 2; imhofbd := -2.0
   END
ELSE
   BEGIN
   count := 0.0;
   sum1 := 0.0; sum2 := 0.0;
   FOR i := 1 TO noterms DO
      BEGIN
      hold := abs(lambda[i]);
      IF hold > eps2 THEN
         BEGIN
         count := count+mult[i];
         sum1 := sum1+mult[i]*ln(hold);
         END;
      sum2 := sum2+delta[i]
      END;
   IF count < 0.9 THEN
      BEGIN
      ifault := 4; imhofbd := -4.0
      END
   ELSE
      BEGIN
      count := 0.5*count;
      sum1 := 0.5*sum1+ln(pi*count);
      range := exp(-(sum1+0.5*sum2+ln(eps1))/count);
      IF sum2=0.0 THEN
      range := range+5.0/count
      ELSE
         BEGIN
            REPEAT
            sum2 := 0.0;
            FOR i := 1 TO noterms DO
               BEGIN
               hold := sqr(range*lambda[i]);
               sum2 := sum2+delta[i]*hold/(1.0+hold)
               END;
            hold := exp(sum1+count*ln(range)+0.5*sum2);
            IF hold*eps1 <= 1.0 THEN range := range+5.0/count ELSE count := 0.0
            UNTIL count=0.0;
         END
      END;
   imhofbd := range
   END
END; {of imhofbd}

FUNCTION imhofint(lambda:realarray; mult:posintarray; delta:realarray;
```

```
          noterms:units; arg,bound,eps3:real; VAR ifault:nonnegint):real;

{Algorithm AS 256.2  Appl. Statist. (1990) Vol.39, No.2}

{Pascal translation of Koerts and Abrahamse's procedure
 for evaluating Imhof's integral by Simpson's Rule}

CONST maxit=14;
      pi=3.14159265;
VAR   i,j,n:nonnegint;
      eps4,hold,int1,int2,step,sum1,sum2,sum4:real;
      cgd:boolean;

FUNCTION imhoffn(u:real):real;

   {imhoffn evaluates Imhof's integrand}

   VAR i:units;
       hold2,hold3,rho,sum,theta:real;

   BEGIN
   rho := 0.0;
   sum := 0.0;
   theta := -u*arg;
   FOR i := 1 TO noterms DO
      BEGIN
      hold := u*lambda[i];
      hold2 := sqr(hold);
      hold3 := 1.0+hold2;
      theta := theta+mult[i]*arctan(hold)+delta[i]*hold/hold3;
      sum := sum+delta[i]*hold2/hold3;
      rho := rho+mult[i]*ln(hold3)
      END;
   imhoffn := sin(0.5*theta)/(u*exp(0.5*sum+0.25*rho))
   END; {of imhoffn}

BEGIN
IF (noterms < 1) OR (bound <= 0.0) OR (eps3 <= 0.0) THEN
   BEGIN
   ifault := 2; imhofint := -2.0
   END
ELSE
   BEGIN
   ifault := 5;
   n := 2; step := 0.5*bound;
   eps4 := 3.0*pi*eps3;
   sum1 := -arg;
   FOR i := 1 TO noterms DO
   sum1 := sum1+lambda[i]*(mult[i]+delta[i]);
   sum1 := 0.5*sum1+imhoffn(bound);
   sum2 := 0.0;
   sum4 := imhoffn(step);
   int2 := (sum1+4.0*sum4)*step;
   i := 0; cgd := false;
      REPEAT
      i := i+1; n := n+n;
      step := 0.5*step;
      int1 := int2;
      sum2 := sum2+sum4;
      int2 := sum1+sum2+sum2;
      sum4 := 0.0; j := 1;
```

```
            REPEAT
            sum4 := sum4+imhoffn(j*step);
            j := j+2
            UNTIL j > n;
         int2 := (int2+4.0*sum4)*step;
         IF i > 3 THEN
            BEGIN
            IF abs(int1-int2) < eps4 THEN
               BEGIN
               IF abs(int2) > 1.5*pi THEN
               ifault := 6 ELSE ifault := 0;
               cgd := true
               END
            END;
         UNTIL (i=maxit) OR cgd;
      imhofint := 0.5-int2/(3.0*pi)
      END
   END; {of imhofint}

   FUNCTION imhof(lambda:realarray; mult:posintarray; delta:realarray;
   noterms:units; arg,bound,eps1,eps2,eps3:real; VAR ifault:nonnegint):real;

   {Algorithm AS 256.3  Appl. Statist. (1990) Vol.39, No.2}

   {Pascal translation of Koerts and Abrahamse's implementation of
    Imhof's procedure for evaluating the probability that a diagonal
    form in normal variables is less than a given value, arg}

   VAR i:units;

   BEGIN

   {check parameters}
   IF (noterms < 1) OR (eps1 <= 0.0) OR (eps2 <= 0.0) OR (eps3 <= 0.0) THEN
      BEGIN
      ifault := 2; imhof := -2.0
      END
   ELSE
      BEGIN
      ifault := 1; i := 1;
      WHILE ifault=1 DO
         BEGIN
         i := i+1;
         IF i=noterms THEN ifault := 0;
         IF (mult[i] < 1) OR (delta[i] < 0.0) THEN
            BEGIN
            ifault := 3; imhof := -i
            END
         END;

      {main body}
      IF ifault=0 THEN
         BEGIN
         IF bound <= 0.0 THEN
         bound := imhofbd(lambda,mult,delta,noterms,eps1,eps2,ifault);
         IF ifault=0 THEN
         imhof := imhofint(lambda,mult,delta,noterms,arg,bound,eps3,ifault)
         ELSE imhof := -ifault
         END
      END
   END; {of imhof}
```

```
FUNCTION imhotep(diag,subd:realarray; noterms:units;
nc,arg,bound,eps0,eps1,eps2,eps3:real; VAR ifault:nonnegint):real;

{Algorithm AS 256.4  Appl. Statist. (1990) Vol.39, No.2}

{imhotep uses Imhof's procedure with real arithmetic
 to evaluate the probability that a tridiagonal form
 in normal variables is less than a given value, arg}

VAR i:units;
    hold:real;
    delta:realarray;
    mult:posintarray;

BEGIN
IF (noterms < 1) OR (nc < 0.0) OR
   (eps0 <= 0.0) OR (eps1 <= 0.0) OR (eps2 <= 0.0) OR (eps3 <= 0.0) THEN
   BEGIN
   ifault := 2; imhotep := -2.0
   END
ELSE
   BEGIN
   ifault := 0; delta[noterms] := 1.0;
   IF noterms > 1 THEN
      BEGIN
      FOR i := noterms-1 DOWNTO 1 DO delta[i] := 0.0;
      tql3(noterms,diag,subd,delta,eps0,ifault)
      END;

   IF ifault <> 0 THEN imhotep := -ifault
   ELSE
      BEGIN
      FOR i := 1 TO noterms DO
         BEGIN
         hold := delta[i]; mult[i] := 1;
         delta[i] := nc*sqr(hold)
         END;
      imhotep := imhof(diag,mult,delta,noterms,arg,bound,eps1,eps2,eps3,ifault)
      END
   END
END; {of imhotep}

FUNCTION sneekbd(diag,subd:realarray; noterms:units;
nc,eps1,eps2:real; VAR ifault:nonnegint):real;

{Algorithm AS 256.5  Appl. Statist. (1990) Vol.39, No.2}

{sneekbd evaluates Imhof's upper bound
 by means of Sturm sequences}

CONST pi=3.14159265;
VAR   i,j:nonnegint;
      hold,sum1,sum2:real;

PROCEDURE sturm(eta:real; VAR det:real; VAR mult:nonnegint);

   {sturm evaluates the determinant of a tridiasonal matrix
    and the number of roots less than a given value, eta}
```

```
      VAR i:units;
          count:nonnegint;
          real0,real1,real2:real;

      BEGIN
      count := 0;
      real1 := 1.0; real2 := 0.0;
      FOR i := 1 TO noterms DO
         BEGIN
         real0 := real1*(diag[i]-eta)-real2*sqr(subd[i]);
         real2 := real1; real1 := real0;
         IF real1*real2 < 0.0 THEN count := count+1
         END;
      det := real0;
      mult := count
      END; {of sturm}

   BEGIN
   IF (noterms < 1) OR (nc < 0.0) OR (eps1 <= 0.0) OR (eps2 <= 0.0) THEN
      BEGIN
      ifault := 2; sneekbd := -2.0
      END
   ELSE
      BEGIN
      {trap small eigenvalues}
      sturm(-eps2,hold,i);
      sturm( eps2,hold,j);
      IF i <> j THEN
         BEGIN
         ifault := 7; sneekbd := -7.0
         END
      ELSE
         BEGIN
         {compute bound}
         ifault := 0;
         sturm(0.0,hold,j);
         sum1 := 0.5*noterms;
         sum2 := sqrt(abs(hold))*pi*sum1*eps1;
         sneekbd := exp(-(ln(sum2)+0.5*nc)/sum1)+5.0/sum1
         END
      END
   END; {of sneekbd}

   FUNCTION sneekint(diag,subd:realarray; noterms:units;
   nc,arg,bound,eps3:real; VAR ifault:nonnegint):real;

   {Algorithm AS 256.6  Appl. Statist. (1990) Vol.39, No.2}

   {sneekint uses complex arithmetic to
    evaluate Imhof's integral by Simpson's Rule}

   CONST maxit=14;
         eta=0.0001;
         pi=3.14159265;
   VAR   i,j,n:nonnegint;
         eps4,int1,int2,step,sum1,sum2,sum4:real;
         cgd:boolean;

   FUNCTION sneekfn(u:real):real;

      {sneekfn evaluates Imhof's integrand}
```

```
    VAR sgn:-1..1;
        i:units;
        hold1,hold2,imag0,imag1,imag2,pies,real0,real1,real2:real;

    BEGIN

    {evaluate determinant}
    real1 := 1.0; imag1 := 0.0;
    real2 := 0.0; imag2 := 0.0;
    sgn := 1; pies := 0.0;
    FOR i := 1 TO noterms DO
       BEGIN
       hold1 := u*diag[i]; hold2 := sqr(u*subd[i]);
       real0 := real1+imag1*hold1+real2*hold2;
       imag0 := imag1-real1*hold1+imag2*hold2;
       real2 := real1; real1 := real0;
       imag2 := imag1; imag1 := imag0;
       IF sgn*real0 < 0.0 THEN
          BEGIN
          sgn := -sgn;
          IF sgn*imag0 < 0.0
              THEN pies := pies+pi
              ELSE pies := pies-pi
          END
       END;

    {evaluate exponent}
    hold1 := -nc/(sqr(real1)+sqr(imag1));
    real0 := nc+(real1*real2+imag1*imag2)*hold1;
    imag0 := (real1*imag2-imag1*real2)*hold1;

    {evaluate integrand}
    hold1 := imag1/real1;
    hold2 := sqr(hold1)+1.0;
    hold2 := sqrt(hold2)*abs(real1);
    hold2 := exp(-0.5*real0)/sqrt(hold2);
    hold1 := arctan(hold1)+pies+imag0+u*arg;
    sneekfn := sin(-0.5*hold1)*hold2/u
    END; {of sneekfn}

BEGIN
IF (noterms < 1) OR (nc < 0.0) OR (bound <= 0.0) OR (eps3 <= 0.0) THEN
   BEGIN
   ifault := 2; sneekint := -2.0
   END
ELSE
   BEGIN
   ifault := 5;
   n := 2; step := 0.5*bound;
   eps4 := 3.0*pi*eps3;
   sum1 := sneekfn(eta*bound)+sneekfn(bound);
   sum2 := 0.0;
   sum4 := sneekfn(step);
   int2 := (sum1+4.0*sum4)*step;
   i := 0; cgd := false;
      REPEAT
      i := i+1; n := n+n;
      step := 0.5*step;
      int1 := int2;
      sum2 := sum2+sum4;
```

```
        int2 := sum1+sum2+sum2;
        sum4 := 0.0; j := 1;
           REPEAT
           sum4 := sum4+sneekfn(j*step);
           j := j+2
           UNTIL j > n;
        int2 := (int2+4.0*sum4)*step;
        IF i > 3 THEN
           BEGIN
           IF abs(int1-int2) < eps4 THEN
              BEGIN
              IF abs(int2) > 1.5*pi THEN
              ifault := 6 ELSE ifault := 0;
              cgd := true
              END
           END
        UNTIL (i=maxit) OR cgd;
     sneekint := 0.5-int2/(3.0*pi)
     END
  END; {of sneekint}

  FUNCTION sneek(diag,subd:realarray; noterms:units;
  nc,arg,bound,eps1,eps2,eps3:real; VAR ifault:nonnegint):real;

  {Algorithm AS 256.7  Appl. Statist. (1990) Vol.39, No.2}

  {sneek uses Imhof's procedure with complex arithmetic
   to evaluate the probability that a tridiagonal form
   in normal variables is less than a given value, arg}

  BEGIN
  IF (noterms < 1) OR (nc < 0.0) OR
     (eps1 <= 0.0) OR (eps2 <= 0.0) OR (eps3 <= 0.0) THEN
     BEGIN
     ifault := 2; sneek := -2.0
     END
  ELSE
     BEGIN
     ifault := 0;
     IF bound <= 0.0 THEN
     bound := sneekbd(diag,subd,noterms,nc,eps1,eps2,ifault);
     IF ifault=0 THEN
     sneek := sneekint(diag,subd,noterms,nc,arg,bound,eps3,ifault)
     END
  END; {of sneek}
```

```
      SUBROUTINE ISRE(K, NH, XO, WO, X, Y, Z, WX, WY, WZ, XA, IFAULT)
C
C        ALGORITHM AS 257.1  APPL.STATIST. (1990), VOL.39, NO.3
C
C        Subroutine to compute isotonic regression for umbrella
C        ordering with known peak
C        Algorithm AMALGM (AS 149, Cran, 1980) is called
C
      INTEGER K, NH, IFAULT
      REAL XO(K), WO(K), X(K), Y(K), Z(K), WX(K), WY(K), WZ(K), XA(K)
      INTEGER I, I1, IH, IH1, IHIGH, ILOW, IXMAX, J, L, N
      REAL S, TOL, XMAX
      DATA TOL / 1.0E-6 /
C
C        Check input data K, NH and WO
C
      IFAULT = 1
      IF (K .LT. 2) RETURN
      IFAULT = 3
      IF (NH .LT. 1 .OR. NH .GT. K) RETURN
      IFAULT = 2
      DO 10 I = 1, K
         IF (WO(I) .LE. TOL) RETURN
         X(I) = XO(I)
         WX(I) = WO(I)
   10 CONTINUE
      IFAULT = 0
      N = K
      IH = NH
C
C        Pool the active block with the next lower block
C        from X(1) to X(IH-1)
C
      IF (IH .LE. 2) GO TO 40
      DO 20 I = 1, IH - 1
         Y(I) = X(I)
         WY(I) = WX(I)
   20 CONTINUE
      CALL AMALGM(IH - 1, Y, WY, Z, WZ, XA, IFAULT)
      DO 30 I = 1, IH - 1
         X(I) = XA(I)
   30 CONTINUE
C
C        Pool the active block with the next higher block
C        from X(IH+1) to X(N)
C
   40 IF (N - IH .LE. 1) GO TO 70
      L = N - IH
      DO 50 I = IH + 1, N
         Y(L) = X(I)
         WY(L) = WX(I)
         L = L - 1
   50 CONTINUE
      CALL AMALGM(N - IH, Y, WY, Z, WZ, XA, IFAULT)
      L = N - IH
      DO 60 I = IH + 1, N
         X(I) = XA(L)
         L = L - 1
   60 CONTINUE
C
C        Pool adjacent violators such that X(IH) is largest
```

```
C
   70 ILOW = MAX(1, IH - 1)
      IHIGH = MIN(N, IH + 1)
      XMAX = X(IH)
      IXMAX = IH
      DO 80 I = ILOW, IHIGH
         IF (I .EQ. IH .OR. XMAX .GE. X(I)) GO TO 80
         XMAX = X(I)
         IXMAX = I
   80 CONTINUE
      IF (IXMAX .EQ. IH) GO TO 100
      IF (IXMAX .LT. IH) IH = IH - 1
      IH1 = IH + 1
      X(IH) = (WX(IH) * X(IH) + WX(IH1) * X(IH1)) / (WX(IH) + WX(IH1))
      WX(IH) = WX(IH) + WX(IH1)
      DO 90 I = IH1, N - 1
         X(I) = X(I + 1)
         WX(I) = WX(I + 1)
   90 CONTINUE
      N = N - 1
      GO TO 70
C
C        Obtain XA(1), ..., XA(K) from X(1), ..., X(N)
C
  100 I1 = 1
      DO 130 I = 1, N
         S = 0.0
         DO 110 J = I1, K
            S = S + WO(J)
            XA(J) = X(I)
            IF (ABS(S - WX(I)) .LT. TOL) GO TO 120
  110    CONTINUE
  120    I1 = J + 1
  130 CONTINUE
      RETURN
      END
      SUBROUTINE ISREU(MAXK, K, XO, WO, X, Y, Z, WX, WY, WZ, MINVAL,
     *                 XMIN, XB, RES, IFAULT)
C
C        ALGORITHM AS 257.2  APPL.STATIST. (1990), VOL.39, NO.3
C
C        Subroutine to compute isotonic regression for umbrella
C        ordering with unknown peak
C        A subroutine ISRE given in this paper is called
C
      INTEGER MAXK, K, MINVAL, IFAULT
      REAL XO(K), WO(K), X(K), Y(K), Z(K), WX(K), WY(K), WZ(K), XMIN(K),
     *     XB(MAXK, K), RES(K)
      INTEGER I, NH
      REAL RESMIN
C
C        Check that K .GE. 2
C
      IF (K .GE. 2) GO TO 10
      IFAULT = 1
      RETURN
C
C        For all cases with the peak NH = 1, ..., K
C
   10 DO 30 NH = 1, K
         CALL ISRE(K, NH, XO, WO, X, Y, Z, WX, WY, WZ, XB(1, NH),
```

```
     *             IFAULT)
          IF (IFAULT .NE. 0) RETURN
C
C         Calculate square error to obtain min
C
          RES(NH) = 0.0
          DO 20 I = 1, K
             RES(NH) = RES(NH) + WO(I) * (XB(I, NH) - XO(I)) ** 2
20        CONTINUE
          IF (NH .NE. 1 .AND. RES(NH) .GT. RESMIN) GO TO 30
          MINVAL = NH
          RESMIN = RES(NH)
30        CONTINUE
          DO 40 I = 1, K
             XMIN(I) = XB(I, MINVAL)
40        CONTINUE
          RETURN
          END
```

```
      SUBROUTINE ARL2(DELTA, K, H, S0, ARL, ARLFIR, IFAULT)
C
C        ALGORITHM AS 258.1  APPL.STATIST. (1990), VOL.39, NO.3
C
C        Computes the average run length for a cumulative
C        sum control scheme
C
      REAL DELTA, K, H, S0, ARL, ARLFIR
      INTEGER IFAULT
      REAL ARLH, ARLHF, ARLL, ARLLF, BIGARL, BIGDEL
      INTEGER JFAULT
      DATA BIGARL / 1.E30 / , BIGDEL / 5.0 /
C
      IFAULT = 0
      IF (DELTA .LT. 0.0) THEN
         IFAULT = 1
      ELSE
C
C        Compute ARL's for upper tail.
C
         CALL ARL1(DELTA, K, H, S0, ARLH, ARLHF, IFAULT)
         IF (IFAULT .EQ. 0) THEN
C
C        If DELTA=0, then ARL's for lower tail are the same as for
C        the upper.
C
            IF (DELTA .EQ. 0.0) THEN
               ARLLF = ARLHF
               ARLL = ARLH
C
C        If DELTA is too large, skip the low-side ARL calculation.
C
            ELSE IF (DELTA .GT. BIGDEL) THEN
               ARLL = BIGARL
               ARLLF = BIGARL
            ELSE
C
C        Otherwise compute ARL's for lower tail.
C
               CALL ARL1(-DELTA, K, H, S0, ARLL, ARLLF, JFAULT)
C
C        Set lower ARL's large if negative JFAULT .GT. 0
C
               IF (ARLL .LE. ARLH .OR. ARLLF .LE. ARLHF .OR.
     *             ARLL .LT. ARLLF .OR. JFAULT .GT. 0) THEN
                  ARLL = BIGARL
                  ARLLF = BIGARL
               END IF
            END IF
C
C        Compute two-sided ARL for S0=0.0
C
            ARL = ARLH / (1.0 + ARLH / ARLL)
C
C        Compute two-sided ARL for specified value of S0.
C
            ARLFIR = ARLHF / (1.0 + ARLH / ARLL) +
     *               ARLH / (ARLH / ARLLF + ARLL / ARLLF) - ARL
C
C        Set IFAULT=3 if two-sided ARL's are lower bounds.
C
```

```
            IF (IFAULT .EQ. 0 .AND. S0 .GT. H / 2.0 + K) IFAULT = 3
         END IF
      END IF
      RETURN
      END
      SUBROUTINE ARL1(DELTA, K, H, S0, ARL, ARLFIR, IFAULT)
C
C        ALGORITHM AS 258.2  APPL.STATIST. (1990), VOL.39, NO.3
C
      REAL DELTA, K, H, S0, ARL, ARLFIR
      INTEGER IFAULT
      INTEGER N, N1, N2, I, J
      REAL XN
      DOUBLE PRECISION XCOND
      PARAMETER (N=12, N1=N + 1, N2=N + 2, XN=N, XCOND=100.D0)
      INTEGER IPVT(N1)
      REAL P1, P2, ALNORM
      DOUBLE PRECISION A(N1, N1), B(N1), R(N1), W(N2),
     *                 C, E1, E2, RCOND, S, T
      EXTERNAL ALNORM
C
C        N is the degree of the polynomial approximation.
C        XCOND defines the criterion for singularity:
C              XCOND+RCOND .LE. XCOND,
C        where RCOND is the reciprocal of the condition number.
C
      IFAULT = 0
      IF (K .LT. 0.0 .OR. H .LT. 0.0 .OR. S0 .LT. 0.0 .OR.
     *    S0 .GT. H) THEN
         IFAULT = 1
      ELSE IF (H .EQ. 0.0) THEN
         ARL = 1.0 / ALNORM(DELTA - K, .FALSE.)
         ARLFIR = ARL
      ELSE
C
C        Set C.
C
         C = MAX(0.0, K - DELTA)
C
C        For each point S at which the polynomial approximation is to be
C        evaluated...
C
         DO 40 I = 0, N
C
C        Compute S
C
            S = H * I / XN
C
C        Calculate necessary exponentials in S.
C
            E1 = EXP(C * S)
            E2 = EXP((S + DELTA - K) * C + C * C / 2.0)
C
C        Apply left-hand-side of integral equation.
C
            T = E1
            DO 10 J = 1, N + 1
               A(I + 1, J) = T
               T = T * S
   10       CONTINUE
C
```

```
C        Apply lower integration limit.
C
            CALL MOMENT(-S - DELTA - C + K, -S - DELTA - C + K, N, R, W)
            DO 20 J = 1, N + 1
               A(I + 1, J) = A(I + 1, J) - R(J) * E2
   20       CONTINUE
C
C        Apply upper integration limit.
C
            CALL MOMENT(H - S - DELTA - C + K, -S - DELTA - C + K, N, R,
     *                  W)
            DO 30 J = 1, N + 1
               A(I + 1, J) = A(I + 1, J) + R(J) * E2
   30       CONTINUE
C
C        Apply term '1 + L(0) F(-S-DELTA+K)'.
C
            A(I + 1, 1) = A(I + 1, 1) - ALNORM(REAL(-S - DELTA + K),
     *                    .FALSE.)
            B(I + 1) = 1.0
   40    CONTINUE
C
C        Normalize the simultaneous equations
C
         DO 70 I = 1, N + 1
            S = 0.0
            DO 50 J = 1, N + 1
               S = MAX(S, ABS(A(I, J)))
   50       CONTINUE
            B(I) = B(I) / S
            DO 60 J = 1, N + 1
               A(I, J) = A(I, J) / S
   60       CONTINUE
   70    CONTINUE
         DO 100 J = 1, N + 1
            W(J) = 0.0
            DO 80 I = 1, N + 1
               W(J) = MAX(W(J), ABS(A(I, J)))
   80       CONTINUE
            DO 90 I = 1, N + 1
               A(I, J) = A(I, J) / W(J)
   90       CONTINUE
  100    CONTINUE
C
C        Factor matrix A.  If equations are singular to working
C        precision, IFAULT=2.
C
C        ************************************
C        SUBROUTINE DGECO(A,LDA,N,IPVT,RCOND,Z)
C        on entry:
C        A:     the matrix to be factored.
C        LDA:   the leading dimension of array A.
C        N:     the order of the matrix A.
C        on return:
C        A:     the lu factorization of A.
C        IPVT:  pivot indices.
C        RCOND: an estimate of the reciprocal condition of A.
C        Z:     a working vector.
C        ************************************
C
         CALL DGECO(A, N + 1, N + 1, IPVT, RCOND, R)
```

```
            IF (XCOND + RCOND .EQ. XCOND) THEN
               IFAULT = 2
            ELSE
C
C           Solve for the polynomial coefficients
C
C           ***************************************
C           SUBROUTINE DGESL(A,LDA,N,IPVT,B,JOB)
C           on entry:
C           A:     the output from dgeco.
C           LDA:   the leading dimension of array A.
C           N:     the order of the matrix A.
C           IPVT:  the pivot vector from dgeco.
C           B:     the right hand side vector.
C           JOB:   = 0       to solve A*X=B.
C                  = nonzero to solve trans(A)*X=B.
C           on return:
C           B:     the solution vector X.
C           ***************************************
C
               CALL DGESL(A, N + 1, N + 1, IPVT, B, 0)
C
C           Get ARL and ARLFIR.
C
               ARL = B(1) / W(1)
               ARLFIR = 0.0
               DO 110 I = 0, N
                  ARLFIR = S0 * ARLFIR + B(N - I + 1) / W(N - I + 1)
  110          CONTINUE
               ARLFIR = ARLFIR * EXP(C * S0)
            END IF
         END IF
         RETURN
         END
         SUBROUTINE MOMENT(X, Y, N, R, W)
C
C           ALGORITHM AS 258.3  APPL.STATIST. (1990), VOL.39, NO.3
C
C           For k=0,...,n,  computes the integral from x to
C           infinity of the quantity
C
C                    R(k+1) = ( t - y )**k z(t) dt,
C               where
C                    z(t) = 1/sqrt(2 pi) exp( -t**2 / 2 ) .
C
         INTEGER N
         DOUBLE PRECISION X, Y, R( * ), W( * )
         INTEGER I, K
         REAL ALNORM
         DOUBLE PRECISION XMY, SQR2PI, FACT(19)
         EXTERNAL ALNORM
         DATA SQR2PI / 2.50662827463100050241576 5D0 /
         DATA FACT / 2 * 1.D0, 2.D0, 6.D0, 24.D0, 120.D0, 720.D0, 5040.D0,
        *     40320.D0, 362880.D0, 3628800.D0, 39916800.D0, 479001600.D0,
        *     6227020800.D0, 87178291200.D0, 1307674368000.D0,
        *     20922789888000.D0, 355687428096000.D0, 6402373705728000.D0 /
C
C           Compute first term of R.
C
         W(1) = EXP(-X * X / 2.0) / SQR2PI
         W(2) = ALNORM(REAL(-X), .FALSE.)
```

```
      R(1) = W(2)
      IF (N .GT. 0) THEN
         DO 10 I = 1, N
            W(I + 2) = (W(I) - X * W(I + 1)) / I
            R(I + 1) = W(I + 2) * FACT(I + 1)
10       CONTINUE
C
C        If X=Y, then R is already computed.
C
         IF (X .NE. Y) THEN
C
C        Compute R.
C
            DO 30 K = 0, N
               R(K + 1) = W(2) / FACT(K + 1)
               XMY = X - Y
               DO 20 I = 1, K
                  R(K + 1) = R(K + 1) * XMY + W(I + 2) / FACT(K - I + 1)
20             CONTINUE
               R(K + 1) = R(K + 1) * FACT(K + 1)
30          CONTINUE
         END IF
      END IF
      RETURN
      END
```

```
      SUBROUTINE RMCONF(EST, CONF, M, STEPK, MODSTP, LIML, SBOUND, LIMU,
     *                  BBOUND, LSTART, SLOLIM, SHILIM, ISTART, IRSTRT,
     *                  INDIC, CLIMLO, CLIMHI, CLLOSE, CLHISE, MLO, MHI,
     *                  PTHPLT, IFAULT)
C
C        ALGORITHM AS 259  APPL. STATIST. (1990) VOL. 39, NO. 3
C
C        Calculates Monte Carlo confidence intervals using the Robbins-
C        Monro process.  The intervals are asymptotically exact as the
C        number of steps tends to infinity under general conditions.
C
      CHARACTER * 1 PTHPLT(76, 105)
      INTEGER M, ISTART, INDIC, MLO, MHI, IFAULT
      LOGICAL MODSTP, LIML, LIMU, LSTART, IRSTRT
      REAL EST, CONF, STEPK, SBOUND, BBOUND, SLOLIM, SHILIM, CLIMLO,
     *     CLIMHI, CLLOSE, CLHISE
      CHARACTER *  1 DIGIT(0:9), SYMB
      CHARACTER * 16 MIDPLT
      CHARACTER * 28 HIPLOT, LOPLOT
      CHARACTER * 30 MIDLIN
      INTEGER IX(76), MULT(2)
      REAL CVHI(20), CVLO(20), SEARCH(2, 2000)
      INTEGER I, IERR, II, IIM, IPTH, IR, ISTRT, IY, J, JJ, JX, K,
     *        KMULT, KOUNT, L, LIMIT, LL, LR, LX, LY, LYY, M0, M1, M2,
     *        MID, MM, NPT
      LOGICAL IRSTAR, ISTEP
      REAL AL2, ALPHA, BB, BEGPOS, BIG, BOOTES, CONF2, D1, D2, DIV,
     *     ENDPOS, FIFTY, HALF, HIPOS, HUN, ONE, ONEPSM, P95, POSN,
     *     PTHREE, QUART, ROOT2, SB, SEPOSN, SEVFIV, SMALL, STEP,
     *     STEP2, STEPC, STEPHI, TEN, THREP5, TRT2PI, TWO, X, YMAX,
     *     YMIN, YSCALE, Z, Z1, ZA, ZERO
      REAL PPND, SIMU
      EXTERNAL PPND, SIMU
C
      DATA DIGIT / '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' /
      DATA ZERO, QUART, PTHREE, HALF / 0.0, 0.25, 0.3, 0.5 /
      DATA ONE, ROOT2, TWO, THREP5, TEN / 1.0, 1.414214, 2.0, 3.5,
     *     10.0 /
      DATA FIFTY, SEVFIV, P95, HUN / 50.0, 75.0, 95.0, 100.0 /
      DATA SMALL, ONEPSM, BIG / 1.0E-6, 1.00001, 1.0E20 /
      DATA HIPLOT / 'Upper limit, number of steps' /
      DATA LOPLOT / 'Lower limit, number of steps' /
      DATA MIDLIN / '-----------------------------' /
      DATA MIDPLT / ' Point estimate ' /
C
C        TRT2PI is 2*SQRT(2*PI); Z1 is z st P(Z > z)=0.005,  Z is N(0,1)
C
      DATA TRT2PI / 5.013257 / , Z1 / 2.576 /
C
      IFAULT = 0
      MULT(1) = 1
      MULT(2) = 1
      IF (CONF .GE. HUN .OR. CONF .LT. ZERO) THEN
         IFAULT = 1
         RETURN
      END IF
      CONF2 = CONF
      IF (CONF .LT. SMALL) CONF2 = P95
      ALPHA = (ONE - CONF2 / HUN) / TWO
      MM = M
      IF (M .EQ. 0) MM = 1000
```

```
      M2 = MM / 2
      IF (CONF2 .GE. FIFTY) THEN
         ISTEP = .FALSE.
         AL2 = ALPHA
      ELSE
         ISTEP = .TRUE.
         AL2 = QUART
      END IF
      IF (AL2 * MM .LT. TEN / ONEPSM) THEN
         IFAULT = 5
         RETURN
      END IF
      IF (STEPK .LT. SMALL) THEN
C
C          PPND cannot fail since IFAULT=1 will have already
C          occurred if the conditions for IERR > 0 arise
C
         ZA = PPND(ONE - AL2, IERR)
         STEPC = TRT2PI * EXP(ZA ** 2 / TWO) / ZA
      ELSE
         STEPC = STEPK
      END IF
      SB = SBOUND
      BB = BBOUND
      IF (LIML) SB = -BIG
      IF (LIMU) BB = BIG
      M1 = INT(TWO / ALPHA - HALF)
      IF (M1 .LT. 39) M1 = 39
      IF (ISTART .EQ. 0) THEN
         ISTRT = INT(HALF + PTHREE * M1)
         IF (ISTRT .GT. 50) ISTRT = 50
      ELSE
         ISTRT = ISTART
      END IF
      IRSTAR = IRSTRT
      IF (CONF2 .LT. SEVFIV .AND. IRSTRT) THEN
         IFAULT = 4
         IRSTAR = .FALSE.
      END IF
      KOUNT = 0
      IF (LSTART .OR. ISTEP) THEN
C
C        Set starting points for search using the percentile method
C
         IR = INT((M1 + 1) * AL2 + HALF)
         DO 10 J = 1, IR
            CVHI(J) = -BIG
            CVLO(J) = BIG
   10    CONTINUE
C
         DO 70 I = 1, M1
C
C        Generate a bootstrap estimate, assuming the true value of
C        the parameter is EST
C
            BOOTES = SIMU(EST)
            IF (BOOTES .LT. SB / ONEPSM) IFAULT = 2
            IF (BOOTES .GT. BB * ONEPSM) IFAULT = 3
            M0 = MIN(I, IR)
C
C        Extract the IR largest bootstrap estimates
```

```
C
            DO 30 J = 1, M0
               IF (CVHI(J) .LT. BOOTES) THEN
                  IF (J .LT. M0) THEN
                     DO 20 JJ = M0, J + 1, -1
                        CVHI(JJ) = CVHI(JJ - 1)
   20                CONTINUE
                  END IF
                  CVHI(J) = BOOTES
                  GO TO 40
               END IF
   30       CONTINUE
C
C        Similarly for the lower limit
C
   40       DO 60 J = 1, M0
               IF (CVLO(J) .GT. BOOTES) THEN
                  IF (J .LT. M0) THEN
                     DO 50 JJ = M0, J + 1, -1
                        CVLO(JJ) = CVLO(JJ - 1)
   50                CONTINUE
                  END IF
                  CVLO(J) = BOOTES
                  GO TO 70
               END IF
   60       CONTINUE
   70    CONTINUE
         IF (ISTEP) THEN
            STEPHI = (CVHI(IR) - EST) * STEPC
            STEP = (CVLO(IR) - EST) * STEPC
            CLIMHI = EST + CONF2 * (CVHI(IR) - EST) / FIFTY
            CLIMLO = EST - CONF2 * (EST - CVLO(IR)) / FIFTY
         ELSE
C
C        The starting point for the upper limit will be the IRth
C        largest of the M1 bootstrap estimates, and that for the
C        lower limit the IRth smallest.  For CONF2 > 87.5%, IR=2
C
            CLIMHI = CVHI(IR)
            CLIMLO = CVLO(IR)
         END IF
      END IF
      IF ( .NOT. LSTART) THEN
C
C        Starting points supplied by the user
C
         CLIMHI = SHILIM
         CLIMLO = SLOLIM
      END IF
      IF (IFAULT .GT. 0 .AND. IFAULT .NE. 4) RETURN
C
C        Start by searching for the lower limit
C
      POSN = CLIMLO
      SEARCH(1, 1) = POSN
      BEGPOS = POSN
      IF (IRSTAR) D1 = ABS(EST - BEGPOS)
      LIMIT = 1
   80 I = ISTRT + 1
      II = 1
C
```

```
C        Generate a bootstrap estimate assuming the true value of the
C        parameter is POSN, the current position of the search
C
   90 BOOTES = SIMU(POSN)
      IF (BOOTES .LT. SB / ONEPSM) THEN
         IFAULT = 2
         RETURN
      ELSE IF (BOOTES .GT. BB * ONEPSM) THEN
         IFAULT = 3
         RETURN
      END IF
      IF (LIMIT .EQ. 1) THEN
         X = EST - BOOTES
      ELSE
         X = BOOTES - EST
      END IF
      IF (X .LT. SMALL * ABS(EST)) THEN
         X = ONE
      ELSE
         X = ZERO
      END IF
      IF ( .NOT. ISTEP) THEN
C
C        Calculate the current estimate of the steplength constant
C
         STEP = (POSN - EST) * STEPC
         IF (MODSTP) THEN
            STEP2 = BB - POSN
            IF (LIMIT .EQ. 1) STEP2 = POSN - SB
            STEP2 = (STEP2 / ALPHA) + STEP * FLOAT(ISTRT) / FLOAT(I)
            IF (STEP2 .LT. STEP) STEP = STEP2
         END IF
      END IF
C
C        Step to the next position in the search
C
      POSN = POSN + STEP * (X - ALPHA) / FLOAT(I)
C
C        Constrain the search to lie within the bounds to the
C        parameter space
C
      IF (POSN .LT. SB) POSN = SB
      IF (POSN .GT. BB) POSN = BB
      I = I + 1
      II = II + 1
      IIM = II / MULT(LIMIT)
      IF (II .EQ. MULT(LIMIT) * IIM) SEARCH(LIMIT, IIM) = POSN
C
C        If array SEARCH is full, reduce the number of points stored
C        by a factor of 10
      IF (IIM .EQ. 200) THEN
         MULT(LIMIT) = MULT(LIMIT) * 10
         DO 100 J = 1, 200
            SEARCH(LIMIT, J) = SEARCH(LIMIT, J * 10)
  100    CONTINUE
      END IF
C
C        Restart search if distance between POSN and EST has halved
C        or doubled and IRSTAR=.TRUE.
C
      IF (IRSTAR) THEN
```

```
            D2 = ABS(EST - POSN)
            IF (D1 .GT. TWO * D2 .OR. D1 .LT. HALF * D2) THEN
               I = ISTRT
               BEGPOS = POSN
               D1 = D2
            END IF
         END IF
C
C        Go to the next step if more steps are required
         IF (I .LT. MM + ISTRT .AND. INDIC .EQ. 0) GO TO 90
         IF (I .LT. M2 + ISTRT) GO TO 90
C
C        End current search
C
         IF (INDIC .GT. 0) THEN
            SEPOSN = ABS(STEP) * SQRT(ALPHA * (ONE - ALPHA) / FLOAT(I - 1))
            I = ISTRT
            BEGPOS = POSN
            IF (IRSTAR) D1 = D2
            KOUNT = KOUNT + 1
C
C        Test whether endpoint of the current search differs
C        significantly from that of previous search
C
            IF (KOUNT .GT. 1 .AND. INDIC .GT. 1) THEN
               IF (SEPOSN .LT. SMALL * POSN) THEN
                  Z = ZERO
               ELSE
                  Z = ABS(POSN - ENDPOS) / (ROOT2 * SEPOSN)
               END IF
               IF (Z .GT. Z1) THEN
                  IF (INDIC .EQ. 2) THEN
                     KOUNT = 4
                  ELSE
                     KOUNT = KOUNT - 1
                  END IF
               END IF
            END IF
C
C        Is a further search required?
C
            IF (KOUNT .EQ. 1) THEN
               ENDPOS = POSN
               GO TO 90
            ELSE IF (KOUNT .NE. 4) THEN
C
C        Average the two end points selected to obtain the estimated
C        confidence limit.  If current search is for the lower limit,
C        start the search for the upper limit
C
               HIPOS = CLIMHI
               CLIMHI = (POSN + ENDPOS) / TWO
               CLHISE = SEPOSN / ROOT2
               MHI = II
               IF (LIMIT .EQ. 1) THEN
                  CLIMLO = CLIMHI
                  CLLOSE = CLHISE
                  MLO = MHI
                  LIMIT = 2
                  KOUNT = 0
                  POSN = HIPOS
```

```
                  SEARCH(2, 1) = POSN
                  BEGPOS = POSN
                  IF (IRSTAR) D1 = ABS(EST - BEGPOS)
                  IF (ISTEP) STEP = STEPHI
                  GO TO 80
               END IF
            END IF
         END IF
C
C        Use the final endpoint as the estimate of the confidence
C        limit.  If current search is for the lower limit, start the
C        search for the upper limit
C
         IF (INDIC .EQ. 0 .OR. KOUNT .EQ. 4) THEN
            IF (INDIC .EQ. 0) SEPOSN = ABS(STEP) *
     *                                 SQRT(ALPHA * (ONE - ALPHA) /
     *                                 FLOAT(I - 1))
            IF (LIMIT .EQ. 1) THEN
               CLIMLO = POSN
               CLLOSE = SEPOSN
               MLO = II
               LIMIT = 2
               KOUNT = 0
               POSN = CLIMHI
               SEARCH(2, 1) = POSN
               BEGPOS = POSN
               IF (IRSTAR) D1 = ABS(EST - BEGPOS)
               IF (ISTEP) STEP = STEPHI
               GO TO 80
            ELSE
               CLIMHI = POSN
               CLHISE = SEPOSN
               MHI = II
            END IF
         END IF
C
C        Enter the path of the search for each limit into the array
C        PTHPLT so that it may be checked by the user for convergence
C
         DO 120 K = 1, 105
            DO 110 J = 1, 76
               PTHPLT(J, K) = ' '
  110       CONTINUE
  120    CONTINUE
         DO 130 J = 1, 28
            PTHPLT(J + 20, 1) = HIPLOT(J:J)
            PTHPLT(J + 20, 105) = LOPLOT(J:J)
  130    CONTINUE
         YMAX = -BIG
         YMIN = BIG
         DO 150 I = 1, 2
            IF (I .EQ. 1) THEN
               NPT = MLO / MULT(1)
            ELSE
               NPT = MHI / MULT(2)
            END IF
            DO 140 J = 1, NPT
               IF (SEARCH(I, J) .GT. YMAX) YMAX = SEARCH(I, J)
               IF (SEARCH(I, J) .LT. YMIN) YMIN = SEARCH(I, J)
  140       CONTINUE
  150    CONTINUE
```

```
      DIV = YMAX - YMIN
      IF (DIV .LE. ZERO) DIV = ONE
      MID = HUN * (YMAX - EST) / DIV + THREP5
      YSCALE = HUN / DIV
      DO 160 J = 1, 30
         PTHPLT(J, MID) = MIDLIN(J:J)
         PTHPLT(J + 46, MID) = MIDLIN(J:J)
         IF (J .LE. 16) PTHPLT(J + 30, MID) = MIDPLT(J:J)
  160 CONTINUE
      DO 210 I = 1, 2
         IF (I .EQ. 1) THEN
            NPT = MLO / MULT(1)
            LYY = YSCALE * (CLIMLO - YMIN) + HALF
            SYMB = '<'
         ELSE
            NPT = MHI / MULT(2)
            LYY = YSCALE * (CLIMHI - YMIN) + HALF
            SYMB = '>'
         END IF
         DO 170 J = 1, 76
            LX = (NPT - 1) * (J - 1) / SEVFIV + ONE + HALF
            IX(J) = LX
            LY = YSCALE * (SEARCH(I, LX) - YMIN) + HALF
            PTHPLT(J, 103 - LY) = SYMB
            IF (PTHPLT(J, 103 - LYY) .EQ. ' ')
     *          PTHPLT(J, 103 - LYY) = '.'
  170    CONTINUE
         DO 190 J = 1, 15
            JX = IX(5 * J)
            DO 180 L = 0, 4
               LL = 10 ** L
               IF (JX .GE. LL) THEN
                  LR = JX / LL - 10 * (JX / (LL * 10))
                  PTHPLT(5 * J - L, 2 + (2 - I) * 102) = DIGIT(LR)
               END IF
  180       CONTINUE
  190    CONTINUE
         IF (MULT(I) .GT. 1) THEN
            KMULT = LOG10(FLOAT(MULT(I))) + HALF
            IY = 1 + (2 - I) * 104
            PTHPLT(50, IY) = '/'
            PTHPLT(52, IY) = '1'
            IPTH = 53
            DO 200 K = 1, KMULT
               PTHPLT(IPTH, IY) = '0'
               IPTH = IPTH + 1
  200       CONTINUE
         END IF
  210 CONTINUE
      RETURN
      END
```

```
      REAL FUNCTION SQMCOR(X, IP, N, RHO2, IFAULT)
C
C        ALGORITHM AS 260  APPL. STATIST. (1991) VOL. 40, NO. 1
C
C        Computes the C.D.F. for the distribution of the
C        square of the multiple correlation coefficient
C        with parameters IP, N, and RHO2
C
C        The following auxiliary algorithms are required:
C        ALOGAM - Log-gamma function (CACM 291)
C         (or  ALNGAM    AS 245)
C        BETAIN - Incomplete beta function (AS 63)
C
      REAL X, RHO2
      INTEGER IP, N, IFAULT
      REAL A, B, BETA, ERRBD, ERRMAX, GX, Q, SUMQ, TEMP, TERM, XJ,
     *     ZERO, HALF, ONE
      INTEGER ITRMAX
      REAL ALOGAM, BETAIN
      EXTERNAL ALOGAM, BETAIN
C
      DATA ERRMAX, ITRMAX / 1.0E-6, 100 /
      DATA ZERO, HALF, ONE / 0.0, 0.5, 1.0 /
C
      SQMCOR = X
      IFAULT = 2
      IF (RHO2 .LT. ZERO .OR. RHO2 .GT. ONE .OR. IP .LT. 2 .OR.
     *    N .LE. IP) RETURN
      IFAULT = 3
      IF (X .LT. ZERO .OR. X .GT. ONE) RETURN
      IFAULT = 0
      IF (X .EQ. ZERO .OR. X .EQ. ONE) RETURN
C
      A = HALF * (IP - 1)
      B = HALF * (N - IP)
C
C        Initialize the series
C
      BETA = EXP(ALOGAM(A, IFAULT) + ALOGAM(B, IFAULT) -
     *       ALOGAM(A + B, IFAULT))
      TEMP = BETAIN(X, A, B, BETA, IFAULT)
C
C        There is no need to test IFAULT since all of the
C        parameter values have already been checked
C
      GX = EXP(A * LOG(X) + B * LOG(ONE - X) - LOG(A)) / BETA
      Q = (ONE - RHO2) ** (A + B)
      XJ = ZERO
      TERM = Q * TEMP
      SUMQ = ONE - Q
      SQMCOR = TERM
C
C        Perform recurrence until convergence is achieved
C
   10 XJ = XJ + ONE
      TEMP = TEMP - GX
      GX = GX * (A + B + XJ - ONE) * X / (A + XJ)
      Q = Q * (A + B + XJ - ONE) * RHO2 / XJ
      SUMQ = SUMQ - Q
      TERM = TEMP * Q
      SQMCOR = SQMCOR + TERM
```

```
C
C        Check for convergence and act accordingly
C
      ERRBD = (TEMP - GX) * SUMQ
      IF ((INT(XJ) .LT. ITRMAX) .AND. (ERRBD .GT. ERRMAX)) GO TO 10
      IF (ERRBD .GT. ERRMAX) IFAULT = 1
      RETURN
      END
============================
The author of algorithms AS 260 & 261 has found an improved algorithm
which has been published in the journal `Computational Statistics &
Data Analysis'.   Here it is.
C-------------------------------------------------------------
C
C The following code for the distribution function of R^2 (cdfr2.for)
C and for its probability density (pdfr2.for) is by the author of AS 260
C and uses an improved algorithm.

      REAL FUNCTION CDFR2(Y, M, SIZE, RHO2, IFAULT)
C
C  Computes the distribution function of the square of the sample
C  multiple correlation coefficient for given abscissa Y, number
C  of random variables M, sample size SIZE, and square of the population
C  multiple correlation coefficient RHO2
C
C  Reference:
C  Ding, C.G. (1996) `On the computation of the distribution of
C  the square of the sample multiple correlation coefficient',
C  Comput. Statist. & Data Analysis, vol. 22, 345-350.
C
C  IFAULT is a fault indicator:
C  = 1 if any of the input values is illegal
C  = 0 otherwise
C
C  No auxiliary algorithm is required
C
      INTEGER SIZE
      REAL N
      DATA EPS / 1.0E-6 /
      DATA HALF, ZERO, ONE, TWO / 0.5, 0.0, 1.0, 2.0 /
      DATA RP / 1.772453850905516028 /

      CDFR2 = ZERO
      IFAULT = 1
      IF (M .LE. 1 .OR. SIZE .LE. M .OR. RHO2 .LT. ZERO .OR.
     $    RHO2 .GT. ONE) RETURN
      IFAULT = 0
      IF (Y .LE. ZERO) RETURN
      CDFR2 = ONE
      IF (Y .GE. ONE) RETURN
      A = (M - 1) / TWO
      B = (SIZE - M) / TWO
      AB = (SIZE - 1) / TWO
      IF (MOD(M + 1, 2) .EQ. 0) THEN
         NA = A + HALF
         GA = ONE
         DO 10 I = 1, NA
   10    GA = GA * I
      ELSE
         NA = A + ONE
         GA = RP
```

```
            DO 20 I = 1, NA
      20    GA = GA * (I - HALF)
         ENDIF
         IF (MOD(SIZE - M, 2) .EQ. 0) THEN
            NB = B - HALF
            GB = ONE
            IF (NB .EQ. 0) GO TO 50
            DO 30 I = 1, NB
      30    GB = GB * I
         ELSE
            NB = B
            GB = RP
            IF (NB .EQ. 0) GO TO 50
            DO 40 I = 1, NB
      40    GB = GB * (I - HALF)
         ENDIF
      50 IF (MOD(SIZE - 1, 2) .EQ. 0) THEN
            NAB = AB - HALF
            GAB = ONE
            IF (NAB .EQ. 0) GO TO 80
            DO 60 I = 1, NAB
      60    GAB = GAB * I
         ELSE
            NAB = AB
            GAB = RP
            DO 70 I = 1, NAB
      70    GAB = GAB * (I - HALF)
         ENDIF

C
C        Evaluate the first term
C
      80 N = ONE
         Q = (ONE - RHO2) ** AB
         V = Q
         T = Y ** A * (ONE - Y) ** B * GAB / GA / GB
         TERM = V * T
         CDFR2 = TERM
C
C        Check if a + n > (a + b + n)y
C
      90 IF (A + N .GT. (A + B + N) * Y) GO TO 100
C
C        Evaluate the next term of the expansion and then the
C        partial sum
C
         Q = Q * (A + B + N - ONE) * RHO2 / N
         V = V + Q
         T = T * Y * (A + B + N - ONE) / (A + N)
         TERM = V * T
         CDFR2 = CDFR2 + TERM
         N = N + ONE
         GO TO 90
C
C        Find the error bound and check for convergence
C
     100 BOUND = T * Y * (A + B + N - ONE) / ((A + N) - (A + B + N) * Y)
         IF (BOUND .LE. EPS) RETURN
C
C        Evaluate the next term of the expansion and then the
C        partial sum
```

```
C
      Q = Q * (A + B + N - ONE) * RHO2 / N
      V = V + Q
      T = T * Y * (A + B + N - ONE) / (A + N)
      TERM = V * T
      CDFR2 = CDFR2 + TERM
      N = N + ONE
      GO TO 100
      END
C
      PROGRAM MAIN
C
C        is a driver program that calls CDFR2 and produces output
C
      INTEGER SIZE
   10 WRITE (*,11)
   11 FORMAT (/1X,'ENTER Y, M (>1), N (>M), RHO2 (BETWEEN 0 AND 1)',
     $        /1X,'FOR CDFR2 ==>')
      READ (*,*) Y, M, SIZE, RHO2
      CDF = CDFR2(Y, M, SIZE, RHO2, IFAULT)
      IF (IFAULT .EQ. 0) THEN
         WRITE (*,21) Y, M, SIZE, RHO2, CDF
   21    FORMAT (/1X, 'CDFR2(', E10.4, ',', 2(I3, ','), E10.4, ') =',
     $           E12.6)
      ELSE
         WRITE (*,31)
   31    FORMAT (/1X,'THE INPUT VALUE IS ILLEGAL!')
      ENDIF
      WRITE(*,41)
   41 FORMAT (/1X, 'ENTER 1 TO CONTINUE OR 0 TO QUIT ==> ')
      READ (*,*)K
      IF (K .EQ. 1) GOTO 10
      STOP
      END


C-------------------------------------------------------------
C


      REAL FUNCTION PDFR2(Y, M, SIZE, RHO2, IFAULT)
C
C  Computes the density (pdf) of the square of the sample multiple
C  correlation coefficient for given abscissa Y, number of random
C  variables M, sample size SIZE, and square of the population
C  multiple correlation coefficient RHO2
C
C  Reference:
C  Ding, C.G. (1996) `On the computation of the distribution of
C  the square of the sample multiple correlation coefficient',
C  Comput. Statist. & Data Analysis, vol. 22, 345-350.
C
C  IFAULT is a fault indicator:
C  = 1 if any of the input values is illegal
C  = 0 otherwise
C
C  No auxiliary algorithm is required
C
      INTEGER SIZE
      REAL N
      DATA EPS / 1.0E-6 /
```

```
      DATA HALF, ZERO, ONE, TWO / 0.5, 0.0, 1.0, 2.0 /
      DATA RP / 1.772453850905516028 /

      PDFR2 = ZERO
      IFAULT = 1
      IF (M .LE. 1 .OR. SIZE .LE. M .OR. RHO2 .LT. ZERO .OR.
     $    RHO2 .GT. ONE) RETURN
      IFAULT = 0
      IF (Y .LE. ZERO .OR. Y .GE. ONE) RETURN
      A = (M - 1) / TWO
      B = (SIZE - M) / TWO
      AB = (SIZE - 1) / TWO
      IF (MOD(M - 1, 2) .EQ. 0) THEN
         NA = A - HALF
         GA = ONE
         IF (NA .EQ. 0) GO TO 25
         DO 10 I = 1, NA
   10    GA = GA * I
      ELSE
         NA = A
         GA = RP
         IF (NA .EQ. 0) GO TO 25
         DO 20 I = 1, NA
   20    GA = GA * (I - HALF)
      ENDIF
   25 IF (MOD(SIZE - M, 2) .EQ. 0) THEN
         NB = B - HALF
         GB = ONE
         IF (NB .EQ. 0) GO TO 50
         DO 30 I = 1, NB
   30    GB = GB * I
      ELSE
         NB = B
         GB = RP
         IF (NB .EQ. 0) GO TO 50
         DO 40 I = 1, NB
   40    GB = GB * (I - HALF)
      ENDIF
   50 IF (MOD(SIZE - 1, 2) .EQ. 0) THEN
         NAB = AB - HALF
         GAB = ONE
         IF (NAB .EQ. 0) GO TO 80
         DO 60 I = 1, NAB
   60    GAB = GAB * I
      ELSE
         NAB = AB
         GAB = RP
         DO 70 I = 1, NAB
   70    GAB = GAB * (I - HALF)
      ENDIF

C
C        Evaluate the first term
C
   80 N = ONE
      Q = (ONE - RHO2) ** AB
      V = Q
      S = Y ** (A - ONE) * (ONE - Y) ** (B - ONE) * GAB / GA / GB
      TERM = Q * S
      PDFR2 = TERM
C
```

```
C        Check if a + n > (a + b + n)y
C
   90 IF (A + N .GT. (A + B + N) * Y) GO TO 100
C
C        Evaluate the next term of the expansion and then the
C        partial sum
C
      Q = Q * (A + B + N - ONE) * RHO2 / N
      V = V + Q
      S = S * Y * (A + B + N - ONE) / (A + N - ONE)
      TERM = Q * S
      PDFR2 = PDFR2 + TERM
      N = N + ONE
      GO TO 90
C
C        Find the error bound and check for convergence
C
  100 BOUND = S * Y * (A + B + N - ONE) * (ONE - V) / (A + N - ONE)
      IF (BOUND .LE. EPS) RETURN
C
C        Evaluate the next term of the expansion and then the
C        partial sum
C
      Q = Q * (A + B + N - ONE) * RHO2 / N
      V = V + Q
      S = S * Y * (A + B + N - ONE) / (A + N - ONE)
      TERM = Q * S
      PDFR2 = PDFR2 + TERM
      N = N + ONE
      GO TO 100
      END
C
      PROGRAM MAIN
C
C        is a driver program that calls PDFR2 and produces output
C
      INTEGER SIZE

   10 WRITE (*,11)
   11 FORMAT (/1X,'ENTER Y, M (>1), N (>M), RHO2 (BETWEEN 0 AND 1)',
     $        /1X,'FOR PDFR2 ==> ')
      READ (*,*) Y, M, SIZE, RHO2
      PDF = PDFR2(Y, M, SIZE, RHO2, IFAULT)
      IF (IFAULT .EQ. 0) THEN
         WRITE (*,21) Y, M, SIZE, RHO2, PDF
   21    FORMAT (/1X, 'PDFR2(', E10.4, ',', 2(I3, ','), E10.4, ') =',
     $           E12.6)
      ELSE
         WRITE (*,31)
   31    FORMAT (/1X, 'THE INPUT VALUE IS ILLEGAL!')
      ENDIF
      WRITE(*,41)
   41 FORMAT (/1X, 'ENTER 1 TO CONTINUE OR 0 TO QUIT ==> ')
      READ (*,*)K
      IF (K .EQ. 1) GOTO 10
      STOP
      END
```

```
      REAL FUNCTION SQMCQ(CDF, IP, N, RHO2, IFAULT)
C
C        ALGORITHM AS 261  APPL. STATIST. (1991) VOL. 40, NO. 1
C
C        Returns the quantile of the distribution of the square
C        of the sample multiple correlation coefficient for
C        given values of CDF, IP, N, and RHO2
C
C        A modification of the secant method, called the
C        Illinois method, is used
C
C        The auxiliary algorithm SQMCOR, used to compute the
C        C.D.F. of the square of the sample multiple correlation
C        coefficient, is required
C
      INTEGER IP, N, IFAULT
      REAL CDF, RHO2
      REAL DIFF, EPS, F0, F1, F2, X0, X1, X2, ZERO, ONE, TWO
      REAL SQMCOR
      EXTERNAL SQMCOR
      DATA ZERO, ONE, TWO / 0.0, 1.0, 2.0 /
      DATA EPS / 1.0E-6 /
C
      SQMCQ = CDF
      IFAULT = 2
C
C        Perform domain check
C
      IF (RHO2 .LT. ZERO .OR. RHO2 .GT. ONE .OR. IP .LT. 2 .OR.
     *    N .LE. IP) RETURN
      IFAULT = 3
      IF (CDF .LT. ZERO .OR. CDF .GT. ONE) RETURN
      IFAULT = 0
      IF (CDF .EQ. ZERO .OR. CDF .EQ. ONE) RETURN
C
C        Use ONE and ZERO as two starting points for the
C        Illinois method
C
      X0 = ONE
      F0 = ONE - CDF
      X1 = ZERO
      F1 = -CDF
C
C        Continue iterations until convergence is achieved
C
   10 DIFF = F1 * (X1 - X0) / (F1 - F0)
      X2 = X1 - DIFF
      F2 = SQMCOR(X2, IP, N, RHO2, IFAULT) - CDF
      IF (IFAULT .NE. 0) RETURN
C
C        Check for convergence
C
      IF (ABS(F2) .LE. EPS * CDF) GO TO 20
      IF (ABS(DIFF) .LE. EPS * X2) GO TO 20
      IF (F2 * F1 .GE. ZERO) THEN
         F0 = F0 / TWO
      ELSE
         X0 = X1
         F0 = F1
      END IF
      X1 = X2
```

```
      F1 = F2
      GO TO 10
C
   20 SQMCQ = X2
      RETURN
      END
==========================
The author of algorithms AS 260 & 261 has found an improved algorithm
which has been published in the journal `Computational Statistics &
Data Analysis'.   Here it is.
C------------------------------------------------------------
C
C The following code for calculating quantiles of R^2 (qr2.for)
C is by the author of AS 261 and uses an improved algorithm.

      REAL FUNCTION QR2(M, SIZE, RHO2, P, IFAULT)
C
C  Computes the quantile of the distribution of the square of the
C  sample multiple correlation coefficient for given number of
C  random variables M, sample size SIZE, square of the population
C  multiple correlation coefficient RHO2, and lower tail area P
C
C  Reference:
C  Ding, C.G. (1996) `On the computation of the distribution of
C  the square of the sample multiple correlation coefficient',
C  Comput. Statist. & Data Analysis, vol. 22, 345-350.
C
C  IFAULT is a fault indicator:
C  = 1 if there is no convergence after 10 Newton's iterations
C  = 2 if any of the input values is illegal
C  = 0 otherwise
C
C  No auxiliary algorithm is required
C
      INTEGER SIZE
      REAL N
      DATA ZERO, HALF, ONE, TWO / 0.0, 0.5, 1.0, 2.0 /
      DATA EPS, DELTA / 1.0E-6, 1.0E-4 /
      DATA ITRMAX / 10 /
      DATA RP / 1.772453850905516028 /

      QR2 = P
C
C        Test for admissibility of arguments
C
      IFAULT = 2
      IF (M .LE. 1 .OR. SIZE .LE. M .OR. RHO2 .LT. ZERO .OR.
     $    RHO2 .GT. ONE .OR. P .LT. ZERO .OR. P .GT. ONE) RETURN
      IFAULT = 0
      IF (P .EQ. ZERO .OR. P .EQ. ONE) RETURN
C
C        Calculate the constants needed for each Newton's iteration
C
      A = (M - 1) / TWO
      B = (SIZE - M) / TWO
      AB = (SIZE - 1) / TWO
      IF (MOD(M + 1, 2) .EQ. 0) THEN
         NA = A + HALF
         GA = ONE
         DO 10 I = 1, NA
   10    GA = GA * I
```

```
      ELSE
         NA = A + ONE
         GA = RP
         DO 20 I = 1, NA
   20    GA = GA * (I - HALF)
      ENDIF
      IF (MOD(SIZE - M, 2) .EQ. 0) THEN
         NB = B - HALF
         GB = ONE
         IF (NB .EQ. 0) GO TO 50
         DO 30 I = 1, NB
   30    GB = GB * I
      ELSE
         NB = B
         GB = RP
         IF (NB .EQ. 0) GO TO 50
         DO 40 I = 1, NB
   40    GB = GB * (I - HALF)
      ENDIF
   50 IF (MOD(SIZE - 1, 2) .EQ. 0) THEN
         NAB = AB - HALF
         GAB = ONE
         IF (NAB .EQ. 0) GO TO 75
         DO 60 I = 1, NAB
   60    GAB = GAB * I
      ELSE
         NAB = AB
         GAB = RP
         DO 70 I = 1, NAB
   70    GAB = GAB * (I - HALF)
      ENDIF
   75 Q0 = (ONE - RHO2) ** AB
      COEFF = GAB / GA / GB
C
C        Use 0.5 as a starting value for Newton's iterations
C
      Y = HALF
C
C        Perform Newton's iterations
C
      DO 120 ITER = 1, ITRMAX
C
C        Evaluate the first terms of the series for CDF (distribution
C        function) and PDF (density)
C
      N = ONE
      YP = ONE - Y
      T = COEFF * Y ** A * YP ** B
      S = A * T / Y / YP
      Q = Q0
      V = Q
      CDF = V * T
      PDF = Q * S
C
C        Check if a + n > (a + b + n)y
C
   80 IF (A + N .GT. (A + B + N) * Y) GO TO 90
C
C        Evaluate the next terms of two series and then the
C        partial sums
C
```

```
      Q = Q * (A + B + N - ONE) * RHO2 / N
      V = V + Q
      S = T * (A + B + N - ONE) / YP
      T = T * Y * (A + B + N - ONE) / (A + N)
      CDF = CDF + V * T
      PDF = PDF + Q * S
      N = N + ONE
      GO TO 80
C
C        Find the error bounds and check for convergence for both
C        series
C
   90 BNDCDF = T * Y * (A + B + N - ONE) / (A + N - (A + B + N) * Y)
      BNDPDF = T * (A + B + N - ONE) * (ONE - V) / YP
  100 IF (BNDCDF .LE. EPS .AND. BNDPDF .LE. EPS) GO TO 110
C
C        Continue to update the terms and then accumulate
C
      Q = Q * (A + B + N - ONE) * RHO2 / N
      V = V + Q
      IF (BNDCDF .LE. EPS) THEN
         S = S * Y * (A + B + N - ONE) / (A + N - ONE)
         PDF = PDF + Q * S
         N = N + ONE
         BNDPDF = S * Y * (A + B + N - ONE) * (ONE - V) / (A + N - ONE)
         GO TO 100
      ELSE IF (BNDPDF .LE. EPS) THEN
              T = T * Y * (A + B + N - ONE) / (A + N)
              CDF = CDF + V * T
              N = N + ONE
              BNDCDF = T * Y * (A+B+N-ONE) / (A+N - (A+B+N)*Y)
              GO TO 100
           ELSE
              S = T * (A + B + N - ONE) / YP
              T = T * Y * (A + B + N - ONE) / (A + N)
              CDF = CDF + V * T
              PDF = PDF + Q * S
              N = N + ONE
              GO TO 90
      ENDIF
C
C        Obtain a new Y and make changes if it is illegal
C
  110 DIFF = (CDF - P) / PDF
      YNEW = Y - DIFF
      IF (YNEW .LE. ZERO) THEN
         Y = Y / TWO
      ELSE IF (YNEW .GE. ONE) THEN
              Y = (Y + ONE) / TWO
           ELSE
              Y = YNEW
      ENDIF
C
C        Check for convergence of Newton's iterations
C
      IF (ABS(DIFF) .LE. DELTA * Y) THEN
         QR2 = Y
         RETURN
      ENDIF
  120 CONTINUE
      IFAULT = 1
```

```
      RETURN
      END
C
      PROGRAM MAIN
C
C        This is a driver program that calls QR2 and produces output
C
      INTEGER SIZE

   10 WRITE (*,11)
   11 FORMAT (/1X,'ENTER M (>1), N (>M), RHO2 (BETWEEN 0 AND 1), and',
     $        /1X,'P (BETWEEN 0 AND 1) FOR QR2 ==> ')
      READ (*,*) M, SIZE, RHO2, P
      Y = QR2(M, SIZE, RHO2, P, IFAULT)
      ICODE = IFAULT + 1
      GO TO (20, 30, 40), ICODE
   20 WRITE (*,21) M, SIZE, RHO2, P, Y
   21 FORMAT (/1X, 'QR2(', I3, ',', I3, ',', E10.4, ',', E10.4, ') =',
     $        E11.5)
      GO TO 60
   30 WRITE (*,31)
   31 FORMAT (/1X,43HNO CONVERGENCE AFTER 10 NEWTON'S ITERATIONS)
      GO TO 60
   40 WRITE (*,41)
   41 FORMAT (/1X,'THE INPUT VALUE IS ILLEGAL!')
   60 WRITE (*,61)
   61 FORMAT (/1X, 'ENTER 1 TO CONTINUE OR 0 TO QUIT ==> ')
      READ (*,*) K
      IF (K .EQ. 1) GO TO 10
      STOP
      END
```

```
      SUBROUTINE WL(NR, N, NN, G, X, S, D, SIGMA, SIGINV, WLT, NRUNIV,
     *              CHIOMB, NRSTOC, METHOD, KERROR, IFAULT)
C
C        ALGORITHM AS 262  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Wei-Lachin multivariate generalization of the generalized
C        Wilcoxon test of Gehan and the log-rank test for incomplete
C        multivariate observations with censoring.
C
      INTEGER NREP, NOBS
      PARAMETER (NREP=10, NOBS=500)
C
C        NREP     the maximum number of repeat times
C        NOBS     the maximum number of observations
C
      INTEGER NR, N(2), NN, G(NOBS), S(NOBS, NREP), METHOD, KERROR,
     *        IFAULT
      REAL X(NOBS, NREP), D(2, NREP), SIGMA(NN), SIGINV(NN), WLT(NREP),
     *     NRUNIV(NREP), CHIOMB, NRSTOC
      INTEGER Y(2, NOBS), I, IFAIL, IPOINT, NULLTY, J, J2, K, K1,
     *        K2, NT
      REAL EE(2, NREP), MU(2, NOBS, NREP), PSI(2, NOBS, NREP), QE(2),
     *     SIG(2, NREP, NREP), WORK(NREP), Q, SIGSUM, SUM, TSUM, Z,
     *     ZERO, ONE, TWO
C
      DATA ZERO, ONE, TWO / 0.0, 1.0, 2.0 /
C
      ZSQRT(I) = SQRT(Z)
      ZREAL(Z) = REAL(I)
C
C        Initialization
C
      IFAULT = 4
      IF (NR .GT. NREP) RETURN
      IFAULT = 5
      NT = N(1) + N(2)
      IF (NT .GT. NOBS) RETURN
      IFAULT = 0
      DO 30 K = 1, NR
         DO 20 I = 1, 2
            EE(I, K) = ZERO
            D(I, K) = ZERO
            DO 10 K2 = 1, NR
               SIG(I, K, K2) = ZERO
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
C
C        Y    number at risk of failure
C        EE   partial sum used in equation 1
C        QE   Q*E (see equation 4)
C        Evaluate MU (equation 4)
C
      DO 110 K = 1, NR
         DO 70 J = 1, NT
            DO 40 I = 1, 2
               Y(I, J) = 0
               MU(I, J, K) = ZERO
   40       CONTINUE
            IF (S(J, K) .EQ. 1) THEN
               DO 50 J2 = 1, NT
```

```
                  IF (X(J2, K) .GE. X(J, K))
     *               Y(G(J2), J) = Y(G(J2), J) + 1
    50          CONTINUE
                SUM = Y(1, J) + Y(2, J)
                IF (SUM .EQ. ZERO) THEN
                   IFAULT = 1
                   RETURN
                END IF
                IF (METHOD .EQ. 1) Q = SUM / NT
                IF (METHOD .EQ. 2) Q = ONE
                DO 60 I = 1, 2
                   QE(I) = Q * Y(I, J) / SUM
    60          CONTINUE
                MU(2, J, K) = QE(1)
                MU(1, J, K) = QE(2)
                EE(G(J), K) = EE(G(J), K) + MU(G(J), J, K)
                D(G(J), K) = D(G(J), K) + ONE
             ELSE
                MU(1, J, K) = ZERO
                MU(2, J, K) = ZERO
             END IF
    70    CONTINUE
C
C        Evaluate PSI (equation 5)
C
          DO 100 J = 1, NT
             DO 90 I = 1, 2
                PSI(I, J, K) = ZERO
                DO 80 J2 = 1, NT
                   IF (G(J2) .EQ. I .AND. X(J2, K) .LE. X(J, K)) THEN
                      IF (S(J2, K) .EQ. 1) THEN
                         PSI(I, J, K) = PSI(I, J, K) +
     *                                   MU(I, J2, K) / Y(I, J2)
                      END IF
                   END IF
    80          CONTINUE
    90       CONTINUE
   100    CONTINUE
   110 CONTINUE
C
C        Compute entries in covariance matrix (see equations 2 and 3)
C
       DO 140 K1 = 1, NR
          DO 130 K2 = 1, K1
             DO 120 J = 1, NT
                I = G(J)
                IF (N(I) .EQ. 0) THEN
                   IFAULT = 2
                   RETURN
                END IF
                SIG(I, K1, K2) = SIG(I, K1, K2) +
     *                           (MU(I, J, K1) * S(J, K1) -
     *                           PSI(I, J, K1)) * (MU(I, J, K2) *
     *                           S(J, K2) - PSI(I, J, K2)) / N(I)
   120       CONTINUE
             IPOINT = K1 * (K1 - 1) / 2 + K2
             SIGMA(IPOINT) = (N(1) * SIG(1, K1, K2) +
     *                        N(2) * SIG(2, K1, K2)) / NT
   130    CONTINUE
   140 CONTINUE
C
```

```
C        Compute Wei-Lachin univariate test NRUNIV  (see eqns 1 and 6)
C
      DO 150 K = 1, NR
         WLT(K) = (EE(1, K) - EE(2, K)) / ZSQRT(ZREAL(NT))
         IPOINT = K * (K + 1) / 2
         IF (SIGMA(IPOINT) .EQ. ZERO) THEN
            IFAULT = 3
            KERROR = K
            RETURN
         END IF
         NRUNIV(K) = WLT(K) / ZSQRT(SIGMA(IPOINT))
  150 CONTINUE
C
C        Compute inverse of covariance matrix and Wei-Lachin
C        multivariate statistics CHIOMB (for omnibus test) and
C        NRSTOC (for test of stochastic ordering)  (see eqn 7)
C
      CALL SYMINV(SIGMA, NR, NN, SIGINV, WORK, NULLTY, IFAIL)
      CHIOMB = ZERO
      TSUM = ZERO
      SIGSUM = ZERO
      DO 170 J = 1, NR
         TSUM = TSUM + WLT(J)
         IPOINT = J * (J + 1) / 2
         CHIOMB = CHIOMB + WLT(J) * WLT(J) * SIGINV(IPOINT)
         SIGSUM = SIGSUM + SIGMA(IPOINT)
         DO 160 J2 = 1, J - 1
            IPOINT = J * (J - 1) / 2 + J2
            CHIOMB = CHIOMB + TWO * WLT(J) * WLT(J2) * SIGINV(IPOINT)
            SIGSUM = SIGSUM + TWO * SIGMA(IPOINT)
  160    CONTINUE
  170 CONTINUE
      NRSTOC = TSUM / ZSQRT(SIGSUM)
      RETURN
      END
```

```
      SUBROUTINE IRRED(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP,
     *                 GROUP, NBI, ISETS, LSETS, LTBP, IOUT, IFAULT)
C
C        ALGORITHM AS 263.1  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Constructs irredundant test sets
C
      INTEGER IFAULT, IOUT, ITBTYP, LSETS, LTBP, NBI, NDIST, NTAXA,
     *        NTESTS
      INTEGER GROUP(NTAXA), ISETS(LSETS)
      LOGICAL LGROUP, LSTAB
      INTEGER I, ITAX1, ITAX2, IT1LIM, JTLIST, KPOWR2, LDBP, NIBP, NB0
C
C        Check parameters
C
      IF ((NTAXA .LE. 0) .OR. (NTESTS .LE. 0) .OR. (NDIST .LE. 0) .OR.
     *    (NBI .LE. 0) .OR. (IOUT .LT. 0)) GO TO 40
      IFAULT = 0
C
C        Set number of integers to store each irredundant test set
C
      NIBP = (NTESTS - 1) / NBI + 1
C
C        Set number of unused bits within each bit pattern
C
      NB0 = NIBP * NBI - NTESTS
C
C        Set up Powers of 2 (2**0,1,2...) in
C        ISETS(KPOWR2...KPOWR2+NBI-1)
C
      KPOWR2 = LSETS - NBI + 1
      ISETS(KPOWR2) = 1
      DO 10 I = KPOWR2 + 1, LSETS
         ISETS(I) = ISETS(I - 1) * 2
   10 CONTINUE
C
C        Initialise taxon indices and limits
C
      ITAX1 = ITBTYP
      IT1LIM = NTAXA
      IF (ITBTYP .GT. 0) THEN
         IF (LGROUP) THEN
C
C        Find 1st taxon of group ITBTYP (to be distinguished from
C        other groups)
C
            DO 20 ITAX1 = 1, NTAXA
               IF (GROUP(ITAX1) .EQ. ITBTYP) GO TO 30
   20       CONTINUE
C
C        No taxa in group
C
            IFAULT = 3
            RETURN
         ELSE
C
C        Test sets to distinguish an individual taxon
C
            IF ((ITBTYP .LT. 0) .OR. (ITBTYP .GT. NTAXA)) GO TO 40
            IT1LIM = ITAX1
         END IF
```

```
      END IF
C
C        Form irredundant sets
C
   30 LDBP = NIBP
C
C        Initialise b.p. (i.e. bit pattern) of essential tests
C
      CALL SETVAL(ISETS(1), 0, NIBP)
      I = ITAX1
      ITAX2 = 0
      CALL IRRDTB(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP, GROUP,
     *            NBI, ISETS, LSETS, IOUT, IFAULT, ITAX1, ITAX2, IT1LIM,
     *            NIBP, NB0, KPOWR2, 0, NIBP, LDBP, JTLIST, KPOWR2 - 1,
     *            .FALSE.)
      IF (IFAULT .NE. 0) RETURN
      ITAX1 = I
      ITAX2 = 0
      CALL IRRMLT(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP, GROUP,
     *            NBI, ISETS, LSETS, LTBP, IOUT, IFAULT, ITAX1, ITAX2,
     *            IT1LIM, NIBP, NB0, KPOWR2, 0, NIBP, LDBP, JTLIST,
     *            KPOWR2 - 1)
      RETURN
   40 IFAULT = 1
      RETURN
      END
      SUBROUTINE IRRDTB(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP,
     *                  GROUP, NBI, ISETS, LSETS, IOUT, IFAULT, ITAX1,
     *                  ITAX2, IT1LIM, NIBP, NB0, KPOWR2, KBPEST, KDBP,
     *                  LDBP, JTLIST, LTLIST, LNEXTE)
C
C        ALGORITHM AS 263.2  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Forms entries of the taxon x taxon table of sets of tests
C        that distinguish each pair of taxa.  LNEXTE is .TRUE. on entry
C        if required to form tests in next non-null entry (by IRRMLT
C        when taxon x taxon table not stored)
C
      INTEGER IFAULT, IOUT, ITAX1, ITAX2, ITBTYP, IT1LIM, JTLIST,
     *        KBPEST, KDBP, KPOWR2, LDBP, LSETS, LTLIST, NBI, NB0,
     *        NDIST, NIBP, NTAXA, NTESTS
      INTEGER GROUP(NTAXA), ISETS(LSETS)
      LOGICAL LGROUP, LNEXTE, LSTAB
      INTEGER IBT, JBT, J1, J2, K1, K2, K3, NTD, NTE
      INTEGER IAND, ICNT, IOR
      LOGICAL LTDIST
      EXTERNAL IAND, ICNT, IOR, LTDIST
C
      IF (ITAX1 .GT. 0) GO TO 20
C
C        Increment ITAX1/2 (scan rows/cols of table)
C
   10 ITAX1 = ITAX1 + 1
      IF (ITAX1 .GT. IT1LIM) RETURN
      IF (LGROUP .AND. (ITBTYP .GT. 0)) THEN
         IF (GROUP(ITAX1) .NE. ITBTYP) GO TO 10
         ITAX2 = 0
      ELSE
C
C        Initialise ITAX2 (to scan across next line of table)
         ITAX2 = ITAX1
```

```
      END IF
C
C        Increment ITAX2
   20 ITAX2 = ITAX2 + 1
      IF (ITAX2 .GT. NTAXA) GO TO 10
      IF (LGROUP) THEN
         IF (GROUP(ITAX1) .EQ. GROUP(ITAX2)) GO TO 20
      ELSE
         IF (ITAX1 .EQ. ITAX2) GO TO 20
      END IF
C
C        Form list of tests that distinguish between ITAX1 and ITAX2
C
      JTLIST = LTLIST + 1
      NTE = 0
      NTD = 0
      DO 30 J1 = 1, NTESTS
C
C        Exit to end of loop if test J1 does not distinguish ITAX1 and
C        ITAX2
C
         IF ( .NOT. LTDIST(J1, ITAX1, ITAX2)) GO TO 30
         NTD = NTD + 1
C
C        If NDIST tests already distinguish ITAX1 and ITAX2, and the
C        aim is merely to form the b.p. of essential tests, take the
C        next pair of taxa
C
         IF ((NTD .GT. NDIST) .AND. ( .NOT. (LNEXTE .OR.
     *       LSTAB))) GO TO 20
C
C        Check whether J1 is an essential test
C
         JBT = (J1 + NB0 - 1) / NBI + 1
         IBT = ISETS(KPOWR2 + JBT * NBI - (J1 + NB0))
         IF (IAND(IBT, ISETS(KBPEST + JBT)) .GT. 0) THEN
C
C        Check whether sufficient essential tests already separate
C        ITAX1/2
C
            NTE = NTE + 1
            IF (NTE .GE. NDIST) GO TO 20
         END IF
C
C        Enter J1 in the list of tests that distinguish ITAX1 and ITAX2
C
         JTLIST = JTLIST - 2
         IF (JTLIST .LE. LDBP) GO TO 80
         ISETS(JTLIST) = JBT
         ISETS(JTLIST + 1) = IBT
   30 CONTINUE
      IF (NTD .LE. NDIST) THEN
C
C        Taxa ITAX1 and ITAX2 distinguished by </= NDIST tests
C
         IF (LNEXTE) GO TO 20
         IF (NTD .EQ. 0) THEN
C
C        Taxa ITAX1 and ITAX2 cannot be distinguished
C
            WRITE (IOUT,'(A,I6,A,I6,A)') ' WARNING: TAXA', ITAX1,
```

```
     *                          ' AND', ITAX2,' CANNOT BE DISTINGUISHED'
            GO TO 20
         END IF
C
C        No more than NDIST tests distinguish ITAX1 and ITAX2
C
         IF (NTD .LT. NDIST) WRITE (IOUT,'(A,I6,A,I6,A,I6,A)')
     *      ' WARNING: TAXA', ITAX1, ' AND', ITAX2,
     *      ' DISTINGUISHED BY ONLY', NTD, ' TEST(S)'
C
C        Add to set of essential tests
C
         DO 40 J1 = JTLIST, LTLIST, 2
            ISETS(KBPEST + ISETS(J1)) = IOR(ISETS(KBPEST + JBT),
     *                                      ISETS(J1 + 1))
   40    CONTINUE
C
C        Delete any sets containing >/= NDIST essential tests
C
         IF (KDBP .GE. LDBP) GO TO 20
         K2 = KDBP
         K3 = LDBP
         LDBP = KDBP
         DO 60 K1 = KDBP, K3, NIBP
            IF (K1 .LT. K3) THEN
               IF (NDIST .EQ. 1) THEN
C
C        Check whether set contains new essential test (IBT/JBT from
C        loop 30)
C
                  IF (IAND(ISETS(K1 + JBT), IBT) .EQ. 0) GO TO 60
               ELSE
C
C        Check whether set contains >/= NDIST essential tests
C
                  NTE = 0
                  DO 50 J2 = K1 + 1, K1 + NIBP
                     NTE = NTE + ICNT(IAND(ISETS(J2),
     *                    ISETS(J2 + KBPEST - K1)))
   50                CONTINUE
                  IF (NTE .LT. NDIST) GO TO 60
               END IF
            END IF
C
C        All sets checked or current set deleted, compact the list
C
            CALL CPSETS(ISETS, LSETS, LDBP, K2, K1 - K2)
            LDBP = LDBP + K1 - K2
            K2 = K1 + NIBP
   60    CONTINUE
      ELSE
C
C        >NDIST tests distinguish ITAX1 and ITAX2
C
         IF (LNEXTE) RETURN
C
C        Form bit pattern for new set at LDBP
C
         IF (LDBP + NIBP .GE. JTLIST) GO TO 80
         CALL SETVAL(ISETS(LDBP + 1), 0, NIBP)
         DO 70 J1 = JTLIST, LTLIST, 2
```

```
            ISETS(LDBP + ISETS(J1)) = ISETS(LDBP + ISETS(J1)) +
     *                                ISETS(J1 + 1)
   70    CONTINUE
C
C        Merge new set into list KDBP...LDBP
C
         J1 = LDBP
         CALL IRRMRG(KDBP, J1, LDBP, NIBP, ISETS, LSETS)
      END IF
      GO TO 20
C
C        Insufficient workspace
C
   80 IFAULT = 2
      RETURN
      END
      SUBROUTINE IRRMLT(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP,
     *                  GROUP, NBI, ISETS, LSETS, LTBP, IOUT, IFAULT,
     *                  ITAX1, ITAX2, IT1LIM, NIBP, NB0, KPOWR2, KBPEST,
     *                  KDBP, LDBP, JTLIST, LTLIST)
C
C        ALGORITHM AS 263.3  APPL.STATIST. (1991), VOL.40, NO.1
C
C        'multiplies' entries in the taxon x taxon table of (sets of)
C        distinguishing tests, thus forming irredundant test sets
C
      INTEGER IFAULT, IOUT, ITAX1, ITAX2, ITBTYP, IT1LIM, JTLIST,
     *        KBPEST, KDBP, KPOWR2, LTBP, LDBP, LSETS, LTLIST, NBI,
     *        NIBP, NB0, NDIST, NTAXA
      INTEGER NTESTS, GROUP(NTAXA), ISETS(LSETS)
      LOGICAL LGROUP, LSTAB
      INTEGER IBT, INTD, JBT, J1, J2, J3, J4, KBPENL, KTBCAB, KTBMBE,
     *        KTBP, KTBP2, K1, K2, K3, K4, LTBCAB, LTBMBE, LTBMRG,
     *        LTBNAB, LTIAB, NTD
      INTEGER IAND, ICNT, IOR
      LOGICAL IRRWSP
      EXTERNAL IAND, ICNT, IOR, IRRWSP
C
C        Initial set contains just the essential tests
C
      KTBP = LDBP
      CALL CPSETS(ISETS, LSETS, KTBP, KBPEST, NIBP)
      LTBP = KTBP + NIBP
C
C        Get the next entry from the taxon x taxon table
C
   10 KTBP2 = KTBP
      IF (LSTAB) THEN
C
C        Generate test list from stored bit pattern
C
         IF (KTBP .LE. KDBP) RETURN
         JTLIST = LTLIST + 1
         KTBP = KTBP - NIBP
         K2 = KPOWR2 + NBI - NB0 - 1
         DO 40 JBT = 1, NIBP
            K1 = ISETS(KTBP + JBT)
            DO 20 J1 = K2, KPOWR2, -1
               IBT = ISETS(J1)
               IF (IAND(K1, IBT) .EQ. 0) GO TO 20
               JTLIST = JTLIST - 2
```

```
                IF (JTLIST .LE. LTBP + NIBP) GO TO 180
                ISETS(JTLIST) = JBT
                ISETS(JTLIST + 1) = IBT
                K1 = K1 - IBT
                IF (K1 .LE. 0) GO TO 30
   20        CONTINUE
   30        K2 = KPOWR2 + NBI - 1
   40     CONTINUE
          ELSE
             CALL IRRDTB(ITBTYP, NTAXA, NTESTS, NDIST, LSTAB, LGROUP, GROUP,
     *               NBI, ISETS, LSETS, IOUT, IFAULT, ITAX1, ITAX2,
     *               IT1LIM, NIBP, NB0, KPOWR2, KBPEST, KDBP, LDBP,
     *               JTLIST, LTLIST, .TRUE.)
             IF ((ITAX1 .GT. IT1LIM) .OR. (IFAULT .NE. 0)) RETURN
          END IF
C
C         Sort sets into (a) pass-cannot-absorb (in KTBP...LTBNAB),
C         (b) pass-can-absorb (in KTBCAB (initially LTBP)...LTBCAB),
C         (c/d) must-be-enlarged (in KTBMBE...LTBMBE); also, if NDIST=1,
C         sort tests into contained-in-can-absorb (JTLIST...LTIAB-1)
C         and not-contained-in-can-absorb (LTIAB...LTLIST)
C
          LTBNAB = KTBP
          KTBCAB = LTBP
          LTBCAB = LTBP
          LTBMBE = JTLIST - 1
C
C         Leave space above LTBMBE for sets with none of the new tests(d)
C
          IF (NDIST .GT. 1) LTBMBE = LTBCAB + (JTLIST - LTBCAB) / 2
          KTBMBE = LTBMBE
          LTIAB = JTLIST
          DO 60 K1 = KTBP2, LTBP - 1, NIBP
C
C            Count number of tests from list already in each set
C
             NTD = 0
             DO 50 J1 = JTLIST, LTLIST, 2
                IF (IAND(ISETS(K1 + ISETS(J1)), ISETS(J1 + 1)) .EQ.
     *               0) GO TO 50
                NTD = NTD + 1
                K3 = J1
   50        CONTINUE
             IF (NDIST .LT. NTD) THEN
C
C               Pass, cannot absorb
C
                CALL CPSETS(ISETS, LSETS, LTBNAB, K1, NIBP)
                LTBNAB = LTBNAB + NIBP
             ELSE
                IF (NDIST .EQ. NTD) THEN
C
C               Pass, can absorb
C
                   LTBCAB = LTBCAB + NIBP
                   IF (LTBCAB .GT. KTBMBE) THEN
                      IF (IRRWSP(LTBCAB, KTBMBE, LTBMBE, JTLIST, ISETS,
     *                     LSETS)) GO TO 180
                   END IF
                   CALL CPSETS(ISETS, LSETS, LTBCAB - NIBP, K1, NIBP)
                   IF ((NDIST .EQ. 1) .AND. (K3 .GE. LTIAB)) THEN
```

```
                  JBT = ISETS(K3)
                  IBT = ISETS(K3 + 1)
                  ISETS(K3) = ISETS(LTIAB)
                  ISETS(K3 + 1) = ISETS(LTIAB + 1)
                  ISETS(LTIAB) = JBT
                  ISETS(LTIAB + 1) = IBT
                  LTIAB = LTIAB + 2
               END IF
            ELSE
C
C        Must be enlarged
C
               IF ((NDIST .EQ. 1) .OR. (NTD .GT. 0)) THEN
                  KTBMBE = KTBMBE - NIBP
                  IF (LTBCAB .GE. KTBMBE) THEN
                     IF (IRRWSP(LTBCAB, KTBMBE, LTBMBE, JTLIST, ISETS,
     *                    LSETS)) GO TO 180
                  END IF
                  CALL CPSETS(ISETS, LSETS, KTBMBE, K1, NIBP)
                  IF (NDIST .EQ. 1) GO TO 60
C
C        (NDIST>1) store number of new tests already in set
C
                  ISETS(KTBMBE) = NTD
                  KTBMBE = KTBMBE - 1
               ELSE
                  LTBMBE = LTBMBE + NIBP + 1
                  IF (LTBMBE .GE. JTLIST) THEN
                     IF (IRRWSP(LTBCAB, KTBMBE, LTBMBE, JTLIST, ISETS,
     *                    LSETS)) GO TO 180
                  END IF
                  CALL CPSETS(ISETS, LSETS, LTBMBE - NIBP, K1, NIBP)
                  ISETS(LTBMBE - NIBP) = 0
               END IF
            END IF
         END IF
   60 CONTINUE
C
C        Copy down pass-can-absorb sets
C
      J3 = LTBCAB - KTBCAB
      CALL CPSETS(ISETS, LSETS, LTBNAB, KTBCAB, J3)
      KTBCAB = LTBNAB
      LTBCAB = KTBCAB + J3
      LTBP = LTBCAB
      IF (LTBMBE .LE. KTBMBE) GO TO 10
C
C        Form and check new b.p.'s from must-be-enlarged list
C
      IF (NDIST .EQ. 1) THEN
         NTD = 0
         K2 = JTLIST
         K3 = LTLIST
         K4 = 2
      ELSE
C
C        Calc number of 'augmenting' patterns (no. combinations of
C        NDIST tests)
C
         J1 = (LTLIST - JTLIST + 1) / 2
         J3 = 1
```

```
            J4 = 1
            DO 70 J2 = 1, NDIST
               J3 = J3 * J2
               J4 = J4 * J1
               J1 = J1 - 1
     70     CONTINUE
C
C           Reserve space for 'augmenting' patterns
C
            KBPENL = JTLIST - 1 - J4 * NIBP / J3
C
C           Copy must-be-enlarged patterns to below 'augmenting' patterns
C
            J1 = LTBMBE - KTBMBE
            KTBMBE = KBPENL - J1
            CALL CPSETS(ISETS, LSETS, KTBMBE, LTBMBE - J1, J1)
            LTBMBE = KTBMBE + J1
C
C           Form 'augmenting' patterns (all combinations of NDIST tests)
C
            K1 = KBPENL
            K4 = LTBP + NDIST
            IF (K4 .GT. KTBMBE) GO TO 180
            K3 = LTBP + 1
            K2 = LTBP
            J1 = JTLIST - 2
     80     K2 = K2 + 1
     90     J1 = J1 + 2
            ISETS(K2) = J1
            IF (K2 .LT. K4) GO TO 80
            CALL SETVAL(ISETS(K1 + 1), 0, NIBP)
            DO 100 J1 = K3, K4
               J2 = ISETS(J1)
               ISETS(K1 + ISETS(J2)) = ISETS(K1 + ISETS(J2)) +
        *                              ISETS(J2 + 1)
    100     CONTINUE
            K1 = K1 + NIBP
    110     J1 = ISETS(K2)
            IF (LTLIST - J1 .GT. (K4 - K2 + 1) * 2) GO TO 90
            K2 = K2 - 1
            IF (K2 .GE. K3) GO TO 110
            K2 = KBPENL
            K3 = K1 - 1
            K4 = NIBP
            INTD = 1
         END IF
C
C           Enlarge b.p.'s (and check for absorption)
C
    120 IF (NDIST .GT. 1) THEN
            KTBMBE = KTBMBE + 1
            NTD = ISETS(KTBMBE)
            LTBMRG = LTBP
            IF ((NTD .EQ. 0) .AND. (INTD .EQ. 1)) THEN
C
C           Sets initially with 0 new tests may be absorbed by sets that
C           had >0
C
               LTBCAB = LTBP
               INTD = 0
            END IF
```

```
      END IF
      DO 170 K1 = K2, K3, K4
C
C        Check space
C
         IF (LTBP + NIBP .GT. KTBMBE) GO TO 180
         CALL CPSETS(ISETS, LSETS, LTBP, KTBMBE, NIBP)
         IF (NDIST .EQ. 1) THEN
C
C        Add 1 test
C
            ISETS(LTBP + ISETS(K1)) = IOR(ISETS(LTBP + ISETS(K1)),
     *                                   ISETS(K1 + 1))
            IF (K1 .GE. LTIAB) GO TO 160
         ELSE
C
C        Add >1 tests
C
            J2 = 0
            DO 130 J1 = 1, NIBP
               J2 = J2 + ICNT(IAND(ISETS(LTBP + J1), ISETS(K1 + J1)))
               ISETS(LTBP + J1) = IOR(ISETS(LTBP + J1), ISETS(K1 + J1))
  130       CONTINUE
C
C        Sets augmented with >NDIST-NTD tests will be absorbed
C
            IF (J2 .LT. NTD) GO TO 170
         END IF
C
C        Check whether absorbed by pass-can-absorb b.p. (b)
C
         DO 150 J2 = KTBCAB, LTBCAB - 1, NIBP
            DO 140 J3 = 1, NIBP
               IF (IAND(ISETS(LTBP + J3), ISETS(J2 + J3)) .NE.
     *             ISETS(J2 + J3)) GO TO 150
  140       CONTINUE
            GO TO 170
  150    CONTINUE
  160    IF (NTD .EQ. 0) THEN
            LTBP = LTBP + NIBP
         ELSE
C
C        Check whether contains/contained in b.p. formed from other
C        sets in (c)
C
            CALL IRRMRG(LTBCAB, LTBMRG, LTBP, NIBP, ISETS, LSETS)
         END IF
  170 CONTINUE
C
C        Next set to be enlarged
C
      KTBMBE = KTBMBE + NIBP
      IF (KTBMBE - LTBMBE) 120, 10, 10
C
C        Insufficient workspace
C
  180 IFAULT = 2
      RETURN
      END
      SUBROUTINE IRRMRG(KBP, LBP, KNBP, NIBP, ISETS, LSETS)
C
```

```
C        ALGORITHM AS 263.4  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Checks new b.p. ISETS(KNBP+1...) vs list in ISETS(KBP+1...LBP):
C        deletes it if iT contains any b.p. in the list; otherwise,
C        deletes b.p.'s that contain it, adjusts LBP & sets KNBP to
C        length of the full list
C
      INTEGER KBP, KNBP, LBP, LSETS, NIBP
      INTEGER ISETS(LSETS)
      INTEGER IBPINT, IBPL, IBPN, INCL, J1, K1, K2, K3
      INTEGER IAND
      EXTERNAL IAND
C
      K2 = KBP
      K3 = KBP
      DO 20 K1 = KBP, LBP - 1, NIBP
C
C        Check whether new set contains/is contained in set at K1
C        (INCL=-/+1)
C
         INCL = 0
         DO 10 J1 = 1, NIBP
            IBPL = ISETS(K1 + J1)
            IBPN = ISETS(KNBP + J1)
            IF (IBPL .EQ. IBPN) GO TO 10
            IBPINT = IAND(IBPL, IBPN)
            IF (IBPINT .EQ. IBPL) THEN
               IF (INCL .GT. 0) GO TO 20
               INCL = -1
            ELSE
               IF ((IBPINT .NE. IBPN) .OR. (INCL .LT. 0)) GO TO 20
               INCL = 1
            END IF
   10    CONTINUE
C
C        Return if new set contains/is same as an already stored set
C
         IF (INCL .LE. 0) RETURN
         CALL CPSETS(ISETS, LSETS, K2, K3, K1 - K3)
C
C        K2 is length of compacted list
         K2 = K2 + K1 - K3
C
C        K3=origin of section to be copied down when next containing
C        set found
         K3 = K1 + NIBP
   20 CONTINUE
      IF (K2 .GT. KBP) THEN
C
C        Copy final section, intermediate b.p.'s (LBP...KNBP) and
C        new b.p.
C
         CALL CPSETS(ISETS, LSETS, K2, K3, KNBP + NIBP - K3)
         LBP = LBP - K3 + K2
         KNBP = K2 + KNBP + NIBP - K3
      ELSE
C
C        No contained/containing sets found
C
         KNBP = KNBP + NIBP
      END IF
```

```
      RETURN
      END
      LOGICAL FUNCTION IRRWSP(KWSP, KUSED, LUSED, LWSP, ISETS, LSETS)
C
C        ALGORITHM AS 263.5  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Reorganises workspace to give room before and after
C        KUSED...LUSED; returns value .TRUE. if insufficient space
C
      INTEGER KUSED, KWSP, LSETS, LUSED, LWSP
      INTEGER ISETS(LSETS)
      INTEGER NUSED
C
      NUSED = LUSED - KUSED
      KUSED = KWSP + (KUSED - KWSP + LWSP - LUSED) / 2
      CALL CPSETS(ISETS, LSETS, KUSED, LUSED - NUSED, NUSED)
      LUSED = KUSED + NUSED
      IRRWSP = (KUSED .LE. KWSP) .OR. (LUSED .GE. LWSP)
      RETURN
      END
      SUBROUTINE CPSETS(IVAL, LVAL, KTO, KFROM, NVAL)
C
C        ALGORITHM AS 263.6  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Copies IVAL(KFROM+1...+NVAL) to IVAL(KTO+1...)
C
      INTEGER KFROM, KTO, LVAL, NVAL
      INTEGER IVAL(LVAL)
      INTEGER FROMTO, INC, ITO, JTO, LTO
C
C        Check whether new location above, same as, or below old
C
      FROMTO = KFROM - KTO
      IF (FROMTO) 10, 50, 20
C
C        Values may overlap, copy in reverse order
C
   10 INC = -1
      JTO = KTO + NVAL
      LTO = KTO + 1
      GO TO 30
C
C        Set/copy values from origin upwards
C
   20 INC = 1
      JTO = KTO + 1
      LTO = KTO + NVAL
C
C        Copy values
C
   30 DO 40 ITO = JTO, LTO, INC
         IVAL(ITO) = IVAL(ITO + FROMTO)
   40 CONTINUE
   50 RETURN
      END
      SUBROUTINE SETVAL(IVAL, ICON, NVAL)
C
C        ALGORITHM AS 263.7  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Sets IVAL(1...NVAL) to ICON
C
```

```
      INTEGER ICON, NVAL
      INTEGER IVAL(NVAL)
      INTEGER I
      DO 10 I = 1, NVAL
         IVAL(I) = ICON
   10 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PRINBP(NBI, NFACT, NBP, NPRINT, IBPS, LIBPS, IOUT,
     *                  IFAULT)
C
C        ALGORITHM AS 264  APPL.STATIST. (1991), VOL.40, NO.1
C
C        Prints bit patterns on unit IOUT
C
      INTEGER IOUT, IFAULT, LIBPS, NBI, NPRINT, NBP, NFACT
      INTEGER IBPS(LIBPS)
      INTEGER JP2IT1, JTLIST, J1, J2, J3, J4, J5, KPOWR2, K1, K2, K3,
     *        K4, K5, LTLIST, N, NB0, NIBP, NMIN, NMAX, NSETP
      INTEGER IAND, ICNT
      EXTERNAL IAND, ICNT
C
      IF ((NFACT .LE. 0) .OR. (NBP .LE. 0)) GO TO 160
C
C        Calculate number of integers to store each bit pattern
C
      NIBP = (NFACT - 1) / NBI + 1
C
C        Set up powers of 2 (2**0,1,2...) in IBPS(KPOWR2...LIBPS)
C
      KPOWR2 = LIBPS - NBI + 1
      IBPS(KPOWR2) = 1
      DO 10 J1 = KPOWR2 + 1, LIBPS
         IBPS(J1) = IBPS(J1 - 1) * 2
   10 CONTINUE
C
C        Address of the power of 2 used to represent the first item
C
      JP2IT1 = KPOWR2 + NFACT - NBI * (NIBP - 1) - 1
      LTLIST = KPOWR2 - 1
      IF (NBP + NBP / NIBP .GE. LTLIST) GO TO 150
      NSETP = 0
      NMIN = NFACT
      NMAX = 1
C
C        Store number of factors in each set in IBPS(NBP+1...K1)
C
      K1 = NBP
      DO 30 K3 = 1, NBP, NIBP
         J2 = 0
         DO 20 J1 = K3, K3 + NIBP - 1
            J2 = J2 + ICNT(IBPS(J1))
   20    CONTINUE
         NMIN = MIN(NMIN, J2)
         NMAX = MAX(NMAX, J2)
         K1 = K1 + 1
         IBPS(K1) = J2
   30 CONTINUE
      IF (NPRINT .LT. 0) NMAX = MIN(NMAX, NMIN - NPRINT - 1)
      IF (NPRINT .GT. 0) NMAX = MIN(NMAX, NPRINT)
C
      DO 140 N = NMIN, NMAX
         JTLIST = LTLIST - N + 1
C
C        Store addresses of sets with N factors in IBPS(K1+1...K2)
C
         K2 = K1
         J1 = NBP
         DO 90 K3 = 1, NBP, NIBP
```

```
            J1 = J1 + 1
            IF (IBPS(J1) .NE. N) GO TO 90
C
C        Place address of set into list, according to
C        lexicographic order
C
            DO 70 J2 = K1 + 1, K2
               J3 = IBPS(J2)
               DO 60 J4 = K3, K3 + NIBP - 1
                  J3 = J3 + 1
                  IF (IBPS(J3) - IBPS(J4)) 40, 60, 70
C
C        Move other sets up and insert current set
C
   40             DO 50 J5 = K2, J2, -1
                     IBPS(J5 + 1) = IBPS(J5)
   50             CONTINUE
                  IBPS(J2) = K3 - 1
                  GO TO 80
   60          CONTINUE
   70       CONTINUE
C
C        Insert set at end of list
C
            IBPS(K2 + 1) = K3 - 1
   80       K2 = K2 + 1
            IF (K2 .GE. JTLIST) GO TO 150
   90    CONTINUE
C
C        Decode each bit pattern, store factors in
C        IBPS(JTLIST...LTLIST), then print
C
         DO 130 J2 = K1 + 1, K2
            K3 = 0
            K4 = JTLIST
            K5 = JP2IT1
            DO 110 J3 = IBPS(J2) + 1, IBPS(J2) + NIBP
               J4 = IBPS(J3)
               DO 100 J5 = K5, KPOWR2, -1
                  K3 = K3 + 1
                  IF (IAND(IBPS(J5), J4) .EQ. 0) GO TO 100
                  IBPS(K4) = K3
                  IF (K4 .EQ. LTLIST) GO TO 120
                  K4 = K4 + 1
  100          CONTINUE
               K5 = LIBPS
  110       CONTINUE
C
C        Print contents of bit pattern (now stored
C        in IBPS(JTLIST...LTLIST))
C
  120       NSETP = NSETP + 1
C
C ***** Lines below can be modified to change the style of output *****
C
            WRITE (IOUT, 9000) NSETP, N, (IBPS(J3), J3=JTLIST, LTLIST)
 9000       FORMAT(1X, I6, ')', I5, '  FACTOR(S):  ',20I4 / (27X, 20I4))
  130    CONTINUE
  140 CONTINUE
      RETURN
C
```

```
      150 IFAULT = IFAULT + 1
      160 IFAULT = IFAULT + 1
          RETURN
          END
```

```
      SUBROUTINE GG1WT(WAIT, DUMM, N, H, CTRL, IFAULT)
C
C        ALGORITHM AS 265.1  APPL.STATIST. (1991), VOL.40, NO.2
C
C        Obtains the distribution of the stationary waiting time for
C        a G/G/1 queue on using the fast Fourier transform algorithm
C
C     Auxiliary routines required: FORRT and REVRT from AS 97.
C
      INTEGER N, IFAULT
      REAL WAIT(N), DUMM(N), H, CTRL(3)
      INTEGER I, J, M, MM
      REAL AN, CA, FOUR, HALF, ONE, PI, Q, SA, TINY, TSI, TSR, TWI,
     *     TWO, TWR, XI, XR, XOLD, XNEW, YR, YI, Z, ZERO
      REAL ARRIVE, SERVE
      EXTERNAL ARRIVE, SERVE
      DATA ZERO, HALF, ONE, TWO, FOUR / 0.0, 0.5, 1.0, 2.0, 4.0 /
      DATA TINY / 1.0E-6 /
C
      PI = FOUR * ATAN(ONE)
      M = N / 2
      MM = M + 1
C
C        Check admissibility of H and N
C
      IFAULT = 1
      IF (H .LE. ZERO) RETURN
      IFAULT = 2
      J = 4
      DO 10 I = 2, 20
         IF (J .EQ. N) GO TO 20
         J = J * 2
   10 CONTINUE
      RETURN
C
C        Find tail transforms of interarrival
C        and service time distributions
C
   20 IFAULT = 3
      XOLD = ONE
      DO 30 I = 1, M
         DUMM(I) = ZERO
         XNEW = ARRIVE((REAL(I) - HALF) * H)
         IF ((XNEW .LT. ZERO) .OR. (XNEW .GT. XOLD)) RETURN
         XOLD = XNEW
         WAIT(I) = XNEW
   30 CONTINUE
      IFAULT = 4
      XOLD = ZERO
      DO 40 I = MM, N
         WAIT(I) = ZERO
         XNEW = SERVE((REAL(N - I) + HALF) * H)
         IF ((XNEW .GT. ONE) .OR. (XNEW .LT. XOLD)) RETURN
         XOLD = XNEW
         DUMM(I) = -XNEW
   40 CONTINUE
      CTRL(1) = WAIT(M)
      CTRL(2) = -DUMM(M + 1)
      CALL TRANSF(WAIT, N)
      CALL TRANSF(DUMM, N)
C
```

```
C        Find transform of normalized harmonic
C        renewal sequence and invert
C
      IFAULT = 5
      Z = WAIT(1) + DUMM(1)
      IF (Z .LE. ZERO) RETURN
      WAIT(1) = -LOG(Z)
      Z = WAIT(MM) + DUMM(MM) - TWO * WAIT(MM) * DUMM(MM)
      IF (Z .LE. ZERO) RETURN
      WAIT(MM) = -LOG(Z)
      DO 50 I = 2, M
         AN = PI * (REAL(I) - ONE) / REAL(M)
         SA = SIN(AN)
         CA = COS(AN)
         TSR = DUMM(I)
         TSI = DUMM(I + M)
         TWR = WAIT(I)
         TWI = WAIT(I + M)
         XR = ONE + (CA - ONE) * TSR - SA * TSI
         XI = (CA - ONE) * TSI + SA * TSR
         YR = TSR + XR * TWR - XI * TWI
         YI = TSI + XR * TWI + XI * TWR
         Z = YR * YR + YI * YI
         IF (Z .LE. TINY) RETURN
         Z = HALF * LOG(Z)
         AN = ATAN2(YI, YR)
         WAIT(I) = -Z
         WAIT(I + M) = -AN
   50 CONTINUE
      CALL REVERT(WAIT, N)
C
C        Change appropriate elements to zero and transform
C
      DO 60 I = 2, M
         WAIT(I) = ZERO
   60 CONTINUE
      CTRL(3) = ABS(WAIT(MM)) + ABS(WAIT(MM + 1)) + ABS(WAIT(MM + 5))
      CALL TRANSF(WAIT, N)
C
C        Find transform of first descending ladder height
C
      WAIT(1) = ONE - EXP(-WAIT(1))
      WAIT(MM) = ONE - EXP(-WAIT(MM))
      DO 70 I = 2, M
         TWR = WAIT(I)
         TWI = WAIT(I + M)
         WAIT(I) = ONE - EXP(-TWR) * COS(TWI)
         WAIT(I + M) = EXP(-TWR) * SIN(TWI)
   70 CONTINUE
C
C        Find transform of associated random walk minimum
C
      Q = ONE - WAIT(1)
      WAIT(1) = ONE
      WAIT(MM) = Q / (ONE - WAIT(MM))
      DO 80 I = 2, M
         TWR = ONE - WAIT(I)
         TWI = WAIT(I + M)
         Z = Q / (TWR * TWR + TWI * TWI)
         WAIT(I) = Z * TWR
         WAIT(I + M) = Z * TWI
```

```
   80 CONTINUE
C
C        Invert and shift
C
         CALL REVERT(WAIT, N)
         DO 90 I = 2, M
            WAIT(I) = WAIT(N - I + 2)
   90 CONTINUE
         DO 100 I = MM + 1, N
            WAIT(I) = ZERO
  100 CONTINUE
         IFAULT = 0
         RETURN
         END

         SUBROUTINE TRANSF(X, N)
C
C        ALGORITHM AS 265.2  APPL.STATIST. (1991), VOL.40, NO.2
C
C        Adapts transform to definition used in algorithm AS 97
C
         INTEGER N
         REAL X(N)
         INTEGER I, MM
         REAL Z
C
         Z = REAL(N)
         MM = N / 2 + 1
         DO 10 I = 1, N
            X(I) = Z * X(I)
   10 CONTINUE
         CALL FORRT(X, N)
         DO 20 I = MM + 1, N
            X(I) = -X(I)
   20 CONTINUE
         RETURN
         END

         SUBROUTINE REVERT(X, N)
C
C        ALGORITHM AS 265.3  APPL.STATIST. (1991), VOL.40, NO.2
C
C        Adapts transform to definition used in algorithm AS 97
C
         INTEGER N
         REAL X(N)
         INTEGER I, MM
         REAL ONE, Z
         DATA ONE / 1.0 /
C
         Z = ONE / REAL(N)
         MM = N / 2 + 1
         DO 10 I = MM + 1, N
            X(I) = -X(I)
   10 CONTINUE
         CALL REVRT(X, N)
         DO 20 I = 1, N
            X(I) = Z * X(I)
   20 CONTINUE
         RETURN
         END
```

```
      SUBROUTINE DIRICH(K, N, X, IX, INIT, ALPHA, RLOGL, V, G, NITER, S,
     *                  EPS, WORK, IFAULT)
C
C        ALGORITHM AS 266  APPL.STATIST. (1991), VOL.40, NO.2
C
C     Auxiliary routines required: ALOGAM (CACM algorithm 291 or AS 245),
C     DIGAMA (AS 103), GAMMAD (AS 239), PPCHI2 (AS 91), TRIGAM (AS 121).
C
      INTEGER K, N, IX, INIT, NITER, IFAULT
      REAL X(IX, K), ALPHA(K), RLOGL, V(K * (K + 1) / 2), G(K), S, EPS,
     *     WORK(2 * K)
      INTEGER I, J, IF1, KK, I2, ITN, MAXIT
      REAL ALOGAM, AN, BETA, CHI2, DIGAMA, GAMMA, GAMMAD, GG, ONE,
     *     PPCHI2, RK, SUM, SUM1, TEMP, TRIGAM, TWO, VARP1, X11, X12,
     *     ZERO
      PARAMETER (GAMMA=0.0001, ZERO=0.0, ONE=1.0, TWO=2.0, MAXIT=100)
      INTRINSIC ABS, LOG, MIN, REAL
      EXTERNAL ALOGAM, DIGAMA, GAMMAD, PPCHI2, TRIGAM
C
C        Check input arguments
C
      AN = REAL(N)
      RK = REAL(K)
      IF (K .LT. 2) THEN
         IFAULT = 1
         RETURN
      ELSE IF (N .LE. K) THEN
         IFAULT = 2
         RETURN
      ELSE IF (IX .LT. N) THEN
         IFAULT = 3
         RETURN
      END IF
C
      IFAULT = 4
      DO 20 I = 1, N
         SUM = ZERO
         NITER = I
         DO 10 J = 1, K
            IF (X(I, J) .LE. ZERO) RETURN
            SUM = SUM + X(I, J)
   10    CONTINUE
         IF (ABS(SUM - ONE) .GE. GAMMA) RETURN
   20 CONTINUE
      IFAULT = 0
C
      IF (INIT .EQ. 1) THEN
C
C        Calculate initial estimates using the method of moments
C
         SUM = ZERO
         DO 40 J = 1, K - 1
            X12 = ZERO
            DO 30 I = 1, N
               X12 = X12 + X(I, J)
   30       CONTINUE
            ALPHA(J) = X12 / AN
            SUM = SUM + ALPHA(J)
   40    CONTINUE
C
         X12 = ZERO
```

```
            DO 50 I = 1, N
               X12 = X12 + X(I, 1) ** 2
      50    CONTINUE
C
            ALPHA(K) = ONE - SUM
            X12 = X12 / AN
            VARP1 = X12 - ALPHA(1) ** 2
C
            X11 = (ALPHA(1) - X12) / VARP1
            DO 60 J = 1, K
               ALPHA(J) = X11 * ALPHA(J)
      60    CONTINUE
C
         END IF
C
         IF (INIT .EQ. 2) THEN
C
C           Calculate initial estimates using Ronning's suggestion
C
            SUM = X(1, 1)
            DO 80 J = 1, K
               DO 70 I = 1, N
                  SUM = MIN(SUM, X(I, J))
      70        CONTINUE
      80    CONTINUE
            DO 90 J = 1, K
               ALPHA(J) = SUM
      90    CONTINUE
C
         END IF
C
C           Check whether any ALPHAs are negative or zero
C
         NITER = 0
         DO 100 J = 1, K
            IF (ALPHA(J) .LE. ZERO) THEN
               IFAULT = 6
               RETURN
            END IF
     100 CONTINUE
C
C        Calculate n * log(G(j)) for j = 1,2,...,k and store in WORK array
C
         DO 120 J = 1, K
            SUM = ZERO
            DO 110 I = 1, N
               SUM = SUM + LOG(X(I, J))
     110    CONTINUE
            WORK(J) = SUM
     120 CONTINUE
C
C           Note that ALOGAM cannot fail
C
         GG = ALOGAM(RK / TWO, IFAULT)
C
C           Call Algorithm AS 91 to compute chi-squared value
C
         CHI2 = PPCHI2(GAMMA, RK, GG, IFAULT)
         IF (IFAULT .GT. 0) THEN
            IFAULT = 5
            RETURN
```

```
      END IF
C
      DO 220 ITN = 1, MAXIT
C
         SUM = ZERO
         DO 130 J = 1, K
            SUM = SUM + ALPHA(J)
  130    CONTINUE
C
C        Note that TRIGAM and DIGAMA cannot fail if the first argument
C        is positive
C
         BETA = TRIGAM(SUM, IFAULT)
         TEMP = DIGAMA(SUM, IFAULT)
         SUM1 = ZERO
         DO 140 J = 1, K
            WORK(K + J) = TRIGAM(ALPHA(J), IFAULT)
            SUM1 = SUM1 + ONE / WORK(K + J)
            G(J) = AN * (TEMP - DIGAMA(ALPHA(J), IFAULT)) + WORK(J)
  140    CONTINUE
         BETA = AN * BETA / (ONE - BETA * SUM1)
C
C        Calculate the lower triangle of the Variance-Covariance matrix
C        (V)
C
         SUM = BETA / (AN * AN)
         DO 160 I = 1, K
            DO 150 J = 1, I
               KK = I * (I - 1) / 2 + J
               V(KK) = SUM / (WORK(K + I) * WORK(K + J))
               IF (I .EQ. J) V(KK) = V(KK) + ONE / (AN * WORK(K + J))
  150       CONTINUE
  160    CONTINUE
C
C        Postmultiply the Variance-Covariance matrix (V) by G and store
C        in the last k elements of WORK
C
         DO 190 I = 1, K
            SUM = ZERO
            I2 = I * (I - 1) / 2
            DO 170 J = 1, I - 1
               SUM = SUM + V(I2 + J) * G(J)
  170       CONTINUE
            DO 180 J = I + 1, K
               SUM = SUM + V(J * (J - 1) / 2 + I) * G(J)
  180       CONTINUE
            WORK(K + I) = SUM + V(I * (I + 1) / 2) * G(I)
  190    CONTINUE
C
C        Update the ALPHAs
C
         NITER = ITN
         DO 200 J = 1, K
            ALPHA(J) = ALPHA(J) + WORK(K + J)
            IF (ALPHA(J) .LE. ZERO) THEN
               IFAULT = 6
               RETURN
            END IF
  200    CONTINUE
C
C        Test for convergence
```

```
C
        S = ZERO
        DO 210 J = 1, K
           S = S + G(J) * WORK(K + J)
  210   CONTINUE
C
        IF (S .LT. CHI2) GO TO 230
C
  220 CONTINUE
C
      IFAULT = 7
C
C        Note that GAMMAD cannot fail
C
  230 EPS = GAMMAD(S / TWO, RK / TWO, IF1)
      RLOGL = ZERO
      SUM = ZERO
      DO 240 J = 1, K
         SUM = SUM + ALPHA(J)
         RLOGL = RLOGL + (ALPHA(J) - ONE) * WORK(J) -
     *           AN * ALOGAM(ALPHA(J), IF1)
  240 CONTINUE
      RLOGL = RLOGL + AN * ALOGAM(SUM, IF1)
C
      END
```

```
      REAL FUNCTION DPROB(DIFF, NDF, NPOPS, MBEST, IFAULT)
C
C        ALGORITHM AS 267.1  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Calculates the lower bound for the probability of correct
C        selection of a subset containing the best populations within
C        the set of NPOPS populations.
C
C     Auxiliary routines required: ALNORM (AS 66) and ALOGAM (CACM
C     algorithm 291 or AS 245).
C
      INTEGER IFAULT, MBEST, NDF, NPOPS
      REAL DIFF
C
      INTEGER I, J, K, JMAX, JMIN, KMAX, KMIN, L, MAXDF
      REAL DFW(2, 10), QW(2, 10), BEST, CLOG, CUTJ, CUTK, DF, DF2,
     *     EHJ, GK, GSTEP, HALF, HJ, HSTEP, ONE, PJ, PK, PROB, PZ1, PZ2,
     *     Q, R1, R2, W0, W2, ZERO
      REAL ALNORM, ALOGAM
      EXTERNAL ALNORM, ALOGAM
C
      DATA CUTK, CUTJ, GSTEP / 0.00003, 0.00001, 0.70 /
      DATA ZERO, HALF, ONE / 0.0, 0.5, 1.0 /
      DATA JMAX, JMIN, KMAX, KMIN, MAXDF / 10, 3, 15, 7, 120 /
C
C        Check input arguments
C
   10 PROB = ZERO
      IFAULT = 0
      IF (DIFF .LE. 0.0) IFAULT = 1
      IF (NDF .LE. 3) IFAULT = 20 + IFAULT
      IF (NPOPS .LT. 2 .OR. NPOPS .GT. 100) IFAULT = 300 + IFAULT
      IF (MBEST .LT. 1 .OR. MBEST .GE. NPOPS) IFAULT = 4000 + IFAULT
      IF (IFAULT .NE. 0) GO TO 100
C
C        Transfer arguments and calculate constants.
C
   20 Q = DIFF
      DF = NDF
      BEST = MBEST
      R1 = NPOPS - MBEST
      R2 = MBEST - 1
      CLOG = LOG(BEST * GSTEP) - 0.918938533
C
      IF (NDF .LE. MAXDF) THEN
         DF2 = DF * HALF
         HSTEP = GSTEP * DF ** (-HALF)
         HJ = ZERO
         CLOG = CLOG + LOG(HSTEP) - ALOGAM(DF2, IFAULT) -
     *          DF2 * (ONE - LOG(DF)) + (ONE - DF2) * 0.693147181
         DO 30 J = 1, JMAX
            HJ = HJ + HSTEP
            EHJ = EXP(HJ)
            QW(1, J) = Q * EHJ
            QW(2, J) = Q / EHJ
            DFW(1, J) = DF * (HALF + HJ - HALF * EHJ * EHJ)
            DFW(2, J) = DF * (HALF - HJ - HALF / EHJ / EHJ)
   30    CONTINUE
      END IF
C
C        Compute integral
```

```
C
   40 DO 90 K = 1, KMAX
         PK = ZERO
         DO 80 L = 1, 2
            IF (L .EQ. 1) THEN
               GK = GSTEP * REAL(K)
            ELSE
               GK = -GSTEP * REAL(K - 1)
            END IF
            W0 = CLOG - GK * GK * HALF
            W2 = ZERO
            PZ1 = ALNORM(-GK - Q, .TRUE.)
            PZ2 = ONE
            IF (MBEST .GT. 1) THEN
               PZ2 = ALNORM(GK, .TRUE.)
               IF (PZ2 .GT. ZERO) W2 = R2 * LOG(PZ2)
            END IF
            IF (PZ1 .GT. ZERO .AND. PZ2 .GT. ZERO) PK = PK +
     *          EXP(W0 + R1 * LOG(PZ1) + W2)
            IF (NDF .LE. MAXDF .AND. PZ2 .GT. ZERO) THEN
   50          DO 70 J = 1, JMAX
                  PJ = ZERO
                  DO 60 I = 1, 2
                     PZ1 = ALNORM(-GK - QW(I, J), .TRUE.)
                     IF (PZ1 .GT. ZERO) PJ = PJ +
     *                                     EXP(W0 + DFW(I, J) +
     *                                     R1 * LOG(PZ1) + W2)
   60             CONTINUE
                  PK = PK + PJ
                  IF ((J .GT. JMIN .OR. K .GT. KMIN) .AND.
     *                PJ .LT. CUTJ) GO TO 80
   70          CONTINUE
            END IF
   80    CONTINUE
         PROB = PROB + PK
         IF (PROB .GT. ONE) THEN
            PROB = ONE
            GO TO 100
         END IF
         IF (K .GT. KMIN .AND. PK .LT. CUTK) GO TO 100
   90 CONTINUE
  100 DPROB = PROB
      RETURN
      END

      REAL FUNCTION DVALU(PROB, NDF, NPOPS, MBEST, IFAULT)
C
C        ALGORITHM AS 267.2  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Given a probability PROB, function DVALU calculates the
C        standardized difference required between the MBEST population
C        means and the remaining means when applying ranking and
C        selection procedures to NPOPS populations.
C
      INTEGER IFAULT, MBEST, NDF, NPOPS
      REAL PROB
      INTEGER J, JMAX
      REAL A, BEST, DF, DCUT, E1, E2, EDIF, ONE, P1, P2, PCUT1, PCUT2,
     *     POPS, Q1, Q2, TWO, ZERO
      REAL DPROB, DVALU0
      EXTERNAL DPROB, DVALU0
```

```
C
      DATA JMAX, ZERO, ONE, TWO, DCUT, PCUT1, PCUT2 / 8, 0.0, 1.0, 2.0,
     *     0.00001, 0.00015, 0.001 /
C
C        Check input arguments.
C
      DVALU = ZERO
      IFAULT = 0
      IF (PROB .LT. 0.80 .OR. PROB .GT. 0.995) IFAULT = 5
      IF (NDF .LE. 3) IFAULT = 20 + IFAULT
      IF (NPOPS .LT. 2 .OR. NPOPS .GT. 100) IFAULT = 300 + IFAULT
      IF (MBEST .LT. 1 .OR. MBEST .GE. NPOPS) IFAULT = 4000 + IFAULT
      IF (IFAULT .NE. 0) GO TO 20
C
C        Obtain initial values.
C
      DF = NDF
      POPS = NPOPS
      BEST = MBEST
      Q2 = DVALU0(PROB, DF, POPS, BEST)
      P2 = DPROB(Q2, NDF, NPOPS, MBEST, IFAULT)
      E2 = P2 - PROB
      A = (ONE - PROB) / (ONE - P2)
      IF (E2 .GT. ZERO) A = PROB / P2
      P1 = PROB - E2 * A
      Q1 = DVALU0(P1, DF, POPS, BEST)
      E1 = DPROB(Q1, NDF, NPOPS, MBEST, IFAULT) - PROB
      DVALU = Q1
C
C        Refine and test approximation.
C
      DO 10 J = 1, JMAX
         IF (ABS(E1) .LT. PCUT1) GO TO 20
         EDIF = E2 - E1
         IF (ABS(EDIF) .LT. DCUT) THEN
            DVALU = (Q1 + Q2) / TWO
            IFAULT = 6
            GO TO 20
         END IF
         DVALU = (Q1 * E2 - Q2 * E1) / EDIF
         IF (ABS(E1) .LT. PCUT2) GO TO 20
         Q2 = Q1
         E2 = E1
         Q1 = DVALU
         E1 = DPROB(Q1, NDF, NPOPS, MBEST, IFAULT) - PROB
   10 CONTINUE
      IFAULT = 6
   20 RETURN
      END
      REAL FUNCTION DVALU0(PROB, DF, POPS, BEST)
C
C        ALGORITHM AS 267.3  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Calculates an initial value for the standardized difference.
C
      REAL ALBEST, ALPOPS, BEST, DF, DFMAX, PROB, POPS, T
      REAL GAUINV
      EXTERNAL GAUINV
      DATA DFMAX / 120.0 /
C
      T = GAUINV(PROB)
```

```
      IF (DF .LT. DFMAX) T = T + (T * T * T + T) / DF / 4.0
      ALBEST = LOG(AMIN1(BEST, POPS - BEST))
      ALPOPS = LOG(POPS)
      DVALU0 = (-0.5185 + 0.7927 / DF + (0.09494 - 0.1824 / DF) * T) * T
      DVALU0 = DVALU0 + 1.193 - 0.05360 * ALPOPS - 0.08017 * ALBEST
      DVALU0 = (1.0 + DVALU0 * ALPOPS * LOG(BEST + 1.7183)) * T
      RETURN
      END
```

```
      SUBROUTINE ALLRQR(R, NR, N, W1, NW1, W2, IW1, NW2, W3, NW3, W4,
     *                  IW2, NW4, IW3, NW5, IORD, IFIN, OUTR, IFAULT)
C
C        ALGORITHM AS 268.1  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Calculates statistics for all possible subset regressions using
C        the R* matrix from a QR decomposition of (X|Y)
C
C        Auxiliary routine required: AS 164 must be called first to form
C        the orthogonal reduction.
C
      INTEGER NR, N, NW1, NW2, IW1(NW2), NW3, IW2(NW4), NW4,
     *       IW3(NW5, 2), NW5, IORD, IFIN, IFAULT
      REAL R(NR), W1(NW1), W2(NW2, 2), W3(NW3), W4(NW4, 3)
      INTEGER I, J, J1, J2, J3, K, L, N1, N2, N3, N4, NB, NC, NF, NI,
     *        NL, NS1, NS2, NS3
      REAL TSS, GC, GS, E1, E2, E3, E4, E5, ONE
      EXTERNAL OUTR
      DATA ONE / 1.E+0 /
C
      IFAULT = 0
C
C        Checks on input arguments
C
      I = N * (N + 1) / 2
      IF (I .NE. NR) THEN
         IFAULT = 1
         RETURN
      END IF
      N1 = N - 1
      IF (N1 .LT. 2) THEN
         IFAULT = 2
         RETURN
      END IF
      N2 = I - N
      IF (N2 .GT. NW3) THEN
         IFAULT = 3
         RETURN
      END IF
      IF (N1 .GT. NW4) THEN
         IFAULT = 4
         RETURN
      END IF
      IF (N .LT. 5) THEN
         J1 = 1
         J2 = 1
         J3 = 1
      ELSE
         J1 = N - 4
         J2 = J1 * (3 + NR / 3)
         J3 = I - 2 * (N + 1)
      END IF
      IF (J1 .GT. NW5) THEN
         IFAULT = 5
         RETURN
      END IF
      IF (J2 .GT. NW1) THEN
         IFAULT = 6
         RETURN
      END IF
      IF (J3 .GT. NW2) THEN
```

```
            IFAULT = 7
            RETURN
         END IF
         IF (IORD .LT. 1 .OR. IORD .GT. 2) THEN
            IFAULT = 8
            RETURN
         END IF
         IF (IFIN .LT. 1 .OR. IFIN .GT. (N1 - 1)) THEN
            IFAULT = 9
            RETURN
         END IF
C
C        Calculate RSS's & TSS from original R* matrix
C
         DO 10 I = 1, N1
            IW2(I) = I
   10    CONTINUE
C*************************************************************************
         TSS = ONE/R(NR)                                          *  1A  *
C*************************************************************************
         CALL OUTR(IW2, TSS, R, N2, R(N2 + 1), N1)
         J = N2
         DO 20 I = N1, 1, -1
C*************************************************************************
          W4(I, 3) = R(N2 + I) * R(N2 + I) / R(J)                 *  2A  *
C*************************************************************************
            TSS = TSS + W4(I, 3)
            J = J - I
            IF (I .GE. IFIN .AND. I .GT. 1) THEN
               CALL OUTR(IW2, TSS, R, J, R(N2 + 1), (I - 1))
            END IF
   20    CONTINUE
         DO 30 I = 2, N1
            W4(I, 3) = W4(I, 3) + W4(I - 1, 3)
   30    CONTINUE
C
C        Proceed dropping each column in turn
C
         IF (IORD .EQ. 1) THEN
            NB = 1
            NI = 1
            NF = N1 - IFIN
            NL = 0
         ELSE
            NB = N1 - IFIN
            NI = -1
            NF = 1
         END IF
   40    NC = NB
         IF (IORD .EQ. 2) THEN
            NL = NC
         END IF
C
C        Initialise temporary storage markers
C
         NS1 = 0
         NS2 = 0
         NS3 = 0
C
C        Start with original matrix.
C
```

```
      DO 50 I = 1, N2
         W3(I) = R(I)
   50 CONTINUE
      DO 60 I = 1, N1
         W4(I, 1) = R(N2 + I)
         W4(I, 2) = W4(I, 3)
         IW2(I) = I
   60 CONTINUE
C
C        Calculate givens rotation factors, perform rotations & fill
C        in elements of work matrix dropping the appropriate column.
C
      N4 = N - NC
   70 N3 = (N4 - 1) * (N4 - 2) / 2
   80 K = N3
      DO 110 L = 1, NC
         J = N4 + L - 2
         K = K + J
         J1 = N4 - NI * NC - NL - 2
C
C        Shift unchanged elements of R matrix
C
         IF (J1 .GE. 1) THEN
            J2 = K - J
            DO 90 I = 1, J1
               W3(J2 + I) = W3(K + I)
   90       CONTINUE
         END IF
C
C        Calculate first diagonal element
C
         J2 = K + J
         GC = W3(K)
         GS = W3(J2 + 1)
         IF (L .EQ. 1) THEN
            E1 = W3(J2)
C*************************************************************************
            E5 = E1 * E1 * GS + GC                                *  3A  *
C*************************************************************************
         ELSE
C*************************************************************************
            E5 = GS + GC                                          *  4A  *
C*************************************************************************
         END IF
C*************************************************************************
         W3(K) = GC * GS / E5                                     *      *
         GC = GC / E5                                             *  5A  *
         GS = GS / E5                                             *      *
C*************************************************************************
         IF (L .EQ. 1) THEN
C*************************************************************************
            GS = E1 * GS                                          *  6A  *
C*************************************************************************
         END IF
C
C        Calculate second diagonal element & first off diagonal element
C
         IF (L .LT. NC) THEN
            J1 = J2 + J + 1
            E3 = W3(J1 + 1)
            IF (L .EQ. 1) THEN
```

```
               E2 = W3(J1)
               E4 = -E2 + E1 * E3
            ELSE
               E2 = W3(J2)
               E4 = -E2 + E3
            END IF
C*************************************************************************
            W3(J2) = E2 * GS + E3 * GC                                *  7A  *
            W3(J2 + 1) = E5 / (E4 * E4)                               *      *
C*************************************************************************
         END IF
C
C        Calculate  elements of rest of R matrix
C
         IF (L .LT. (NC - 1)) THEN
            J2 = K + 2 * J + 1
            DO 100 I = 1, NC - L - 1
               J1 = J2 + J + I + 1
               E3 = W3(J1 + 1)
               IF (L .EQ. 1) THEN
                  E2 = W3(J1)
                  W3(J2 + 1) = (-E2 + E1 * E3) / E4
               ELSE
                  E2 = W3(J2)
                  W3(J2 + 1) = (-E2 + E3) / E4
               END IF
C*************************************************************************
               W3(J2) = E2 * GS + E3 * GC                             *  8A  *
C*************************************************************************
               J2 = J1
  100       CONTINUE
         END IF
C
C        Apply rotation to vector of C's
C
         E2 = W4(J, 1)
         E3 = W4(J + 1, 1)
         IF (L .LT. NC) THEN
            IF (L .EQ. 1) THEN
               W4(J + 1, 1) = (-E2 + E1 * E3) / E4
            ELSE
               W4(J + 1, 1) = (-E2 + E3) / E4
            END IF
         END IF
C*************************************************************************
         W4(J, 1) = E2 * GS + E3 * GC                                 *  9A  *
         W4(J, 2) = W4(J, 1) * W4(J, 1) / W3(K)                       *      *
C*************************************************************************
         IF (J .GT. 1) W4(J, 2) = W4(J, 2) + W4(J - 1, 2)
  110 CONTINUE
C
C        Remove deleted variable from list of variables in model
C
      DO 120 L = N4 - 1, J
         IW2(L) = IW2(L + 1)
  120 CONTINUE
C
C        Print RSS's etc.
C
C*************************************************************************
      CALL OUTR(IW2, (TSS - W4(J, 2)), W3, K, W4(1, 1), J)             *  1B  *
```

```
C*************************************************************************
      IF (NC .GT. 1) THEN
C*************************************************************************
C                                                                    *  2B  *
C*************************************************************************
         J2 = K
         J1 = J + 1
         DO 130 L = 1, NC - 1
            I = J - L
            J2 = J2 - J1 + L
C*************************************************************************
            CALL OUTR(IW2, (TSS - W4(I, 2)), W3, J2, W4(1, 1), I)       *  3B  *
C*************************************************************************
  130    CONTINUE
      END IF
C
C        Storing  required R matrices in temporary storage
C
      IF (NC .LE. 2) GO TO 160
      DO 140 J1 = 1, K
         NS1 = NS1 + 1
         W1(NS1) = W3(J1)
  140 CONTINUE
      DO 150 J1 = 1, J
         NS2 = NS2 + 1
         W2(NS2, 1) = W4(J1, 1)
         W2(NS2, 2) = W4(J1, 2)
         IW1(NS2) = IW2(J1)
  150 CONTINUE
      NS3 = NS3 + 1
      IF (IORD .EQ. 1) THEN
         IW3(NS3, 1) = 2
         N4 = N1 - NS3
         IW3(NS3, 2) = NC
         NC = 2
      ELSE
         IW3(NS3, 1) = NC - 2
         IW3(NS3, 2) = 1
      END IF
  160 NC = NC - 1
      IF (IORD .EQ. 1 .AND. NC .EQ. 1) THEN
         GO TO 70
      END IF
      IF (IORD .EQ. 2 .AND. NC .NE. 0) THEN
         GO TO 80
      END IF
C
C        Working through R matrices in temporary storage
C
  170 IF (NS3 .EQ. 0) GO TO 200
      J = N1 - NS3
      J1 = NS2
      DO 180 L = J, 1, -1
         W4(L, 1) = W2(J1, 1)
         W4(L, 2) = W2(J1, 2)
         IW2(L) = IW1(J1)
         J1 = J1 - 1
  180 CONTINUE
      K = J * (J + 1) / 2
      J2 = NS1
      DO 190 L = K, 1, -1
```

```
            W3(L) = W1(J2)
            J2 = J2 - 1
  190 CONTINUE
         NC = IW3(NS3, 1)
         N4 = N - NS3 - NC
         IF (NC .EQ. IW3(NS3, 2)) THEN
            NS1 = J2
            NS2 = J1
            NS3 = NS3 - 1
            IF (IORD .EQ. 1) THEN
               GO TO 170
            END IF
         ELSE
            IW3(NS3, 1) = IW3(NS3, 1) + NI
         END IF
         GO TO 70
  200 IF (NB .NE. NF) THEN
            NB = NB + NI
            GO TO 40
         END IF
         RETURN
         END
         SUBROUTINE OUTR(IMV, RSS, R, NR, C, NC)
C
C        ALGORITHM AS 268.2  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Version A: For all possible subsets regression
C
         INTEGER NC, NR, IMV(NC)
         REAL R(NR), C(NC), RSS
         WRITE (6, '(A,10(1X,I3))') ' MODEL VARIABLES ', IMV
         WRITE (6, '(A,F20.8)') ' RESIDUAL SUM OF SQUARES ', RSS
         WRITE (6, '(A, I3)') ' R MATRIX IN PACKED VECTOR FORM LENGTH ', NR
         WRITE (6, * ) R
         WRITE (6, '(A, I3)') ' C VECTOR LENGTH ', NC
         WRITE (6, * ) C
         RETURN
         END
         SUBROUTINE OUTR(IMV, RSS, R, NR, C, NC)
C
C        ALGORITHM AS 268.3  APPL.STATIST. (1991), VOL.40, NO.3
C
C        Version B:  Best subsets
C
         INTEGER NR, NC, IMV(NC)
         REAL R(NR), C(NC), RSS
         INTEGER IMVB
         REAL BRSS
         COMMON / BEST / BRSS(20), IMVB(210)
         INTEGER I, J
C
         J = NC * (NC - 1) / 2
         DO 10 I = 1, NC
            IMVB(J + I) = IMV(I)
   10 CONTINUE
         BRSS(NC) = RSS
         RETURN
         END
```

```
      SUBROUTINE CF(NORD, X, AC, A, D, H, P, DEL, IFAULT)
C
C     ALGORITHM AS 269 APPL.STATIST. (1992), VOL.41, NO.1
C
C     Calculates the Cornish-Fisher adjustment to the normal deviate.
C
      INTEGER NORD, IFAULT
      REAL X, AC(NORD), A(NORD), D(NORD), H(3 * NORD),
     *     P(3 * NORD * (NORD+1)/2), DEL(NORD)
C
C     Local variables
C
      INTEGER J, JA, JAL, JB, JBL, K, L
      REAL AA, BC, CC, DD, FAC, LIMIT, ONE, ZERO
      DATA LIMIT, ONE, ZERO / 3.719017274, 1.0, 0.0 /
C
C     Check input arguments
C
      IFAULT = 0
      IF (NORD .GT. 18) THEN
        IFAULT = 1
      ELSE IF (X .LT. -LIMIT .OR. X .GT. LIMIT) THEN
        IFAULT = 2
      END IF
      IF (IFAULT .NE. 0) RETURN
C
C     Compute the adjusted cumulants
C
      CC = -ONE
      DO 10 J = 1, NORD
        A(J) = CC * AC(J) / ((J+1) * (J+2))
        CC = -CC
   10 CONTINUE
C
C     Compute the Hermite polynomial values.
C
      H(1) = -X
      H(2) = X * X - ONE
      DO 20 J = 3, 3 * NORD
        H(J) = - (X * H(J-1) + (J-1) * H(J-2))
   20 CONTINUE
C
C     Clear the polynomial array.
C
      DO 30 J = 1, 3 * NORD * (NORD+1)/2
        P(J) = ZERO
   30 CONTINUE
      D(1) = - A(1) * H(2)
      DEL(1) = D(1)
      P(1) = D(1)
      P(3) = A(1)
      JA = ZERO
      FAC = ONE
C
C     Main loop
C
      DO 70 J = 2, NORD
C
C     Initialize.
C
        FAC = FAC * J
```

```
        JA = JA + 3 * (J-1)
        JB = JA
        BC = ONE
C
C     Calculate coefficients of Hermite polynomials.
C
        DO 50 K = 1, J-1
          DD = BC * D(K)
          AA = BC * A(K)
          JB = JB - 3 * (J - K)
          DO 40 L = 1, 3 * (J - K)
            JBL = JB + L
            JAL = JA + L
            P(JAL+1) = P(JAL+1) + DD * P(JBL)
            P(JAL+K+2) = P(JAL+K+2) + AA * P(JBL)
   40     CONTINUE
          BC = BC * (J - K) / K
   50   CONTINUE
        P(JA+J+2) = P(JA+J+2) + A(J)
C
C     Calculate the adjustments.
C
        D(J) = ZERO
        DO 60 L = 2, 3 * J
          D(J) = D(J) - P(JA+L) * H(L-1)
   60   CONTINUE
        P(JA+1) = D(J)
        DEL(J) = D(J) / FAC
   70 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE CUM(CU, MU, NORD)
C
C     Calculates the cumulants from the moment series by a
C     recursion relation.
C
C     This code was published in Appl. Statist. vol.42 (1993),
C     pp. 268-269.
C
      INTEGER NORD
      REAL CU(NORD), MU(NORD)
C
C     Local variables
C
      INTEGER J, JJ
      REAL ONE, C
      DATA ONE /1.0/
C
      CU(1) = MU(1)
      DO 1 J = 1, NORD - 1
        C = ONE
        CU(J+1) = MU(J+1)
        DO 1 JJ = 0, J-1
          CU(J+1) = CU(J+1) - C * CU(JJ+1) * MU(J-JJ)
          C = C * (J-JJ) / (JJ+1)
    1 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PLYFIT(NN,Y,NINTS,NOBS,DIVBEG,DIVEND,NK,NC,NOTE,
     * AGUESS,BOUNDD,BOUNDU,DBOUND,UBOUND,GROUP,KIND,ITERMX,QUICK,
     * A,SEA,CORR,ALOGL,P,CHISQ,NDF,AINV,AMU,DMUDA,YFN,IFAULT)
C
C        ALGORITHM AS 270.1 APPL.STATIST. (1992), VOL.41, NO.1

C        Subroutine for fitting a parametric 'key' function to data
C        (grouped or ungrouped) and for making polynomial adjustments
C        to that fit.  An iterative maximum likelihood routine, based
C        on Newton-Raphson search, is employed.


C        Stephen T Buckland, Scottish Agricultural Statistics Service,
C        Aberdeen Unit, MLURI, Craigiebuckler, Aberdeen AB9 2QJ, Scotland

C        January, 1991

C        Auxiliary routines required: LUDCMP and LUBKSB from Numerical
C        Recipes by Press, Flannery, Teukolsky and Vetterling, Cambridge
C        University Press, 1986.

C        To change the key function, edit subroutines STD & KEY

C       IFAULT = 0 if convergence achieved
C              = 1 if convergence not achieved after ITERMX steps
C              = 2 if at most one group contains observations
C              = 3 if DIVBEG and/or DIVEND values are invalid
C              = 4 if both NN=0 (grouped data) and GROUP=.FALSE.
C              = 5 if AGUESS = 0.0 or lies outside specified bounds
C                  for any key parameter
C              = 6 if NK<1 or NK>3
C              = 7 if NC<0 or NC>10
C              = 8 if ITERMX < 10
C              = 9 if QUICK=.TRUE. and NC=0
C              = 10 if NOTE contains unexpected zeros or invalid values
C              = 11 if values of NOTE are not ordered from smallest to
C                   largest or if not all values of NOTE are unique
C              = 12 if at least one correlation between parameter
C                   estimates is very close to unity
C              = 13 if fitted density is negative somewhere in the
C                   range (DIVBEG(1),DIVEND(NINTS))

      PARAMETER (KEYMAX=3,NMAX=13,INTMAX=85,NOBMAX=1000,EPS=1.0E-6,
     * JMAX=10)

C     KEYMAX    maximum number of key parameters
C     NMAX      maximum total number of parameters (key parameters and
C               adjustment terms)
C     INTMAX    maximum number of intervals (groups)
C     NOBMAX    maximum number of observations
C     EPS       determines the level of precision of numerical integration
C     JMAX      maximum number of iterations for numerical integration

      REAL DIVBEG(*),DIVEND(*),Y(*),AGUESS(*),DBOUND(*),UBOUND(*),
     * Y2(NOBMAX),YFN(0:50)

C     On a 32 bit machine, change REAL to DOUBLE PRECISION below:

      DOUBLE PRECISION DFNDA(NMAX),A(*),A2(NMAX),A3(NMAX),
     * A4(KEYMAX),ALOGL,ALOGL2,AMU,AINFO(NMAX,NMAX),AINV(NMAX,*),
     * DLLDA(NMAX),DPGDA(INTMAX,NMAX),DMUDA(*),DMUDA2(NMAX),
```

```
      * SEA(*),CORR(NMAX,*),B(NMAX),P(*),ALL,FNEV,DPDA(INTMAX,NMAX),
      * DLFDA(NMAX),DAREA(NMAX),DAREA2(NMAX),ODAREA(NMAX),SUMD(NMAX),
      * BMAT(NMAX,NMAX),BINV(NMAX,NMAX),DKEYDA(KEYMAX),DXSDA(KEYMAX),
      * BB(NMAX,NMAX)
        INTEGER NOBS(*),NOTE(*),INDX(NMAX)
        LOGICAL GROUP,KIND,QUICK,BOUNDD(*),BOUNDU(*)

        DATA SMALL1,SMALL2,P999,BIG1/1.0E-3,1.0E-8,9.99E-1,1.0E10/

C       CALL GETTIM(IHR,IMIN,ISEC,I100TH)
C       TSTRT=(IHR*3600)+(IMIN*60)+ISEC+I100TH/100.0
        IFAULT=0

C       Check for errors in the input parameters

        DO 10 J=1,NINTS
           IF(DIVEND(J).LE.DIVBEG(J)) IFAULT=3

           IF(J.NE.NINTS) THEN
              IF(DIVEND(J+1).LE.DIVEND(J)) IFAULT=3
              IF(DIVBEG(J+1).LE.DIVBEG(J)) IFAULT=3
              IF(DIVEND(J).GT.DIVBEG(J+1)+SMALL2) IFAULT=3
              IF(.NOT.GROUP.AND.DIVEND(J).LT.DIVBEG(J+1)-SMALL2) IFAULT=3
           ENDIF

 10     CONTINUE

        IF(.NOT.GROUP.AND.NN.EQ.0) IFAULT=4
        IF(NK.LT.1.OR.NK.GT.3) IFAULT=6
        IF(NC.LT.0.OR.NC.GT.10) IFAULT=7
        IF(ITERMX.LT.10) IFAULT=8
        IF(NC.EQ.0.AND.QUICK) IFAULT=9

        IF(NC.GT.0) THEN

           DO 20 J=1,NC
              IF(NOTE(J).LE.0.OR.NOTE(J).GT.10) IFAULT=10
              IF(J.GT.1.AND.NOTE(J).LE.NOTE(J-1)) IFAULT=11
 20        CONTINUE

        ENDIF

        DO 30 J=1,NK
           IF(ABS(AGUESS(J)).LT.SMALL2) IFAULT=5
           IF(BOUNDD(J).AND.AGUESS(J).LT.DBOUND(J)) IFAULT=5
           IF(BOUNDU(J).AND.AGUESS(J).GT.UBOUND(J)) IFAULT=5
 30     CONTINUE

        IF(IFAULT.NE.0) RETURN

        IF(NN.GT.0) THEN

C          Calculate group frequencies

           DO 40 J=1,NINTS
 40        NOBS(J)=0

           N=0

           DO 60 I=1,NN
```

```
               IF(Y(I).GT.DIVBEG(1)-SMALL2) THEN

                  DO 50 J=1,NINTS

                     IF(Y(I).LE.DIVEND(J)+SMALL2) THEN
                        N=N+1
                        Y2(N)=Y(I)
                        NOBS(J)=NOBS(J)+1
                        GOTO 60
                     ENDIF

 50                CONTINUE

               ENDIF

 60          CONTINUE

          ELSE
             N=0

             DO 65 J=1,NINTS
 65          N=N+NOBS(J)

          ENDIF

          NZ=0

          DO 70 J=1,NINTS
             IF(NOBS(J).EQ.0) NZ=NZ+1
 70       CONTINUE

          IF(NZ.GE.NINTS-1) THEN
             IFAULT=2
             RETURN
          ENDIF

          NP=NK+NC
          KOUNT=0
          ITER2=0
          ITER3=0
          ICON=0

          IF(GROUP) THEN

C            Calculate the constant part of the log likelihood function

             CONST=0.0
             K=0

             DO 90 J=2,NINTS
                K=K+NOBS(J-1)

                DO 80 I=1,NOBS(J)
 80             CONST=CONST+LOG(FLOAT(I+K))-LOG(FLOAT(I))

 90          CONTINUE

          ENDIF

          DO 110 J=1,NP
```

```
            IF(J.LE.NK) A4(J)=-1.0

            DO 100 K=1,NP
 100        CORR(J,K)=0.0

            B(J)=AGUESS(J)
 110     CONTINUE

         IF(.NOT.QUICK) THEN

C           In MODE 2, the search is conditional on the current key parameter
C           estimates, and only the polynomial coefficients are updated

            MODE=2
         ELSE

C           In MODE 4, only the key parameters are estimated

            MODE=4
         ENDIF

         M=NP
         INIT=0

C        This initialising ensures that erroneous 'convergence'
C        cannot occur at the 1st iteration

         DO 120 J=1,M
 120     A3(J)=-1.0

         ALOGL2=-BIG1

C        If there are no polynomial coefficients to be fitted, only a
C        single set of iterations to fit the key parameters is required

         IF(M.EQ.NK) MODE=3

         IF(QUICK) THEN
            MOLD=M
            M=NK
         ENDIF

         M2=M

 130     ITER=0

 140     ALOGL=ALL(CONST,P,NINTS,DIVBEG,DIVEND,B,DPGDA,AMU,DMUDA,DAREA,
     *   DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,DXSDA,M2,NOTE,NK,N,Y2,NOBS,
     *   KIND,GROUP,INTMAX,EPS,JMAX)

C        If the log likelihood has decreased, enter the Marquardt
C        procedure to find a point at least as good as the previous one

         IF(ALOGL.LT.ALOGL2) THEN
            CALL MARQUA(B,A2,AINFO,DLLDA,NK,MM,MODE,KOUNT,NMAX,BMAT,BINV,
     *      BB,INDX)
            GOTO 230
         ENDIF

         INIT=1
```

```
C      The following subroutine call allows progress in the search to
C      be monitored.  The subroutine is only entered if the likelihood
C      has increased or stayed the same relative to the previous call.
C      If the user wishes to monitor progress, he/she should write a
C      short subroutine MONITR with the arguments listed below that
C      simply prints each argument to a file or to the screen.  Note
C      that, if double precision is used, the first line following the
C      SUBROUTINE statement should be  DOUBLE PRECISION ALOGL,B(*)
C      For single precision, replace DOUBLE PRECISION by REAL

C      ALOGL is the current value of the likelihood
C      MODE is the current mode of search
C      ITER is the number of iterations in the current mode
C      ITER3 is the total number of iterations
C      B(J), J=1,M, is the vector of estimated parameters and
C      polynomial coefficients

       CALL MONITR(ALOGL,MODE,ITER,ITER3,M,B)

C      CALL GETTIM(IHR,IMIN,ISEC,I100TH)
C      TIME=(IHR*3600)+(IMIN*60)+ISEC+I100TH/100.0
C      PRINT *,' Time =',TIME-TSTRT
       ITER=ITER+1
       ITER3=ITER3+1

       IF(ICON.EQ.1) ITER2=ITER2+1
       IF(ITER.EQ.1) GOTO 210

       IF(MODE.EQ.3.OR.QUICK) THEN

C         Has convergence occurred?  If so, GOTO statement 250.
C         If not, return to statement 210

          IF(M.GT.NK) THEN

             DO 150 J=NK+1,M
                IF(ABS(B(J)-A3(J))/(SMALL1+ABS(A3(J))).GT.
     *          SMALL1/SQRT(FLOAT(N))) GOTO 210
 150         CONTINUE

          ENDIF

          IF(MODE.NE.2) THEN

             DO 160 J=1,NK
                IF(ABS(B(J)-A3(J))/(SMALL1+ABS(A3(J))).GT.
     *          SMALL1/SQRT(FLOAT(N))) GOTO 210
 160         CONTINUE

          ENDIF

          GOTO 250
       ENDIF

C      Update parameter estimates

       DO 170 J=1,M
 170   A(J)=B(J)

       IF(MODE.EQ.1) THEN
```

```
C         Search alternates between MODE 1 (in which the key parameters
C         are fitted conditional on the current polynomial coefficient
C         estimates) and MODE 2 (the converse).  If MODE 3 has already
C         been entered once, it is automatically returned to after
C         several cycles, provided convergence seems close.

          IF(ITER2.LE.3.AND.ICON.EQ.1) THEN
             MODE=2
             GOTO 190
          ENDIF

C         If convergence is close, enter MODE 3

          DO 180 J=1,NK

             IF(ABS(B(J)-A4(J))/(SMALL1+ABS(A4(J))).GT.SMALL1) THEN
                MODE=2
                GOTO 190
             ENDIF

 180      CONTINUE

          MODE=3
          ITER=0
          GOTO 210

C         Array A4 is used to test for convergence when the search is
C         conditional on the polynomial coefficient estimates (MODE 1)

 190      DO 200 J=1,NK
 200      A4(J)=B(J)

       ELSEIF(.NOT.QUICK.AND.MODE.EQ.2) THEN

C         After one step in MODE 2, switch to MODE 1

          MODE=1
       ENDIF

 210   IF(ITER3.GT.ITERMX) THEN
          IFAULT=1
          RETURN
       ENDIF

C      If 12 iterations fail to bring convergence in MODE 3, and there is
C      at least one polynomial coefficient to be fitted, revert to
C      alternation between MODEs 1 and 2 for several cycles.

       IF(ITER.GE.12.AND.MODE.EQ.3.AND.NP.GT.NK) THEN
          MODE=2
          ITER2=0
          ICON=1
       ENDIF

C      Initialise variables for Newton-Raphson search

       ALOGL2=ALOGL

       DO 220 J=1,M
 220   A3(J)=B(J)
```

```
        KOUNT=0

C       MM is the number of parameters to be fitted at this stage of the search

        MM=NK
        IF(MODE.EQ.2) MM=M-NK
        IF(MODE.EQ.3) MM=M

        CALL INFO(P,NINTS,B,AINFO,DLLDA,DPGDA,AMU,DMUDA,DPDA,DLFDA,
     * DFNDA,DKEYDA,DXSDA,M,MM,NOTE,NK,N,Y2,NOBS,KIND,GROUP,MODE,
     * NMAX,INTMAX)

        CALL NRAPH(B,A2,AINFO,AINV,DLLDA,NK,MM,MODE,BB,INDX,NMAX)

C       If search exceeds bounds on the parameters, enter Marquardt routine

 230    DO 240 J=1,NK

           IF((BOUNDD(J).AND.B(J).LT.DBOUND(J)).OR.(BOUNDU(J).AND.
     *     B(J).GT.UBOUND(J))) THEN
              CALL MARQUA(B,A2,AINFO,DLLDA,NK,MM,MODE,KOUNT,NMAX,
     *        BMAT,BINV,BB,INDX)
              GOTO 230
           ENDIF

 240    CONTINUE

C       Otherwise, continue to the next iteration of Newton-Raphson

        GOTO 140

C       After convergence has occurred:

 250    IF(MODE.EQ.4) THEN

C          Store the results from fitting the key function, then GOTO
C          statement 130 to fit the polynomial coefficients

           DO 270 K=1,M

              DO 260 L=1,M
 260          CORR(K,L)=AINV(K,L)

              IF(AINV(K,K).LT.0.0) AINV(K,K)=1.0E-20
              SEA(K)=SQRT(AINV(K,K))
 270       CONTINUE

           M=MOLD
           M2=M
           MODE=2

           DO 280 J=1,NK
 280       A(J)=B(J)

           GOTO 130
        ENDIF

        IF(QUICK) THEN
           MINUS=NK
        ELSE
           MINUS=0
```

```
      ENDIF

      CHISQ=0.0

C     Evaluate chi-square goodness-of-fit statistic;  if analysis is of
C     ungrouped data, first evaluate areas under the estimated probabilty
C     density function corresponding to the divisions defined by the
C     cutpoints DIVEND and DIVBEG

      DO 290 J=1,NINTS

         IF(.NOT.GROUP) THEN

            CALL AREAS(P(J),DMUDA2,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,
     *      DXSDA,DIVBEG(J),DIVEND(J),B,M,NOTE,NK,KIND,EPS,JMAX)

            P(J)=P(J)/AMU
         ENDIF

         PJN=P(J)*N
         CHISQ=CHISQ+(NOBS(J)-PJN)**2/ABS(PJN)
 290  CONTINUE

C     Degrees of freedom for chi-square test

      NDF=NINTS-1-M

C     Evaluations of the fitted probability density function at 51
C     equally spaced points;  may be used for plotting

      DO 300 I=0,50
         YVAL=DIVBEG(1)+I*(DIVEND(NINTS)-DIVBEG(1))/50.0
         CALL FNEVAL(YVAL,B,M2,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,DXSDA)
         YFN(I)=FNEV/AMU
         IF(FNEV.LT.0.0) IFAULT=13
 300  CONTINUE

C     Standard errors of the parameter estimates

      DO 310 K=1,M-MINUS
         IF(AINV(K,K).LT.0.0) AINV(K,K)=1.0E-20
         SEA(K+MINUS)=SQRT(AINV(K,K))
 310  CONTINUE

C     Transfer of parameter estimates from array B to A, and corresponding
C     reassignment of the elements of the inverse of the information matrix
C     and of the estimated derivatives of the normalising function (DMUDA)
C     wrt each parameter

      DO 330 K=1,M-MINUS

         DO 320 L=1,M-MINUS
 320     CORR(K+MINUS,L+MINUS)=AINV(K,L)

 330  CONTINUE

      DO 350 K=MINUS+1,M
         A(K)=B(K)

         DO 340 L=MINUS+1,M
 340     AINV(K,L)=CORR(K,L)
```

```
  350   CONTINUE

        IF(MINUS.GT.0) THEN

           DO 370 K=1,MINUS

              DO 360 L=1,MINUS
  360         AINV(K,L)=CORR(K,L)

  370      CONTINUE

           DO 390 K=1,MINUS

              DO 380 L=MINUS+1,M
                 AINV(K,L)=0.0
                 AINV(L,K)=0.0
  380         CONTINUE

  390      CONTINUE

        ENDIF

C       Calculation of correlations between parameter estimates

        IF(MINUS.GT.0) THEN
           CORR(MINUS,MINUS)=1.0

           IF(MINUS.GT.1) THEN

              DO 410 K=1,MINUS-1
                 CORR(K,K)=1.0

                 DO 400 L=K+1,MINUS
                    CORR(K,L)=AINV(K,L)/SEA(K)/SEA(L)
                    CORR(L,K)=CORR(K,L)
                    IF(ABS(CORR(K,L)).GT.P999) IFAULT=12
  400            CONTINUE

  410         CONTINUE

           ENDIF

        ENDIF

        CORR(M,M)=1.0
        IF(M.LE.MINUS+1) RETURN

        DO 430 K=MINUS+1,M-1
           CORR(K,K)=1.0

           DO 420 L=K+1,M
              CORR(K,L)=AINV(K,L)/SEA(K)/SEA(L)
              CORR(L,K)=CORR(K,L)
              IF(ABS(CORR(K,L)).GT.P999) IFAULT=12
  420      CONTINUE

  430   CONTINUE

        RETURN
        END
```

```
C*************************************************************

      SUBROUTINE AREAS(AREA,DAREA,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,
     * DXSDA,DIV1,DIV2,A,M2,NOTE,NK,KIND,EPS,JMAX)

C     Calculation of area under the fitted curve and under the derivative
C     of the fitted curve wrt each parameter between DIV1 and DIV2
C     using Simpson's rule for numerical integration.  The number of
C     intervals is increased until a level of precision, determined
C     by EPS, is achieved.  The method is described by Press et al. (1986)

C     On a 32 bit machine, change REAL to DOUBLE PRECISION below:

      DOUBLE PRECISION AREA,DFNDA(*),A(*),DAREA(*),OAREA,OAREAT,
     * AREA2,DAREA2(*),ODAREA(*),SUM,SUMD(*),FNEV,DKEYDA(*),DXSDA(*)
      INTEGER NOTE(*)
      LOGICAL KIND

      OAREAT=-1.0E30
      OAREA=-1.0E30

      DO 10 K=1,M2
 10   ODAREA(K)=-1.0E30

      DO 100 J=1,JMAX

         IF(J.EQ.1) THEN
            CALL FNEVAL(DIV1,A,M2,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,
     *       DXSDA)
            AREA2=FNEV

            DO 20 K=1,M2
 20         DAREA2(K)=DFNDA(K)

            CALL FNEVAL(DIV2,A,M2,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,
     *       DXSDA)
            CON=0.5*(DIV2-DIV1)
            AREA2=CON*(AREA2+FNEV)

            DO 30 K=1,M2
 30         DAREA2(K)=CON*(DAREA2(K)+DFNDA(K))

            IT=1
         ELSE
            TNM=IT
            DIFF=(DIV2-DIV1)/TNM
            ARG=DIV1+0.5*DIFF
            SUM=0.0

            DO 40 K=1,M2
 40         SUMD(K)=0.0

            DO 60 JJ=1,IT
               CALL FNEVAL(ARG,A,M2,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,
     *          DXSDA)
               SUM=SUM+FNEV

               DO 50 K=1,M2
 50            SUMD(K)=SUMD(K)+DFNDA(K)
```

```
                  ARG=ARG+DIFF
 60            CONTINUE

               AREA2=0.5*(AREA2+(DIV2-DIV1)*SUM/TNM)

               DO 70 K=1,M2
 70            DAREA2(K)=0.5*(DAREA2(K)+(DIV2-DIV1)*SUMD(K)/TNM)

               IT=2*IT
             ENDIF

             AREA=(4.0*AREA2-OAREAT)/3.0

             DO 80 K=1,M2
 80          DAREA(K)=(4.0*DAREA2(K)-ODAREA(K))/3.0

             IF(ABS(AREA-OAREA).LT.EPS*ABS(AREA+OAREA)) RETURN

             OAREA=AREA
             OAREAT=AREA2

             DO 90 K=1,M2
 90          ODAREA(K)=DAREA2(K)

 100   CONTINUE

       RETURN
       END

C*************************************************************

       SUBROUTINE PROBS(P,NINTS,DIVBEG,DIVEND,A,PTOTAL,DPGDA,DMUDA,
     * DAREA,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,DXSDA,M2,NOTE,NK,KIND,
     * INTMAX,EPS,JMAX)

C      Evaluation of the area under each section of the fitted
C      probability density function

       REAL DIVBEG(*),DIVEND(*)

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION A(*),DAREA(*),DMUDA(*),AREA,PTOTAL,P(*),
     * DPGDA(INTMAX,*),DFNDA(*),DAREA2(*),ODAREA(*),SUMD(*),
     * DKEYDA(*),DXSDA(*)
       LOGICAL KIND
       INTEGER NOTE(*)

       PTOTAL=0.0

       DO 10 K=1,M2
 10    DMUDA(K)=0.0

       DO 30 J=1,NINTS

C         Calculation of the area under section J of the fitted curve
C         before normalisation

          DIV2=DIVEND(J)
          DIV1=DIVBEG(J)
```

```
            CALL AREAS(AREA,DAREA,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,
     *      DXSDA,DIV1,DIV2,A,M2,NOTE,NK,KIND,EPS,JMAX)

            P(J)=AREA
            PTOTAL=PTOTAL+P(J)

            DO 20 K=1,M2
               DPGDA(J,K)=DAREA(K)
               DMUDA(K)=DMUDA(K)+DPGDA(J,K)
 20         CONTINUE

 30      CONTINUE

C        PTOTAL is now the value that normalises the fitted curve, to make
C        it a valid density function.  DPGDA is the matrix of estimated
C        derivatives of each of the NINTS non-normalised areas wrt the
C        parameters, and DMUDA is the estimated derivative of the
C        normalising value wrt the parameters.  The following DO loop
C        normalises the P's so that they sum to unity.

         DO 40 J=1,NINTS
            IF(PTOTAL.NE.0.0) P(J)=P(J)/PTOTAL
 40      CONTINUE

         RETURN
         END

C*************************************************************

C        On a 32 bit machine, change REAL to DOUBLE PRECISION below:

         DOUBLE PRECISION FUNCTION ALL(CONST,P,NINTS,DIVBEG,DIVEND,B,
     *   DPGDA,AMU,DMUDA,DAREA,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,DXSDA,M2,
     *   NOTE,NK,N,Y,NOBS,KIND,GROUP,INTMAX,EPS,JMAX)

C        On a 32 bit machine, change REAL to DOUBLE PRECISION below:

         DOUBLE PRECISION P(*),B(*),AMU,DPGDA(INTMAX,*),DMUDA(*),
     *   FNEV,DFNDA(*),DAREA2(*),ODAREA(*),SUMD(*),DKEYDA(*),DXSDA(*),
     *   DAREA(*)
         REAL DIVBEG(*),DIVEND(*),Y(*)
         INTEGER NOTE(*),NOBS(*)
         LOGICAL KIND,GROUP
         DATA SMALL/1.0E-10/

         IF(GROUP) THEN

C           Grouped data:

            CALL PROBS(P,NINTS,DIVBEG,DIVEND,B,AMU,DPGDA,DMUDA,DAREA,
     *      DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,DXSDA,M2,NOTE,NK,KIND,
     *      INTMAX,EPS,JMAX)

            ALL=CONST

C           Add the part of the log likelihood that varies with
C           the parameter estimates to the constant part

            DO 10 J=1,NINTS

C              If an estimated probability < or = 0, set it equal
```

```
C            to a very small value

             IF(P(J).LT.SMALL) P(J)=SMALL
             ALL=ALL+NOBS(J)*LOG(P(J))
 10        CONTINUE

         ELSE

C          Ungrouped data:

           CALL AREAS(AMU,DMUDA,DFNDA,DAREA2,ODAREA,SUMD,DKEYDA,
     *     DXSDA,DIVBEG(1),DIVEND(NINTS),B,M2,NOTE,NK,KIND,EPS,JMAX)

C          Evaluate the log likelihood;  if the estimated fit is
C          impossible, give the log likelihood a large -ve value

           IF(AMU.LE.0.0) AMU=1.0E25
           ALL=-N*LOG(AMU)

           DO 20 J=1,N

              CALL FNEVAL(Y(J),B,M2,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,
     *        DXSDA)

              IF(FNEV.LT.SMALL) FNEV=SMALL
              ALL=ALL+LOG(FNEV)
 20       CONTINUE

         ENDIF

         RETURN
         END

C*************************************************************

         SUBROUTINE INFO(P,NINTS,B,AINFO,DLLDA,DPGDA,AMU,DMUDA,DPDA,
     * DLFDA,DFNDA,DKEYDA,DXSDA,M,MM,NOTE,NK,N,Y,NOBS,KIND,GROUP,MODE,
     * NMAX,INTMAX)

C        On a 32 bit machine, change REAL to DOUBLE PRECISION below:

         DOUBLE PRECISION AMU,AINFO(NMAX,*),DPGDA(INTMAX,*),DMUDA(*),
     * DPDA(INTMAX,*),DLFDA(*),DFNDA(*),P(*),B(*),DLLDA(*),FNEV,
     * DKEYDA(*),DXSDA(*)
         INTEGER NOTE(*),NOBS(*)
         REAL Y(*)
         LOGICAL KIND,GROUP

         DO 20 K=1,M
            DLLDA(K)=0.0

            DO 10 L=1,M
 10         AINFO(K,L)=0.0

 20      CONTINUE

         IF(GROUP) THEN

C          Grouped data:

C          Evaluate derivative wrt each parameter to be fitted of the area
```

```
C         under each section of the probability density function

          DO 40 K=1,M

             IF((MODE.EQ.1.AND.K.LE.NK).OR.(MODE.EQ.2.AND.K.GT.NK).
     *       OR.MODE.GT.2) THEN

                DO 30 J=1,NINTS
                   DPDA(J,K)=(DPGDA(J,K)-P(J)*DMUDA(K))/AMU
 30             CONTINUE

             ENDIF

 40       CONTINUE

C         Evaluate derivative of the log likelihood function
C         wrt each parameter to be fitted

          DO 60 K=1,M

             IF((MODE.EQ.1.AND.K.LE.NK).OR.(MODE.EQ.2.AND.K.GT.NK).
     *       OR.MODE.GT.2) THEN

                DO 50 J=1,NINTS
 50             DLLDA(K)=DLLDA(K)+NOBS(J)*DPDA(J,K)/P(J)

             ENDIF

 60       CONTINUE

C         Calculate the information matrix

          DO 90 K=1,MM
             KK=K
             IF(MODE.EQ.2) KK=K+NK

             DO 80 L=1,MM
                LL=L
                IF(MODE.EQ.2) LL=L+NK

                DO 70 J=1,NINTS
 70             AINFO(K,L)=AINFO(K,L)+N*DPDA(J,KK)*DPDA(J,LL)/P(J)

 80          CONTINUE

 90       CONTINUE

       ELSE

C         Ungrouped data:

          DO 130 J=1,N

C            For each observation, calculate its contribution to the
C            derivatives wrt each parameter to be fitted of the log of
C            the probability density function and of the log likelihood.

             CALL FNEVAL(Y(J),B,M,NOTE,NK,KIND,FNEV,DFNDA,DKEYDA,
     *       DXSDA)

             DO 100 K=1,M
```

```
                IF((MODE.EQ.1.AND.K.LE.NK).OR.(MODE.EQ.2.AND.K.GT.NK).
     *          OR.MODE.GT.2) THEN
                   DLFDA(K)=DFNDA(K)/FNEV -DMUDA(K)/AMU
                   DLLDA(K)=DLLDA(K)+DLFDA(K)
                ENDIF

 100         CONTINUE

C            Calculate the contribution of each observation to the
C            information matrix

             DO 120 K=1,MM
                KK=K
                IF(MODE.EQ.2) KK=K+NK

                DO 110 L=1,MM
                   LL=L
                   IF(MODE.EQ.2) LL=L+NK
                   AINFO(K,L)=AINFO(K,L)+DLFDA(KK)*DLFDA(LL)
 110            CONTINUE

 120         CONTINUE

 130      CONTINUE

       ENDIF

       RETURN
       END

C*************************************************************

       SUBROUTINE NRAPH(B,A2,AINFO,AINV,DLLDA,NK,MM,MODE,BB,INDX,
     * NMAX)

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION B(*),A2(*),AINFO(NMAX,*),AINV(NMAX,*),
     * DLLDA(*),BB(NMAX,*)
       INTEGER INDX(*)

C      Invert the information matrix

       IF(MM.EQ.1) THEN
          AINV(1,1)=1.0/AINFO(1,1)
       ELSE
          CALL INVERS(AINFO,AINV,BB,INDX,MM,NMAX)
       ENDIF

C      For each parameter being fitted, update its estimate, using
C      Newton-Raphson

       DO 20 K=1,MM
          KK=K
          IF(MODE.EQ.2) KK=K+NK
          A2(KK)=B(KK)

          DO 10 L=1,MM
             LL=L
             IF(MODE.EQ.2) LL=L+NK
```

```
            B(KK)=B(KK)+AINV(K,L)*DLLDA(LL)
 10       CONTINUE

 20    CONTINUE

       RETURN
       END

C*************************************************************

       SUBROUTINE MARQUA(B,A2,AINFO,DLLDA,NK,MM,MODE,KOUNT,NMAX,
     * BMAT,BINV,BB,INDX)

C      The Marquardt procedure, for when Newton-Raphson fails to increase
C      the value of the log likelihood.  The procedure is very similar to
C      Newton-Raphson, but progressively adds larger values to the
C      diagonal elements of the information matrix until a point with a
C      larger log likelihood is found.

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION B(*),A2(*),AINFO(NMAX,*),DLLDA(*),
     * BMAT(NMAX,*),BINV(NMAX,*),BB(NMAX,*)
       INTEGER INDX(*)

       KOUNT=KOUNT+1

       DO 20 K=1,MM

          DO 10 L=1,MM
             BMAT(K,L)=AINFO(K,L)
             IF(K.EQ.L) BMAT(K,L)=BMAT(K,L)*(1.0+2.0**KOUNT)
 10       CONTINUE

 20    CONTINUE

       IF(MM.EQ.1) THEN
          BINV(1,1)=1.0/BMAT(1,1)
       ELSE
          CALL INVERS(BMAT,BINV,BB,INDX,MM,NMAX)
       ENDIF

       DO 40 K=1,MM
          KK=K
          IF(MODE.EQ.2) KK=K+NK
          B(KK)=A2(KK)

          DO 30 L=1,MM
             LL=L
             IF(MODE.EQ.2) LL=L+NK
             B(KK)=B(KK)+BINV(K,L)*DLLDA(LL)
 30       CONTINUE

 40    CONTINUE

       RETURN
       END

C*************************************************************

       SUBROUTINE POLY(XS,H)
```

```
C      Simple polynomials up to order 10

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION H(0:10),XS

       H(0)=1.0
       H(1)=XS
       H(2)=XS*XS
       H(3)=XS**3
       H(4)=XS**4
       H(5)=XS**5
       H(6)=XS**6
       H(7)=XS**7
       H(8)=XS**8
       H(9)=XS**9
       H(10)=XS**10

       RETURN
       END

C*************************************************************

       SUBROUTINE HERM(XS,H)

C      Hermite polynomials up to order 10

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION H(0:10),XS

       H(0)=1.0
       H(1)=XS
       H(2)=XS*XS-1.0
       H(3)=XS**3-3.0*XS
       H(4)=XS**4-6.0*XS*XS+3.0
       H(5)=XS**5-10.0*XS**3+15.0*XS
       H(6)=XS**6-15.0*XS**4+45.0*XS*XS-15.0
       H(7)=XS**7-21.0*XS**5+105.0*XS**3-105.0*XS
       H(8)=XS**8-28.0*XS**6+210.0*XS**4-420.0*XS*XS+105.0
       H(9)=XS**9-36.0*XS**7+378.0*XS**5-1260.0*XS**3+945.0*XS
       H(10)=XS**10-45.0*XS**8+630.0*XS**6-3150.0*XS**4+4725.0*XS*XS
     * -945.0

       RETURN
       END

C*************************************************************

       SUBROUTINE FNEVAL(X,A,M2,NOTE,NK,KIND,FNEV,DFNDA,
     * DKEYDA,DXSDA)

C      Evaluates the function and its derivative wrt A (before
C      normalisation) at point X, using parameter estimates A

C      On a 32 bit machine, change REAL to DOUBLE PRECISION below:

       DOUBLE PRECISION A(*),DFNDA(*),DKEYDA(*),XS,DXSDA(*),
     * VALKEY,H(0:10),FNEV,TOP,DTOP
       INTEGER NOTE(*)
```

```
      LOGICAL KIND

      CALL STD(X,A,XS,DXSDA)

      TOP=1.0

C     VALKEY is the value of the key function

      CALL KEY(X,A,VALKEY,DKEYDA)

      IF(M2.GT.NK) THEN

         IF(KIND) THEN
            CALL HERM(XS,H)
         ELSE
            CALL POLY(XS,H)
         ENDIF

C        TOP is the contribution of the polynomial terms to the fitted
C        density, apart from the normalising value, which is a function
C        of the parameters alone

         DO 10 J=NK+1,M2
 10      TOP=TOP+A(J)*H(NOTE(J-NK))

      ENDIF

      FNEV=VALKEY*TOP

C     Evaluate the derivative of the curve (before normalisation) at point
C     X, wrt each parameter, using parameter estimates A

      TOP=1.0
      DTOP=0.0

      IF(M2.GT.NK) THEN

         DO 20 J=NK+1,M2
 20      DFNDA(J)=VALKEY*H(NOTE(J-NK))

         DO 30 J=NK+1,M2
            TOP=TOP+A(J)*H(NOTE(J-NK))
            DTOP=DTOP+A(J)*(NOTE(J-NK))*H(NOTE(J-NK)-1)
 30      CONTINUE

      ENDIF

      DO 40 J=1,NK
 40   DFNDA(J)=VALKEY*DTOP*DXSDA(J)+DKEYDA(J)*TOP

      RETURN
      END

C*************************************************************

      SUBROUTINE INVERS(A,Y,B,INDX,N,NMAX)

C     Calculates the inverse of a matrix.
C     See 'Numerical Recipes', Press et al., 1986, p38.

C     On a 32 bit machine, change REAL to DOUBLE PRECISION below:
```

```
      DOUBLE PRECISION A,B,Y
      DIMENSION A(NMAX,*),B(NMAX,*),Y(NMAX,*),INDX(*)

      DO 20 I=1,N

         DO 10 J=1,N
            B(I,J)=A(I,J)
            Y(I,J)=0.0
10       CONTINUE

         Y(I,I)=1.0
20    CONTINUE

      CALL LUDCMP(B,N,NMAX,INDX,D)

      DO 30 J=1,N
         CALL LUBKSB(B,N,NMAX,INDX,Y(1,J))
30    CONTINUE

      RETURN
      END

C*************************************************

      SUBROUTINE STD(X,A,XS,DXSDA)

C     Reduces observations to standard measure, to avoid numerical
C     problems, and differentiates the standardised observation wrt A

C     On a 32 bit machine, change REAL to DOUBLE PRECISION below:

      DOUBLE PRECISION A(*),XS,DXSDA(*)

      XS=(X-A(1))/A(2)
      DXSDA(1)=-1.0/A(2)
      DXSDA(2)=-(X-A(1))/(A(2)**2)

      RETURN
      END

C*************************************************

      SUBROUTINE KEY(X,A,VALKEY,DKEYDA)

C     Evaluates the fitted key function and its derivative wrt A at X

C     On a 32 bit machine, change REAL to DOUBLE PRECISION below:

      DOUBLE PRECISION A(*),VALKEY,DKEYDA(*)

      DATA SMALL2,SMALL3/1.0E-8,1.0E-10/

      VALKEY=1.0
      D=0.0
      IF(ABS(X-A(1)).GT.SMALL3) D=((X-A(1))/A(2))**2/2.0

      IF(D.GT.20.0) THEN
         VALKEY=0.0
      ELSEIF(D.GT.SMALL2) THEN
         VALKEY=EXP(-D)
```

```
      ENDIF

C     Estimate the differential at X of the key function wrt A

      DKEYDA(1)=(X-A(1))*VALKEY/(A(2)**2)
      DKEYDA(2)=((X-A(1))**2)*VALKEY/(A(2)**3)

      RETURN
      END
```

```
      SUBROUTINE MCE (N, KMAX, A1, PQ, PMIN, LKA, UPMQM, ICR, IFAULT)
C
C        ALGORITHM AS 271 APPL.STATIST. (1992), VOL.41, NO.1
C
C        Obtains the optimal (smallest misclassification probability)
C        joint classification procedure combining at most KMAX of N
C        independent marginal classification rules.
C
      INTEGER N, KMAX, LKA(N  + 2), ICR(2, N), IFAULT
      REAL A1, PQ(2, N), PMIN, UPMQM(3, N * (N + 1) / 2)
C
      INTRINSIC ABS, MIN
C
      INTEGER HELP, HOLD, I, I0, I2, II, J, J0, JI, JJ, K, L, LL, M,
     *        MM, N1, N2, UPPER, R
      REAL A2, ERP, ERQ, NEGONE, ONE, PSTAR, SIGN, STORE, ZERO
C
      PARAMETER (ZERO = 0.0E0, ONE = 1.0E0 , NEGONE = -1.0E0)
C
C        Check for errors in the input variables
C
      IFAULT = 0
      IF (N .LT. 2) IFAULT = 1
      IF (A1 .LE. ZERO .OR. A1 .GE. ONE) IFAULT = IFAULT + 2
      DO 10 I = 2, N
         IF (PQ(1, I - 1) .GT. PQ(1, I)) THEN
            IFAULT = IFAULT + 4
            GOTO 20
         ENDIF
   10 CONTINUE
   20 DO 30 I = 1, N
         IF (PQ(2, I) .LT. ZERO .OR. PQ(2, I) .GT. ONE) THEN
            IFAULT = IFAULT + 8
            GOTO 40
         ENDIF
   30 CONTINUE
   40 IF (PQ(1, 1) .LT. ZERO .OR. PQ(1, N) .GT. ONE)
     *   IFAULT = IFAULT + 16
      IF (KMAX .LT. 1 .OR. KMAX .GT. N) IFAULT = IFAULT + 32
      IF (IFAULT .GT. 0) RETURN
C
C         Initialization step
C
      A2 = ONE - A1
      PMIN = ONE
      R = 1
      ICR(2, 1) = 1
      DO 50 I = 2, N
         IF (PQ(2, I - 1) .GT. PQ(2, I)) R = R + 1
         ICR(2, I) = R
   50 CONTINUE
      UPMQM(1, 1) = ONE
      UPMQM(1, 2) = NEGONE
      UPMQM(1, 3) = ONE
      I0 = 2
      II = 4
      DO 70 J = 3, N
         UPMQM(1, II) = -UPMQM(1, I0)
         SIGN = UPMQM(1, I0)
         II = II + 1
         DO 60 I = 3, J
```

```
               UPMQM(1, II) = SIGN * (ABS(UPMQM(1, I0)) +
     *                        ABS(UPMQM(1, I0 + 1)))
               SIGN = -SIGN
               I0 = I0 + 1
               II = II + 1
      60    CONTINUE
            UPMQM(1, II) = ONE
            I0 = I0 + 1
            II = II + 1
      70 CONTINUE
C
C           Start with the full candidate problem (1, 2, ..., N)
C
            K = N
            MM = N
            DO 80 I = 1, N
               ICR(1, I) = I
      80 CONTINUE
            N1 = N + 1
            N2 = N + 2
C
C           Optimality check
C
      90 M = K - MM
            HOLD = M + 1
            HELP = M + MM
            UPPER = MIN(HELP, KMAX)
            IF (HOLD .EQ. 1) THEN
               UPMQM(2, 1) = PQ(1, ICR(1, 1))
               UPMQM(3, 1) = PQ(2, ICR(1, 1))
               PSTAR  = A1 * UPMQM(2, 1) + A2 * UPMQM(3, 1)
               IF (PSTAR .LT. PMIN) THEN
                  PMIN = PSTAR
                  LKA(N1) = 1
                  LKA(N2) = 1
                  LKA(1) = ICR(1, 1)
               ENDIF
               HOLD = 2
            ENDIF
            IF (HELP .NE. 1) THEN
               II = HOLD * (HOLD - 1) / 2 + 1
               I0 = II - HOLD + 1
               DO 110 J = HOLD, UPPER
                  L = ICR(1, J)
                  UPMQM(2, II) = UPMQM(2, I0) + PQ(1, L)
                  UPMQM(3, II) = UPMQM(3, I0) + PQ(2, L)
                  II = II + 1
                  DO 100 I = 3, J
                     UPMQM(2, II) = UPMQM(2, I0 + 1) +
     *                           UPMQM(2, I0) * PQ(1, L)
                     UPMQM(3, II) = UPMQM(3, I0 + 1) +
     *                           UPMQM(3, I0) * PQ(2, L)
                     II = II + 1
                     I0 = I0 + 1
     100          CONTINUE
                  UPMQM(2, II) = UPMQM(2, I0) * PQ(1, L)
                  UPMQM(3, II) = UPMQM(3, I0) * PQ(2, L)
                  II = II + 1
                  I0 = I0 + 1
     110    CONTINUE
            ENDIF
```

```
          J0 = HOLD * (HOLD - 1) / 2
          DO 160 J = HOLD, UPPER
             I0 = 1
             DO 150 I = 1, J
                ERP = ZERO
                ERQ = ZERO
                II = I0
                JI = J0 + I
                DO 120 L = I, J
                   ERQ = ERQ + UPMQM(1, II) * UPMQM(3, JI)
                   II = II + L
                   JI = JI + 1
    120         CONTINUE
                LL = J - I + 1
                II = LL * (LL + 1) / 2
                JI = J0 + LL
                DO 130 L = LL, J
                   ERP = ERP + UPMQM(1, II) * UPMQM(2, JI)
                   II = II + L
                   JI = JI + 1
    130         CONTINUE
                I0 = I0 + I + 1
                PSTAR = A1 * ERP + A2 * ERQ
                IF (PSTAR .LT. PMIN .OR.
       *           (PSTAR .EQ. PMIN .AND. LKA(N2) .GT. J)) THEN
                   PMIN = PSTAR
                   LKA(N2) = J
                   LKA(N1) = I
                   DO 140 L = 1, J
                      LKA(L) = ICR(1, L)
    140            CONTINUE
                ENDIF
    150 CONTINUE
          J0 = J0 + J
    160 CONTINUE
          MM = 0
C
C           Backward step
C
    170 IF (ICR(2, ICR(1, 1)) .EQ. R) RETURN
          L = ICR(2, ICR(1, K))
    180 K = K - 1
          IF (K .GT. 0) THEN
             IF (ICR(2, ICR(1, K)) .GE. L) GOTO 180
          ENDIF
          IF (K .GT. 0 .AND. L .EQ. R) THEN
            L = ICR(2, ICR(1, K))
            K = K - 1
          ENDIF
          L = L + 1
          K = K + 1
          HELP = ICR(1, K) + 1
          DO 190 I = HELP, N
             IF (ICR(2, I) .EQ. L) GOTO 200
    190 CONTINUE
    200 ICR(1, K) = I
C
C           Fathoming step
C
    210 JJ = ICR(1, K) - 1
          J = 1
```

```
      STORE = PQ(2, ICR(1, K))
      DO 230 I = 1, JJ
         IF (ICR(1, J) .GT. I) THEN
            IF (PQ(2, I) .LE. STORE) THEN
              IF (MM .EQ. 0) THEN
                 GOTO 170
              ELSE
C
C        Side step
C
                 HOLD = ICR(2, ICR(1, K))
                 K = K - 1
                 IF (HOLD .EQ. R) GOTO 90
                 K = K + 1
                 HELP = ICR(1, K) + 1
                 DO 220 I2 = HELP, N
                    IF (ICR(2, I2) .NE. HOLD) THEN
                       ICR(1, K) = I2
                       GOTO 210
                    ENDIF
 220             CONTINUE
              ENDIF
            ENDIF
            ELSE
               J = J + 1
        ENDIF
  230 CONTINUE
C
C        Forward step
C
      MM = MM + 1
      IF (ICR(1, K) .EQ. N) GOTO 90
      IF (ICR(1, K) .LT. N) THEN
          K = K + 1
          ICR(1, K) = ICR(1, K - 1) + 1
          GOTO 210
      ENDIF
      RETURN
      END
```

```
      SUBROUTINE BXPLT(NG, RVALS, IVALS, SCALEX, SCALEY, NY, NX, XTYPE,
     *   WIDTH, NOTCH, IWRITE, OUTARR, INTARR, IVALS2, IFAULT)
C
C     ALGORITHM AS 272.1 APPL.STATIST. (1992), VOL.41, NO.1
C
C     PRODUCES A BOX-PLOT USING SUPPLIED QUARTILES
C
      REAL RVALS(6, NG), SCALEX(2), SCALEY(2), OUTARR(*), VALS(20)
      INTEGER IVALS(6, NG), WIDTH, WID, NX, IVALS2(8, NG),
     *   INTARR(*), POINTS, PNTR
      CHARACTER INTCH(0:9), COLON, DASH, DOT
      CHARACTER*19 IFORM1
      CHARACTER*20 IFORM2
      CHARACTER*120 IOUT
      LOGICAL LWIDOK, NOTCH, XTYPE
      EXTERNAL AXIS, SCALE
C
      DATA MPVX / 10 /, MPVY / 5 /, MAXHT / 66 /, MAXWID / 130 /
      DATA IFORM1 /'(1H ,F8.0,1X,120A1)' /
      DATA IFORM2 /'(1H ,5X,16(F8.0,2X))' /
      DATA COLON / ':' / , DASH / '-' / ,  DOT / '.' /
      DATA INTCH / '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' /
C
C CHECK PARAMETERS
C
      IFAULT = 0
      IF (NG .LT. 1) GOTO 900
      XMIN = SCALEX(1)
      XMAX = SCALEX(2)
      YMIN = SCALEY(1)
      YMAX = SCALEY(2)
C
C CHECK IF NLX NLY ARE IN LEGAL RANGE
C
      NLX = NX
      NLY = NY
      IF (NLY + 5 .GT. MAXHT) NLY = MAXHT - 5
      IF (NLY .LT. MPVY) NLY = MPVY
      IF (NLX + 11 .GT. MAXWID) NLX = MAXWID - 11
      IF (NLX .LT. MPVX) NLX = MPVX
      IF (XTYPE) NLX = (NG + 1) * (NLX / (NG + 1))
      NOUT = 0
      DO  10 I = 1, NG
         NOUT = NOUT + IVALS(1, I) + IVALS(2, I)
   10 CONTINUE
      WID = WIDTH
      IF (WIDTH .EQ. 0) WID = NLX / (2 * NG + 1) + 1
C
C CALCULATE WIDTH EQUAL MARK SPACE RATIO
C
      IF (XTYPE) THEN
C
C QUALITATIVE X SIMULATE CALL  OF SCALE
C
         XMIN = 0.0
         XSTEP = 1.0 / (NLX / (NG + 1))
C
C MUST CALCULATE SCALE, FIRST CHOOSE XMIN AND XMAX
C
      ELSE IF (XMIN .GE. XMAX) THEN
         XMIN = RVALS(1, 1)
```

```
            XMAX = XMIN
            DO 20 I = 2, NG
            IF (RVALS(1, I) .LT. XMIN) THEN
               XMIN = RVALS(1, I)
            ELSE IF (RVALS(1, I) .GT. XMAX) THEN
               XMAX = RVALS(1, I)
            ENDIF
      20    CONTINUE
            XMN = XMIN
            XMX = XMAX
      30    CALL SCALE(XMN, XMX, NLX, MPVX, TEMP, XVSTEP, NXVALS, IRX,
        *         IFAIL)
            IF (IFAIL .GT. 0) GOTO 901
            XSTEP = XVSTEP / MPVX
            IF ((XMIN - TEMP) / XSTEP .LT. WID) THEN
C
C OFFPLOT ON LEFT
C
               XMN = XMN - XSTEP
               GOTO  30
            ENDIF
            IF ((XMAX - TEMP) / XSTEP .GT. (NLX - WID)) THEN
C
C OFFPLOT ON RIGHT
C
               XMX = XMX + XSTEP
               GOTO  30
            ENDIF
            XMIN = TEMP
         ELSE
            CALL SCALE(XMIN, XMAX, NLX, MPVX, TEMP, XVSTEP, NXVALS, IRX,
        *         IFAIL)
            IF (IFAIL .GT. 0) GOTO 901
            XMIN = TEMP
            XSTEP = XVSTEP / MPVX
         ENDIF
         IF (YMIN .GE. YMAX) THEN
C
C CHOOSE YMIN AND YMAX
C
            YMAX = RVALS(2, 1)
            YMIN = RVALS(3, 1)
            DO 40 I = 2, NG
            IF (RVALS(2, I) .GT. YMAX) THEN
               YMAX = RVALS(2, I)
            ELSE IF (RVALS(3, I) .LT. YMIN) THEN
               YMIN = RVALS(3, I)
            ENDIF
      40    CONTINUE
C
C CHECK IN OUTLIER ARRAY, ADJUST YMIN AND YMAX
C
            DO 50 I = 1, NOUT
            IF (OUTARR(I) .GT. YMAX) THEN
               YMAX = OUTARR(I)
            ELSE IF (OUTARR(I) .LT. YMIN) THEN
               YMIN = OUTARR(I)
            ENDIF
      50    CONTINUE
         ENDIF
         CALL SCALE(YMIN, YMAX, NLY, MPVY, TEMP, YVSTEP, NYVALS, IRY,
```

```
     * IFAIL)
      IF (IFAIL .GT. 0) GOTO 902
      YMIN = TEMP
      YSTEP = YVSTEP / MPVY
C
C CONVERT TO PLOT UNITS
C
      DO 70 I = 1, NG
      DO 60 J = 2, 6
      IVALS2(J, I) = (RVALS(J, I) - YMIN) / YSTEP + 0.5
   60 CONTINUE
C
C NOTCHES AT MEDIAN +- 1.57 * IQR/SQRT(N)
C
      IF(NOTCH .AND. IVALS(5, I) .GT. 0) THEN
         QR = (RVALS(4, I) - RVALS(5, I)) * 1.57/SQRT(REAL(IVALS(5, I)))
         YDIFF = RVALS(6, I) - YMIN
         IVALS2(7, I) = (YDIFF + QR)/YSTEP + 0.5
         IVALS2(8, I) = (YDIFF - QR)/YSTEP + 0.5
      ENDIF
   70 CONTINUE
      DO 80 I = 1, NG
         IF(XTYPE) THEN
            IVALS2(1, I) = (I - XMIN) / XSTEP + 1.5
         ELSE
            IVALS2(1, I) = (RVALS(1, I) - XMIN) / XSTEP + 1.5
         ENDIF
   80 CONTINUE
      DO 90 I = 1, NOUT
      INTARR(I) = (OUTARR(I) - YMIN) / YSTEP + 0.5
   90 CONTINUE
C
C CHECK OVERPRINTING ONLY IF WIDTH CALCULATED
C
      IF (WIDTH .EQ. 0) THEN
  100    IF (WID .GT. 1) THEN
            LWIDOK = (IVALS2(1, 1) .GE. WID / 2 + 1 .AND.
     *                IVALS2(1, NG) .LE. NLX - (WID / 2 + 1))
            IF (.NOT. LWIDOK) GOTO 903
            DO 110 I = 2, NG
C
C CHECK GAPS BETWEEN BOXES
C
            IF (IVALS2(1, I) - IVALS2(1, I - 1) .LT. WID)
     *          LWIDOK = .FALSE.
  110       CONTINUE
            IF (LWIDOK) GOTO 120
            WID = WID - 2
            GOTO 100
         ENDIF
  120    CONTINUE
C
C LIMIT MUST PROVIDE ROOM FOR NOTCHES
C
         LIMIT = 3
         IF (NOTCH) LIMIT = 5
         IF (WID .LT. LIMIT) GOTO 903
      ENDIF
      DO 130 I = 1, NG
      IF (WIDTH .GE. 0) IVALS(6, I) = WID
C
```

```
C WIDTH MUST BE ODD EVEN IF EVEN VALUE SUPPLIED
C
      IF (MOD(IVALS(6, I), 2) .EQ. 0) IVALS(6, I) = IVALS(6, I) + 1
      IF (IVALS(6, I) .LT. 1) IVALS(6, I) = 1
  130 CONTINUE
      CALL AXIS(YMIN, YVSTEP, NYVALS, 6, IRY, IRPR, OFFSET, IFACT,
     *       VALS, 20, IFAIL)
      IF (IFAIL .GT. 0) GOTO 904
      IFORM1(9:9) = INTCH(IRPR)
      IF (IFACT .NE. 0) WRITE(IWRITE, 2) IFACT
      IF (OFFSET .NE. 0.0) WRITE(IWRITE, 3) OFFSET
      DO 170 I = 1, NLY
C
C MAIN LOOP TO BUILD PLOT STARTS HERE
C
      IY = NLY - I
      IOUT(1:NLX) = ' '
C
C FIND CORRECT CODE
C
      DO 160 J = 1, NG
      IF (IVALS(5, J) .GE. 1) THEN
         IX = IVALS2(1, J)
         ICODE = 0
         IF (IY .LE. IVALS2(2, J) .AND. IY .GE. IVALS2(3, J)) ICODE = 1
         POINTS = 0
         IF (IVALS(1, J) .GT. 0) THEN
            PNTR = IVALS(3, J)
            DO 140 K = 1, IVALS(1, J)
            IF (INTARR(PNTR) .EQ. IY) POINTS = POINTS + 1
            PNTR = PNTR + 1
  140       CONTINUE
         ENDIF
         IF (IVALS(2, J) .GT. 0) THEN
            PNTR = IVALS(4, J)
            DO 150 K = 1, IVALS(2, J)
            IF (INTARR(PNTR) .EQ. IY) POINTS = POINTS + 1
            PNTR = PNTR + 1
  150       CONTINUE
         ENDIF
         IF (POINTS .EQ. 1) ICODE = 2
         IF (POINTS .GT. 1) ICODE = 3
         IF (IY .EQ. IVALS2(4, J) .OR. IY .EQ. IVALS2(5, J)) ICODE = 4
         IF (IY .LT. IVALS2(4, J) .AND. IY .GT. IVALS2(5, J)) THEN
            ICODE = 5
            IF (NOTCH) THEN
               IF ((IY .LE. IVALS2(7, J) .AND. IY .GE. IVALS2(8, J)))
     *            ICODE = 6
            ENDIF
         ENDIF
         IF (IY .EQ. IVALS2(6, J)) ICODE = 7
         CALL SETGR(IOUT, IVALS(6, J), ICODE, IX, NOTCH, NLX, IFAIL)
         IF (IFAIL .EQ. 1) IFAULT =  - 1
      ENDIF
C
C UNABLE TO FIT ON PLOT
C
  160 CONTINUE
      IF (MOD(IY, MPVY) .EQ. 0) THEN
         WRITE(IWRITE, IFORM1) VALS(NYVALS), DASH, COLON,
     *                         (IOUT(IX:IX), IX = 1, NLX)
```

```
            NYVALS = NYVALS - 1
         ELSE
            WRITE(IWRITE, 1)(IOUT(IX:IX), IX = 1, NLX)
         ENDIF
  170 CONTINUE
         WRITE(IWRITE, 1)(DOT, I = 1, NLX)
         IF ( .NOT. XTYPE) THEN
            CALL AXIS(XMIN, XVSTEP, NXVALS, 6, IRX, IRPR, OFFSET, IFACT,
     *              VALS, 20, IFAIL)
            IF (IFAIL .GT. 0) GOTO 905
            IFORM2(15:15) = INTCH(IRPR)
            WRITE(IWRITE, 6)(COLON, I = 1, NXVALS)
            WRITE(IWRITE, IFORM2)(VALS(I), I = 1, NXVALS)
            IF (IFACT .NE. 0) WRITE(IWRITE, 4) IFACT
            IF (OFFSET .NE. 0.0) WRITE(IWRITE, 5) OFFSET
         ENDIF
C
C CHECK POINTS AND PARTS OF BOXES LYING BEYOND Y LIMITS
C
         IFAIL = 0
         DO 180 I = 1, NG
         IF (IVALS2(2, I) .GT. NLY - 1 .OR. IVALS2(3, I) .LT. 0) GOTO 190
  180 CONTINUE
         GOTO 200
  190 IFAULT = IFAULT - 2
  200 DO 210 I = 1, NOUT
         IF (INTARR(I) .GT. NLY - 1 .OR. INTARR(I) .LT. 0) GOTO 220
  210 CONTINUE
         RETURN
  220 IFAULT = IFAULT - 4
         RETURN
1        FORMAT(11X, ':', 120A1)
2        FORMAT(' TIMES 10**', I3)
3        FORMAT(' OFFSET', F10.0)
4        FORMAT(15X, 'TIMES 10**', I3)
5        FORMAT(15X, 'OFFSET', F10.0)
6        FORMAT(3X, 16(9X, A1))
900      IFAULT = 1
         RETURN
901      IFAULT = 2
         RETURN
902      IFAULT = 3
         RETURN
903      IFAULT = 4
         RETURN
904      IFAULT = 5
         RETURN
905      IFAULT = 6
         RETURN
         END
C
C
         SUBROUTINE SETGR(IOUT, WID, ICODE, IX, NOTCH, NLX, IFAIL)
C
C        ALGORITHM AS 272.2 APPL.STATIST. (1992), VOL.41, NO.1
C
C PLACES CHARACTERS INTO IOUT AT POSITION IX
C CURRENT PART OF BOX CODED BY ICODE
C
         INTEGER WID, ICODE, IX, IFAIL
         CHARACTER VERT, STAR, PLUS, HORIZ, DASH
```

```
      CHARACTER*120 IOUT
      LOGICAL NOTCH
      DATA VERT / ':' / , STAR / 'X' / , PLUS / '+' / , HORIZ / '-' / ,
     * DASH / '=' /
      IFAIL = 1
      IF (IX .LT. 1 .OR. IX .GT. NLX) RETURN
      IFAIL = 0
      IW = WID / 2
      IF (ICODE .EQ. 0) RETURN
      IF (ICODE .EQ. 1) IOUT(IX:IX) = VERT
      IF (ICODE .EQ. 2) IOUT(IX:IX) = STAR
      IF (ICODE .EQ. 3) IOUT(IX:IX) = PLUS
      IF (ICODE .LT. 4) RETURN
      IXMIN = MAX(IX - IW, 1)
      IXPLUS = MIN(IX + IW, NLX)
      IF (ICODE .EQ. 4) THEN
         DO 10 I = IXMIN, IXPLUS
         IOUT(I:I) = HORIZ
   10    CONTINUE
         RETURN
      ENDIF
      IF (ICODE .LT. 7) THEN
C
C IF WID LT 3 NO SIDES OF BOX OR NOTCHES
C
         IF (IW .LE. 0) RETURN
         IF (ICODE .EQ. 5) THEN
            IF (IX - IW .GE. 1) IOUT(IXMIN:IXMIN) = VERT
            IF (IX + IW .LE. NLX) IOUT(IXPLUS:IXPLUS) = VERT
         ENDIF
         IF (ICODE .EQ. 6) THEN
C
C IF ROOM MOVE SIDES OF BOX IN TO FORM NOTCH
C
            IF (WID .GT. 3) IXPLUS = IXPLUS - 1
            IF (IX + IW .LE. NLX) IOUT(IXPLUS:IXPLUS) = VERT
            IF (WID .GT. 3) IXMIN = IXMIN + 1
            IF (IX - IW .GE. 1) IOUT(IXMIN:IXMIN) = VERT
         ENDIF
         RETURN
      ENDIF
      IF (NOTCH .AND. WID .GT. 3) THEN
C
C MEDIAN SHORTER IF NOTCHED
C
         IXPLUS = IXPLUS - 1
         IXMIN = IXMIN + 1
      ENDIF
      DO 20 I = IXMIN, IXPLUS
      IOUT(I:I) = DASH
   20 CONTINUE
      IF (IW .GT. 0) THEN
C
C IF WID LT 3 DO NOT PUT SIDES OF BOX AT MEDIAN
C
         IF (IX - IW .GE. 1) IOUT(IXMIN:IXMIN) = VERT
         IF (IX + IW .LE. NLX) IOUT(IXPLUS:IXPLUS) = VERT
      ENDIF
      RETURN
      END
```

Notes on AS 273 "Comparing subsets of regressor variables" by Alan Miller

This algorithm allows two regression subsets to be compared. It uses the method of Spj0tvoll (Ann. Math. Statist. vol.43, 1076-1088, 1972). The algorithm will typically be used after the use of some subset selection procedure, to make comparisons between subsets. It is a Scheffe-type procedure which gives a confidence level which applies simultaneously to all pairs of subsets from the same data set. The subsets may be of different numbers of variables, and there is no requirement that some or all of the variables in one subset must be in the other. For more details of the algorithm see chapter 4 of `Subset selection in regression' by Alan Miller, Chapman & Hall, 1990 (ISBN 0 412 35380 6).

The algorithm calculates upper and lower confidence limits for the difference in regression sums of squares for the two subsets. If both limits are positive then the first subset has a significantly larger regression sum of squares than the first. If the range includes zero, the subsets do not differ significantly.

The user must enter F-values for the confidence levels required. The number of degrees of freedom for the numerator is equal to the total number of variables less any (such as the optional constant) forced into all subsets. The number of degrees of freedom for the denominator is equal to the number of degrees of freedom of the residual variance, or NCASES - NP.

Before calling this algorithm, the user must have first performed an orthogonal reduction of the data using Gentleman's algorithm (AS 75 or 274), or an equivalent algorithm.

The calling arguments are (real/dp means real or double precision):

NP	Input,	integer	No. of variables in the orthogonal reduction
NRBAR	Input,	integer	No. of elements in array RBAR (at least NP(NP-1)/2).
D	Input,	real/dp array	Row multipliers from AS 75 or AS 274
RBAR	Input,	real/dp array	Upper triangle of Cholesky factor without the diagonal elements (which are implicit 1's)
THETAB	Input,	real/dp array	The least-squares projections of the dependent variable on the NP orthogonal directions
RSS	Input,	real/dp array	RSS(I) is the residual sum of squares with the first I variables in the model (output from SS of AS 274)
TOL	Input,	real/dp array	Array of tolerances from TOLSET in AS 274
NCASES	Input,	integer	The number of cases on which the orthogonal reduction was based
VORDER	Input,	integer array	An array of integer identifiers, one per variable
NFORCE	Input,	integer	The number of variables which were forced into all subsets in the subset selection procedure (often = 1 for the constant term in all models)
LIST1	Input,	integer array	Array of identifying integers (see VORDER above) for the variables in the first subset to be compared. Include variables which are forced in.
N1	Input,	integer	The number of variables in LIST1.
LIST2	Input,	integer array	The list of variables in the second subset.
N2	Input,	integer	The number of variables in the second subset.
FVAL	Input,	integer array	Array of F-values for the required confidence levels.
NF	Input,	integer	The number of confidence levels required, usually only 1 or 2.
A1	Output,	real/dp	Array of lower confidence limits corresponding to the

```
              array            input F values.
A2     Output, real/dp  Array of the corresponding upper confidence limits.
              array
WK     --, real/dp      Workspace, must be dimensioned at least 2N(N+1) where
              array     N = the number of variables which are in one subset
                        but not both.
DIMWK  Input, integer   The dimension of WK in the calling program.
IWK    --, integer      Workspace, must be dimensioned at least N.
              array
DIMIWK Input, integer   The dimension of IWK in the calling program.
IFAULT Output, integer  Error indicator
                        = 0, no error detected
                        = -NEED, insufficient workspace provided for WK,
                                 increase to NEED.
                        = 1  NP < 2
                        = 2  NRBAR < NP(NP-1)/2
                        = 4  NCASES <= NP
                        = 8  either N1 or N2 exceeds NP
                        = 16 either N1 or N2 is negative
                        = 32 NF < 1
                        = 64 either NFORCE < 0 or NFORCE >= NP
                        = 128 LIST1 = LIST2, i.e. identical subsets
                        = 129 error in both lists, e.g. a repeated number or
                             a number which is not in array VORDER
                        = 130 as for error 129 but only in LIST1
                        = 131 as for error 129 but only in LIST2
                        = 132 inadequate integer work space

Auxiliary routines required: TRED2 and TQL2 from EISPACK, or AS 60 to calculate
the eigenvalues and vectors for a symmetric matrix.   REORDR and VMOVE from
algorithm AS 274.

Alan Miller
25 November 1991

C------------------------------------------------------------------------

      SUBROUTINE CONREG(NP, NRBAR, D, RBAR, THETAB, RSS, TOL, NCASES,
     +    VORDER, NFORCE, LIST1, N1, LIST2, N2, FVAL, NF, A1, A2, WK,
     +    DIMWK, IWK, DIMIWK, IFAULT)
C
C     ALGORITHM AS273  APPL. STATIST. (1992) VOL. 41, NO. 2
C
C     Calculate the Spj0tvoll confidence limits for the difference in
C     regression sums of squares of two subsets of variables.
C
C     Auxiliary routines required: TRED2, TQL2 (from EISPACK or AS 60),
C     and REORDR, VMOVE (from AS 274).
C
      INTEGER NP, NRBAR, NCASES, VORDER(NP), NFORCE, N1, LIST1(N1), N2,
     +    LIST2(N2), NF, DIMWK, DIMIWK, IWK(DIMIWK), IER
      DOUBLE PRECISION D(NP), RBAR(NRBAR), THETAB(NP), RSS(NP), TOL(NP),
     +    FVAL(NF), A1(NF), A2(NF), WK(DIMWK)
C
C     Local variables
C
      INTEGER N0, I, L, J, K, N10, N20, N10N20, NEED, POS2, INC, POS1,
     +    ROW, COL, ID, ITHETA, IRSS, ITOL, ROWPN0, ROWL1, NVAR, IZ,
     +    M, IVAL, IMAX, IMIN
      DOUBLE PRECISION ZERO, ONE, HALF, SCALE, TEMP, SUM, DIFF, SUME,
     +    EMAX, EMIN, VAR, TEMPS, X, X1, X2, XTOL, A, B, GAMMA2
```

```
C
      DATA ZERO/0.D0/, ONE/1.D0/, HALF/0.5D0/, XTOL/0.001D0/
C
C     Some checks.
C
      IER = 0
      IF (NP .LT. 2) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (NCASES .LE. NP) IER = IER + 4
      IF (N1 .GT. NP .OR. N2 .GT. NP) IER = IER + 8
      IF (N1 .LT. 0 .OR. N2 .LT. 0) IER = IER + 16
      IF (NF .LT. 1) IER = IER + 32
      IF (NFORCE .LT. 0 .OR. NFORCE .GE. NP) IER = IER + 64
      IF (IER .NE. 0) RETURN
C
C     Find which variable numbers are on both lists and move them to
C     the start.
C
      N0 = 0
      IF (N1 .EQ. 0 .OR. N2 .EQ. 0) GO TO 70
      DO 60 I = 1, N1
        L = LIST1(I)
        DO 10 J = N0+1, N2
          IF (L .EQ. LIST2(J)) GO TO 20
   10   CONTINUE
        GO TO 60
   20   N0 = N0 + 1
        IF (I .EQ. N0) GO TO 40
        K = I
   30   LIST1(K) = LIST1(K-1)
        K = K - 1
        IF (K .GT. N0) GO TO 30
        LIST1(N0) = L
C
   40   IF (J .EQ. N0) GO TO 60
        K = J
   50   LIST2(K) = LIST2(K-1)
        K = K - 1
        IF (K .GT. N0) GO TO 50
        LIST2(N0) = L
   60 CONTINUE
C
C     Exit if LIST1 = LIST2.
C
   70 N10 = N1 - N0
      N20 = N2 - N0
      N10N20 = N10 + N20
      IF (N10N20 .EQ. 0) THEN
        IER = 128
        RETURN
      END IF
C
C     Re-order the orthogonal reduction so that the N0 variables common
C     to both lists are at the start.
C
      IF (N0 .GT. 0) CALL REORDR(NP, NRBAR, VORDER, D, RBAR, THETAB,
     +         RSS, TOL, LIST1, N0, 1, IER)
      IF (IER .NE. 0) THEN
        IER = 129
        RETURN
      END IF
```

```
C
C     Follow with the rest of the variables, if any, in LIST1, then any
C     any remaining from LIST2.
C
      IF (N10 .GT. 0) CALL REORDR(NP, NRBAR, VORDER, D, RBAR, THETAB,
     +         RSS, TOL, LIST1(N0+1), N10, N0+1, IER)
      IF (IER .NE. 0) THEN
        IER = 130
        RETURN
      END IF
      IF (N20 .GT. 0) CALL REORDR(NP, NRBAR, VORDER, D, RBAR, THETAB,
     +         RSS, TOL, LIST2(N0+1), N20, N1+1, IER)
      IF (IER .NE. 0) THEN
        IER = 131
        RETURN
      END IF
C
C     Estimate residual variance.
C
      VAR = RSS(NP) / (NCASES - NP)
C
C     If N10 > 0 and N20 > 0, calculate the product of planar rotations
C     needed to place the N10 variables from LIST1 after the N20 from
C     LIST2 instead of before them.
C
      IF (N10 .EQ. 0 .OR. N20 .EQ. 0) GO TO 400
C
C     Check that adequate workspace has been provided.
C
      NEED = 2 * N10N20 * (N10N20 + 1)
      IF (DIMWK .LT. NEED) THEN
        IER = -NEED
        RETURN
      END IF
      IF (DIMIWK .LT. N10N20) THEN
        IER = 132
        RETURN
      END IF
C
C     Copy the triangle of RBAR for the N10N20 rows and columns of
C     interest into WK with an N10N20 x N10N20 identity matrix added
C     as additional columns to store the product.   Thus the contents
C     of WK look like (for N10N20 = 4):
C              X  X  X  1  0  0  0
C                 X  X  0  1  0  0
C                    X  0  0  1  0
C                       0  0  0  1
C
C     As row multipliers are used by the square-root free method, the
C     reciprocals of the square roots of these multipliers are stored
C     on the diagonal instead of 1's.
C     POS2 = position within RBAR; INC = number of columns of RBAR to
C     be skipped in each row.
C
      POS2 = N0 * (NP + NP - N0 - 1)/2 + 1
      INC = NP - N0 - N10N20
      POS1 = 1
      DO 100 ROW = 1, N10N20
        DO 80 COL = ROW+1, N10N20
          WK(POS1) = RBAR(POS2)
          POS1 = POS1 + 1
```

```
            POS2 = POS2 + 1
     80   CONTINUE
          POS2 = POS2 + INC
          DO 90 COL = 1, N10N20
            IF (COL .EQ. ROW) THEN
              WK(POS1) = ONE / SQRT(D(ROW+N0))
            ELSE
              WK(POS1) = ZERO
            END IF
            POS1 = POS1 + 1
     90   CONTINUE
    100 CONTINUE
C
C     Set up dummy arrays in WK for D, THETAB, RSS and TOL, and in IWK
C     for VORDER.
C
      ID = POS1
      ITHETA = ID + N10N20
      IRSS = ITHETA + N10N20
      ITOL = IRSS + N10N20
      DO 110 ROW = 1, N10N20
        ROWPN0 = ROW + N0
        ROWL1 = ROW - 1
        WK(ID + ROWL1) = D(ROWPN0)
        WK(ITHETA + ROWL1) = ZERO
        WK(IRSS + ROWL1) = RSS(ROWPN0)
        WK(ITOL + ROWL1) = TOL(ROWPN0)
        IWK(ROW) = VORDER(ROWPN0)
    110 CONTINUE
C
C     Swap positions of the N10 and N20 variables in WK.
C
      NVAR = 2 * N10N20
      DO 120 ROW = N10, 1, -1
    120 CALL VMOVE(NVAR, DIMWK, IWK, WK(ID), WK, WK(ITHETA), WK(IRSS),
     +    ROW, N10N20+ROW-N10, WK(ITOL), IER)
C
C     Scale the first N20 rows of the product and place at the start of
C     WK.   This is (P1, P2).
C
      POS2 = N10N20
      POS1 = 1
      DO 140 ROW = 1, N20
        SCALE = SQRT(WK(ID))
        ID = ID + 1
        DO 130 COL = 1, N10N20
          WK(POS1) = SCALE * WK(POS2)
          POS1 = POS1 + 1
          POS2 = POS2 + 1
    130   CONTINUE
          POS2 = POS2 + N10N20 - ROW - 1
    140 CONTINUE
C
C     Construct the matrix, Z, in WK from position IZ.
C          Z  =  ( I  0 )  -  ( P1' ) ( P1  P2 )
C                ( 0  0 )    ( P2' )
C
      IZ = N10N20**2 + 1
      POS1 = IZ
      POS2 = IZ
      DO 170 ROW = 1, N10N20
```

```
            TEMP = ZERO
            IF (ROW .LE. N10) TEMP = ONE
            DO 160 COL = ROW, N10N20
              L = ROW
              J = COL
              DO 150 I = 1, N20
                TEMP = TEMP - WK(L)*WK(J)
                L = L + N10N20
                J = J + N10N20
    150       CONTINUE
              WK(POS1) = TEMP
              WK(POS2) = TEMP
              POS1 = POS1 + 1
              POS2 = POS2 + N10N20
              TEMP = ZERO
    160     CONTINUE
            POS1 = POS1 + ROW
            POS2 = POS1
    170   CONTINUE
C
C       Put a copy of Z at the start of WK.
C
        POS2 = IZ
        DO 180 I = 1, IZ-1
          WK(I) = WK(POS2)
          POS2 = POS2 + 1
    180   CONTINUE
C
C       Calculate the eigenvalues (stored from location IVAL in WK), and
C       the normalized eigenvectors (stored from location IZ).
C
        IVAL = 2*IZ - 1
        L = IVAL + N10N20**2
        CALL TRED2(N10N20, N10N20, WK, WK(IVAL), WK(L), WK(IZ))
        CALL TQL2(N10N20, N10N20, WK(IVAL), WK(L), WK(IZ), IER)
C
C       Calculate the transformed projections (Spj0tvoll's gammas) and
C       store at the start of WK.   Put the sum of squares of gamma(i).
C       lambda(i) into SUME.
C
        L = IZ
        SUME = ZERO
        DO 250 I = 1, N10N20
          M = N0 + 1
          SUM = ZERO
          DO 240 J = 1, N10N20
            SUM = SUM + WK(L)*THETAB(M)*SQRT(D(M))
            L = L + 1
            M = M + 1
    240     CONTINUE
            WK(I) = SUM
            SUME = SUME + (SUM*WK(IVAL+I-1))**2
    250   CONTINUE
C
C       Find the max. and min. non-zero eigenvalues.
C
        EMAX = WK(IVAL)
        EMIN = EMAX
        IMAX = 1
        IMIN = 1
        L = IVAL + 1
```

```
      DO 260 I = 2, N10N20
        TEMP = WK(L)
        L = L + 1
        IF (TEMP .GT. EMAX) THEN
          EMAX = TEMP
          IMAX = I
        ELSE IF (TEMP .LT. EMIN) THEN
          EMIN = TEMP
          IMIN = I
        END IF
  260 CONTINUE
C
C     Cycle through the NF confidence levels.
C
      DO 310 K = 1, NF
        TEMP = VAR * (NP - NFORCE) * FVAL(K)
        TEMPS = SQRT(TEMP)
C
C     Find largest and smallest roots of Spj0tvoll's equation (3.2).
C
        X1 = EMIN - ABS(EMIN * WK(IMIN)) / TEMPS
        X2 = EMIN - SQRT(SUME / TEMP)
        DO 290 I = 1, 2
  270     X = HALF * (X1 + X2)
          L = IVAL
          SUM = -TEMP
          DO 280 J = 1, N10N20
            SUM = SUM + (WK(L)*WK(J)/(WK(L) - X))**2
            L = L + 1
  280     CONTINUE
          IF (SUM .LE. ZERO) THEN
            X2 = X
          ELSE
            X1 = X
          END IF
          IF (ABS(X1 - X2) .GT. XTOL) GO TO 270
          X = HALF * (X1 + X2)
          IF (I .EQ. 1) THEN
            A = - X
            X1 = EMAX + ABS(EMAX * WK(IMAX)) / TEMPS
            X2 = EMAX + SQRT(SUME / TEMP)
          ELSE
            B =  X
          END IF
  290   CONTINUE
C
C     Calculate A1 and A2.
C
        A1(K) = -TEMP
        A2(K) = TEMP
        L = IVAL
        DO 300 I = 1, N10N20
          GAMMA2 = WK(I)**2
          TEMP = WK(L)
          L = L + 1
          A1(K) = A1(K) + TEMP * GAMMA2 / (A + TEMP)
          A2(K) = A2(K) + TEMP * GAMMA2 / (B - TEMP)
  300   CONTINUE
        A1(K) = A * A1(K)
        A2(K) = B * A2(K)
  310 CONTINUE
```

```
C
      RETURN
C-----------------------------------------------------------------------
C     Special cases; either N10 or N20 = 0.
C
  400 GAMMA2 = RSS(N0) - RSS(N1+N20)
      DO 410 K = 1, NF
        TEMP = SQRT(VAR * (NP - NFORCE) * FVAL(K))
        TEMPS = SQRT(GAMMA2)
        SUM = (TEMPS + TEMP)**2
        DIFF = MAX(ZERO, TEMPS-TEMP)
        DIFF = DIFF**2
        IF (N10 .EQ. 0) THEN
          A1(K) = -SUM
          A2(K) = -DIFF
        ELSE
          A1(K) = DIFF
          A2(K) = SUM
        END IF
  410 CONTINUE
C
      RETURN
      END
```

Notes on AS 274 "Least-squares routines to supplement those of Gentleman"
by Alan Miller.

This algorithm provides a set of high accuracy least squares routines which expand upon those provided by Gentleman in AS 75. In particular, it includes facilities for (weighted) least-squares for a subset of the variables, for changing the order of the variables, for testing for singularities, and for calculating an estimated covariance matrix for the regression coefficients.

The algorithm is NOT consistent with those which have been published in the same journal by Clarke (AS 163), Stirling (AS 164) or Smith (AS 268), in that the orthogonal reduction is stored in a different way. It is unfortunate that these authors have not used the same format as Gentleman.

The basic algorithm is the same as Gentleman's. As each new case is added, the orthogonal reduction is updated. In traditional least-squares notation, the algorithm goes straight from the X-matrix (the 'design' matrix) to the Cholesky factorization WITHOUT the intermediate step of forming a sum of squares and products matrix. Thus it avoids squaring the condition number. To be pedantic, it is actually the Banachiewicz factorization which is used. We use:

$$Q\,X = \mathrm{sqrt}(D)\,R$$

where Q is an orthonormal matrix such that Q'Q = I, D is a diagonal matrix (sqrt(D) is my crude way of indicating a diagonal matrix with elements on the diagonal which are the square roots of the elements stored in array D in the routines), and R is an upper triangular matrix with 1's on the diagonal. The array R is stored in a 1-dimensional array with elements stored by rows as shown below (the 1's on the diagonal are NOT stored).

```
Storage of R:    (1)  1   2   3   4   5
                      (1)  6   7   8   9
                           (1) 10  11  12
                               (1) 13  14
                                   (1) 15
                                       (1)
```

N.B. The matrix Q is a product of planar-rotation matrices (Jacobi/Givens rotations); it is not calculated or stored.
N.B. If you want to fit a constant (intercept) in your model, you can do it by putting the first element of each row of X equal to 1.0.
N.B. You may like to dimension XROW as XROW(0:K) in your calling program, where K is the number of variables (other than the constant). The parameter NP passed to routine INCLUD (and others) then takes the value K+1.

To perform the calculation of regression coefficients (and nothing else), you need to call the following routines in the order given:

1. clear - initializes arrays
2. includ - call once for each case to update the orthogonal reduction
3. tolset - calculates tolerances for each variable
4. sing (optional) - tests for singularities
5. regcf - calculates regression coefficients for the first NREQ variables

You can then add extra data and recalculate the regression coefficients.

Other routines (all require that 1 and 2 above have been used first).

ss calculates array RSS such that RSS(I) = the residual sum of squares with the first I variables in the model

cov calculates the estimated covariance matrix of the regression coefficients (the user must input a value for the residual variance VAR, usually the

residual sum of squares SSERR divided by NOBS - NP for all variables, or RSS(I) divided by NOBS - I for the first I variables).

pcorr calculates partial correlations with the first IN variables in the model. If IN = 1 and the first variable is identically equal to 1 then these are the usual full correlations. If IN = 0 it gives the cross products (not centered) divided by the square root of the product of the second moments (about zero) of the two variables.

vmove moves the variable in position FROM to position TO.

reordr calls vmove repeatedly to re-order the variables from the current order in integer array VORDER to that in the integer array LIST.

hdiag calculates the diagonal elements of the `hat' matrix used in various diagnostic statistics.

For vmove and reordr, the user must first set up an integer array VORDER which assigns a unique integer value with each variable. You may like to associate the value 0 with the constant in the model.

For cov, vmove, reordr and hdiag you must call tolset first.

*** Warning *** Routine INCLUD (from Gentleman's AS 75) overwrites the elements in array XROW.

Alan Miller
25 November 1991

Notes on handling singularities

If there is a singularity amongst your X-variables and routine SING is not used, you will get wrong answers from most routines. The orthogonal reduction can be written as:

$$X = Q' \, \mathrm{sqrt}(D) \, R$$

If we denote the columns of X as X1, X2, ..., Xk, and the columns of Q' as q1, q2, ..., qk, then we have:

$$x1 = r(11).q1$$
$$x2 = r(12).q1 + r(22).q2$$
$$x3 = r(13).q1 + r(23).q2 + r(33).q3$$

etc., where r(ij) = the (i,j)-element of R multiplied by the square root of the i-th diagonal element of D. That is x1 is equal to the first orthogonal direction, q1, multiplied by a scaling factor. x2 = a projection of length r(12) in the first direction, q1, plus a projection of length r(22) in a new direction, q2, which is orthogonal to q1, etc. If the X-directions are all orthogonal then all of the r(ij)'s for off-diagonal elements of R will be zero.

Let us suppose that there is a singularity amongst the X-variables, that is that one of the X-variables is exactly linearly related to some of the others. For simplicity, let us suppose that x3 = a.x1 + b.x2. The direction q3 is formed from x3 by subtracting its projections in directions q1 and q2 and then scaling so that it has length equal to 1. When the projections in directions q1 and q2 are subtracted in this case, there should be nothing left. In practice, there will be almost always be small rounding errors. Direction q3 will be formed from these rounding errors. Thus we will have a rogue direction in the columns of Q'. The projections of the dependant variable on this direction can be large. It is like correlating the Y-variable on a variable formed by using random numbers or a column of numbers from the phone directory.

Routine SING sets the projections on this direction equal to zero.

In the case just mentioned, a full rank X could be obtained by removing any one of the variables x1, x2 or x3. In AS274, the equivalent to removing a variable is to move it to the last position and then to only use the first (k-1) variables in subsequent calculations. You still tell routines such as REGCF that there are NP variables, but you set NREQ = NP - 1.

If you just want properties of a subset of variables, then you use either VMOVE or REORDR to re-order the variables so that the first ones are those which you are interested in, and you set NREQ equal to the number of those variables. In most cases, the constant (intercept) will be included in the model and so will be included in the count of variables, NREQ (N-required).

Features NOT included in AS274

There is no facility for removing or adding variables (columns). This could be done but adding variables requires the storage of the Q-matrix, or at least of the rotations used to form it.

There is no facility for two or more dependant variables, though this is fairly easy to program. The extra dependant variables can be treated as X-variables. Thus if we have dependant variables Y1, Y2 and Y3, then Y2 and Y3 can be added at the end of the list of X-variables. When looking at the properties of Y1 related to the X-variables, use only the first (NP-2) columns by setting NREQ = NP-2. When you want to look at Y2, just copy column (NP-1) of RBAR into THETAB - this requires a little thought to work out just where those elements are stored. You may want to first copy the original THETAB, or to copy the column of RBAR into a different array, perhaps THETAB2 say. The residual sum of squares, SSERR, for variable Y2 is D(NP-1).

Y3 is a little more difficult. You must first change the order of the variables so that Y3 is before Y2, otherwise you will be regressing Y3 on Y2, and you probably don't want to do that.

The easy way of thinking about the above is by considering each column of R (after scaling by the square roots of the diagonal elements of D) as the projections of the corresponding variable on all of those which have preceded it. The Y-variable is just another variable being projected upon a set of orthogonal directions formed by the variables which came before it. The vector THETAB is stored separately just for fast access as most users will have only one Y-variable; it could have been incorporated as the last column of RBAR.

Alan Miller
2 May 1993

```
C------------------------------------------------------------------------

      SUBROUTINE INCLUD(NP, NRBAR, WEIGHT, XROW, YELEM, D, RBAR, THETAB,
     +      SSERR, IER)
C
C     ALGORITHM AS274.1  APPL. STATIST. (1992) VOL 41, NO. 2
C
C     DOUBLE PRECISION VERSION
C
C     Calling this routine updates d, rbar, thetab and sserr by the
C     inclusion of xrow, yelem with the specified weight.
C     This version has been modified to make it slightly faster when the
C     early elements of XROW are not zeroes.
```

```
C
C     *** WARNING ***   The elements of XROW are over-written.
C
      INTEGER NP, NRBAR, IER
      DOUBLE PRECISION WEIGHT, XROW(NP), YELEM, D(NP), RBAR(*),
     +       THETAB(NP), SSERR
C
C     Local variables
C
      INTEGER I, K, NEXTR
      DOUBLE PRECISION ZERO, W, Y, XI, DI, WXI, DPI, CBAR, SBAR, XK
C
      DATA ZERO/0.D0/
C
C     Some checks.
C
      IER = 0
      IF (NP .LT. 1) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (IER .NE. 0) RETURN
C
      W = WEIGHT
      Y = YELEM
      NEXTR = 1
      DO 30 I = 1, NP
C
C     Skip unnecessary transformations.   Test on exact zeroes must be
C     used or stability can be destroyed.
C
        IF (W .EQ. ZERO) RETURN
        XI = XROW(I)
        IF (XI .EQ. ZERO) THEN
          NEXTR = NEXTR + NP - I
          GO TO 30
        END IF
        DI = D(I)
        WXI = W * XI
        DPI = DI + WXI*XI
        CBAR = DI / DPI
        SBAR = WXI / DPI
        W = CBAR * W
        D(I) = DPI
        IF (I .EQ. NP) GO TO 20
        DO 10 K = I+1, NP
          XK = XROW(K)
          XROW(K) = XK - XI * RBAR(NEXTR)
          RBAR(NEXTR) = CBAR * RBAR(NEXTR) + SBAR * XK
          NEXTR = NEXTR + 1
   10   CONTINUE
   20   XK = Y
        Y = XK - XI * THETAB(I)
        THETAB(I) = CBAR * THETAB(I) + SBAR * XK
   30 CONTINUE
C
C     Y * SQRT(W) is now equal to the Brown, Durbin & Evans recursive
C     residual.
C
      SSERR = SSERR + W * Y * Y
C
      RETURN
      END
```

```
C
      SUBROUTINE CLEAR(NP, NRBAR, D, RBAR, THETAB, SSERR, IER)
C
C     ALGORITHM AS274.2  APPL. STATIST. (1992) VOL.41, NO.2
C
C     Sets arrays to zero prior to calling AS75.1
C
      INTEGER NP, NRBAR, IER
      DOUBLE PRECISION D(NP), RBAR(*), THETAB(NP), SSERR
C
C     Local variables
C
      INTEGER I
      DOUBLE PRECISION ZERO
C
      DATA ZERO/0.D0/
C
C     Some checks.
C
      IER = 0
      IF (NP .LT. 1) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (IER .NE. 0) RETURN
C
      DO 10 I = 1, NP
        D(I) = ZERO
        THETAB(I) = ZERO
   10 CONTINUE
      DO 20 I = 1, NRBAR
   20 RBAR(I) = ZERO
      SSERR = ZERO
      RETURN
      END
C
      SUBROUTINE REGCF(NP, NRBAR, D, RBAR, THETAB, TOL, BETA, NREQ,
     +     IER)
C
C     ALGORITHM AS274.3  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Modified version of AS75.4 to calculate regression coefficients
C     for the first NREQ variables, given an orthogonal reduction from
C     AS75.1.
C
      INTEGER NP, NRBAR, NREQ, IER
      DOUBLE PRECISION D(NP), RBAR(*), THETAB(NP), TOL(NP), BETA(NP)
C
C     Local variables
C
      INTEGER I, J, NEXTR
      DOUBLE PRECISION ZERO
C
      DATA ZERO/0.D0/
C
C     Some checks.
C
      IER = 0
      IF (NP .LT. 1) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (NREQ .LT. 1 .OR. NREQ .GT. NP) IER = IER + 4
      IF (IER .NE. 0) RETURN
C
```

```
      DO 20 I = NREQ, 1, -1
        IF (SQRT(D(I)) .LT. TOL(I)) THEN
          BETA(I) = ZERO
          D(I) = ZERO
          GO TO 20
        END IF
        BETA(I) = THETAB(I)
        NEXTR = (I-1) * (NP+NP-I)/2 + 1
        DO 10 J = I+1, NREQ
          BETA(I) = BETA(I) - RBAR(NEXTR) * BETA(J)
          NEXTR = NEXTR + 1
   10   CONTINUE
   20 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE TOLSET(NP, NRBAR, D, RBAR, TOL, WORK, IER)
C
C     ALGORITHM AS274.4  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Sets up array TOL for testing for zeroes in an orthogonal
C     reduction formed using AS75.1.
C
      INTEGER NP, NRBAR, IER
      DOUBLE PRECISION D(NP), RBAR(*), TOL(NP), WORK(NP)
C
C     Local variables.
C
      INTEGER COL, ROW, POS
      DOUBLE PRECISION EPS, SUM
C
C     EPS is a machine-dependent constant.   For compilers which use
C     the IEEE format for floating-point numbers, recommended values
C     are 1.E-06 for single precision and 1.E-15 for double precision.
C
      DATA EPS/1.D-15/
C
C     Some checks.
C
      IER = 0
      IF (NP .LT. 1) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (IER .NE. 0) RETURN
C
C     Set TOL(I) = sum of absolute values in column I of RBAR after
C     scaling each element by the square root of its row multiplier.
C
      DO 10 COL = 1, NP
   10 WORK(COL) = SQRT(D(COL))
      DO 30 COL = 1, NP
        POS = COL - 1
        SUM = WORK(COL)
        DO 20 ROW = 1, COL-1
          SUM = SUM + ABS(RBAR(POS)) * WORK(ROW)
          POS = POS + NP - ROW - 1
  20    CONTINUE
        TOL(COL) = EPS * SUM
   30 CONTINUE
C
      RETURN
```

```
      END
C
      SUBROUTINE SING(NP, NRBAR, D, RBAR, THETAB, SSERR, TOL, LINDEP,
     +   WORK, IER)
C
C     ALGORITHM AS274.5  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Checks for singularities, reports, and adjusts orthogonal
C     reductions produced by AS75.1.
C
      INTEGER NP, NRBAR, IER
      DOUBLE PRECISION D(NP), RBAR(NRBAR), THETAB(NP), SSERR, TOL(NP),
     +        WORK(NP)
      LOGICAL LINDEP(NP)
C
C     Local variables
C
      DOUBLE PRECISION ZERO, TEMP
      INTEGER COL, POS, ROW, NP2, POS2
C
      DATA ZERO/0.D0/
C
C     Check input parameters
C
      IER = 0
      IF (NP .LE. 0) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (IER .NE. 0) RETURN
C
      DO 10 COL = 1, NP
   10 WORK(COL) = SQRT(D(COL))
C
      DO 40 COL = 1, NP
C
C     Set elements within RBAR to zero if they are less than TOL(COL) in
C     absolute value after being scaled by the square root of their row
C     multiplier.
C
        TEMP = TOL(COL)
        POS = COL - 1
        DO 30 ROW = 1, COL-1
          IF (ABS(RBAR(POS)) * WORK(ROW) .LT. TEMP) RBAR(POS) = ZERO
          POS = POS + NP - ROW - 1
   30   CONTINUE
C
C     If diagonal element is near zero, set it to zero, set appropriate
C     element of LINDEP, and use INCLUD to augment the projections in
C     the lower rows of the orthogonalization.
C
        LINDEP(COL) = .FALSE.
        IF (WORK(COL) .LT. TEMP) THEN
          LINDEP(COL) = .TRUE.
          IER = IER - 1
          IF (COL .LT. NP) THEN
            NP2 = NP - COL
            POS2 = POS + NP - COL + 1
            CALL INCLUD(NP2, NP2*(NP2-1)/2, D(COL), RBAR(POS+1),
     +           THETAB(COL), D(COL+1), RBAR(POS2), THETAB(COL+1),
     +           SSERR, IER)
          ELSE
            SSERR = SSERR + D(COL) * THETAB(COL)**2
```

```
            END IF
            D(COL) = ZERO
            WORK(COL) = ZERO
            THETAB(COL) = ZERO
          END IF
   40   CONTINUE
        RETURN
        END
C
        SUBROUTINE SS(NP, D, THETAB, SSERR, RSS, IER)
C
C       ALGORITHM AS274.6  APPL. STATIST. (1992) VOL 41, NO.2
C
C       Calculates partial residual sums of squares from an orthogonal
C       reduction from AS75.1.
C
        INTEGER NP, IER
        DOUBLE PRECISION D(NP), THETAB(NP), SSERR, RSS(NP)
C
C       Local variables
C
        INTEGER I
        DOUBLE PRECISION SUM
C
C       Some checks.
C
        IER = 0
        IF (NP .LT. 1) IER = 1
        IF (IER .NE. 0) RETURN
C
        SUM = SSERR
        RSS(NP) = SSERR
        DO 10 I = NP, 2, -1
          SUM = SUM + D(I) * THETAB(I)**2
          RSS(I-1) = SUM
   10   CONTINUE
        RETURN
        END
C
        SUBROUTINE COV(NP, NRBAR, D, RBAR, NREQ, RINV, VAR, COVMAT,
     +      DIMCOV, STERR, IER)
C
C       ALGORITHM AS274.7  APPL. STATIST. (1992) VOL 41, NO.2
C
C       Calculate covariance matrix for regression coefficients for the
C       first NREQ variables, from an orthogonal reduction produced from
C       AS75.1.
C
C       Auxiliary routine called: INV
C
        INTEGER NP, NRBAR, NREQ, DIMCOV, IER
        DOUBLE PRECISION D(NP), RBAR(*), RINV(*), VAR, COVMAT(DIMCOV),
     +       STERR(NP)
C
C       Local variables.
C
        INTEGER POS, ROW, START, POS2, COL, POS1, K
        DOUBLE PRECISION ZERO, ONE, SUM
C
        DATA ZERO/0.D0/, ONE/1.D0/
C
```

```
C     Some checks.
C
      IER = 0
      IF (NP .LT. 1) IER = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
      IF (DIMCOV .LT. NREQ*(NREQ+1)/2) IER = IER + 4
      DO 10 ROW = 1, NREQ
        IF (D(ROW) .EQ. ZERO) IER = -ROW
   10 CONTINUE
      IF (IER .NE. 0) RETURN
C
      CALL INV(NP, NRBAR, RBAR, NREQ, RINV)
      POS = 1
      START = 1
      DO 40 ROW = 1, NREQ
        POS2 = START
        DO 30 COL = ROW, NREQ
          POS1 = START + COL - ROW
          IF (ROW .EQ. COL) THEN
            SUM = ONE / D(COL)
          ELSE
            SUM = RINV(POS1-1) / D(COL)
          END IF
          DO 20 K = COL+1, NREQ
            SUM = SUM + RINV(POS1) * RINV(POS2) / D(K)
            POS1 = POS1 + 1
            POS2 = POS2 + 1
   20     CONTINUE
          COVMAT(POS) = SUM * VAR
          IF (ROW .EQ. COL) STERR(ROW) = SQRT(COVMAT(POS))
          POS = POS + 1
   30   CONTINUE
        START = START + NREQ - ROW
   40 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE INV(NP, NRBAR, RBAR, NREQ, RINV)
C
C     ALGORITHM AS274.8  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Invert first NREQ rows and columns of Cholesky factorization
C     produced by AS75.1.
C
      INTEGER NP, NRBAR, NREQ
      DOUBLE PRECISION RBAR(*), RINV(*)
C
C     Local variables.
C
      INTEGER POS, ROW, COL, START, K, POS1, POS2
      DOUBLE PRECISION SUM, ZERO
C
      DATA ZERO/0.D0/
C
C     Invert RBAR ignoring row multipliers, from the bottom up.
C
      POS = NREQ * (NREQ-1)/2
      DO 30 ROW = NREQ-1, 1, -1
        START = (ROW-1) * (NP+NP-ROW)/2 + 1
        DO 20 COL = NREQ, ROW+1, -1
```

```
            POS1 = START
            POS2 = POS
            SUM = ZERO
            DO 10 K = ROW+1, COL-1
              POS2 = POS2 + NREQ - K
              SUM = SUM - RBAR(POS1) * RINV(POS2)
              POS1 = POS1 + 1
   10       CONTINUE
            RINV(POS) = SUM - RBAR(POS1)
            POS = POS - 1
   20     CONTINUE
   30   CONTINUE
C
        RETURN
        END
C
        SUBROUTINE PCORR(NP, NRBAR, D, RBAR, THETAB, SSERR, IN, WORK,
       +      CORMAT, DIMC, YCORR, IER)
C
C       ALGORITHM AS274.9  APPL. STATIST. (1992) VOL 41, NO.2
C
C       Calculate partial correlations after the first IN variables
C       have been forced into the regression.
C
C       Auxiliary routine called: COR
C
        INTEGER NP, NRBAR, IN, DIMC, IER
        DOUBLE PRECISION D(NP), RBAR(*), THETAB(NP), SSERR, WORK(NP),
       +        CORMAT(*), YCORR
C
C       Local variables.
C
        INTEGER START, IN1, I
        DOUBLE PRECISION ZERO
C
        DATA ZERO/0.D0/
C
C       Some checks.
C
        IER = 0
        IF (NP .LT. 1) IER = 1
        IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
        IF (IN .LT. 0 .OR. IN .GT. NP-1) IER = IER + 4
        IF (DIMC .LT. (NP-IN)*(NP-IN-1)/2) IER = IER + 8
        IF (IER .NE. 0) RETURN
C
        START = IN * (NP+NP-IN-1)/2 + 1
        IN1 = IN + 1
        CALL COR(NP-IN, D(IN1), RBAR(START), THETAB(IN1), SSERR, WORK,
       +      CORMAT, YCORR)
C
C       Check for zeroes.
C
        DO 10 I = 1, NP-IN
          IF (WORK(I) .LE. ZERO) IER = -I
   10   CONTINUE
C
        RETURN
        END
C
        SUBROUTINE COR(NP, D, RBAR, THETAB, SSERR, WORK, CORMAT, YCORR)
```

```
C
C     ALGORITHM AS274.10  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Calculate correlations from an orthogonal reduction.   This
C     routine will usually be called from PCORR, which will have
C     removed the appropriate number of rows at the start.
C
      INTEGER NP
      DOUBLE PRECISION D(NP), RBAR(*), THETAB(NP), SSERR, WORK(NP),
     +      CORMAT(*), YCORR(NP)
C
C     Local variables.
C
      INTEGER ROW, POS, COL1, POS1, COL2, POS2, DIFF
      DOUBLE PRECISION SUMY, SUM, ZERO
C
      DATA ZERO/0.D0/
C
C     Process by columns, including the projections of the dependent
C     variable (THETAB).
C
      SUMY = SSERR
      DO 10 ROW = 1, NP
   10 SUMY = SUMY + D(ROW) * THETAB(ROW)**2
      SUMY = SQRT(SUMY)
      POS = NP*(NP-1)/2
      DO 70 COL1 = NP, 1, -1
C
C     Calculate the length of column COL1.
C
        SUM = D(COL1)
        POS1 = COL1 - 1
        DO 20 ROW = 1, COL1-1
          SUM = SUM + D(ROW) * RBAR(POS1)**2
          POS1 = POS1 + NP - ROW - 1
   20   CONTINUE
        WORK(COL1) = SQRT(SUM)
C
C     If SUM = 0, set all correlations with this variable to zero.
C
        IF (SUM .EQ. ZERO) THEN
          YCORR(COL1) = ZERO
          DO 30 COL2 = NP, COL1+1, -1
            CORMAT(POS) = ZERO
            POS = POS - 1
   30     CONTINUE
          GO TO 70
        END IF
C
C     Form cross-products, then divide by product of column lengths.
C
        SUM = D(COL1) * THETAB(COL1)
        POS1 = COL1 - 1
        DO 40 ROW = 1, COL1-1
          SUM = SUM + D(ROW) * RBAR(POS1) * THETAB(ROW)
          POS1 = POS1 + NP - ROW - 1
   40   CONTINUE
        YCORR(COL1) = SUM / (SUMY * WORK(COL1))
C
        DO 60 COL2 = NP, COL1+1, -1
          IF (WORK(COL2) .GT. ZERO) THEN
```

```
              POS1 = COL1 - 1
              POS2 = COL2 - 1
              DIFF = COL2 - COL1
              SUM = ZERO
              DO 50 ROW = 1, COL1-1
                SUM = SUM + D(ROW) * RBAR(POS1) * RBAR(POS2)
                POS1 = POS1 + NP - ROW - 1
                POS2 = POS1 + DIFF
   50         CONTINUE
              SUM = SUM + D(COL1) * RBAR(POS2)
              CORMAT(POS) = SUM / (WORK(COL1) * WORK(COL2))
            ELSE
              CORMAT(POS) = ZERO
            END IF
            POS = POS - 1
   60     CONTINUE
   70   CONTINUE
C
        RETURN
        END
C
        SUBROUTINE VMOVE(NP, NRBAR, VORDER, D, RBAR, THETAB, RSS, FROM,
       +    TO, TOL, IER)
C
C       ALGORITHM AS274.11 APPL. STATIST. (1992) VOL 41, NO.2
C
C       Move variable from position FROM to position TO in an
C       orthogonal reduction produced by AS75.1.
C
        INTEGER NP, NRBAR, VORDER(NP), FROM, TO, IER
        DOUBLE PRECISION D(NP), RBAR(*), THETAB(NP), RSS(NP), TOL(NP)
C
C       Local variables
C
        DOUBLE PRECISION ZERO, D1, D2, X, ONE, D1NEW, D2NEW, CBAR, SBAR, Y
        INTEGER M, FIRST, LAST, INC, M1, M2, MP1, COL, POS, ROW
C
        DATA ZERO/0.D0/, ONE/1.D0/
C
C       Check input parameters
C
        IER = 0
        IF (NP .LE. 0) IER = 1
        IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
        IF (FROM .LT. 1 .OR. FROM .GT. NP) IER = IER + 4
        IF (TO .LT. 1 .OR. TO .GT. NP) IER = IER + 8
        IF (IER .NE. 0) RETURN
C
        IF (FROM .EQ. TO) RETURN
C
        IF (FROM .LT. TO) THEN
          FIRST = FROM
          LAST = TO - 1
          INC = 1
        ELSE
          FIRST = FROM - 1
          LAST = TO
          INC = -1
        END IF
        DO 70 M = FIRST, LAST, INC
C
```

```
C     Find addresses of first elements of RBAR in rows M and (M+1).
C
        M1 = (M-1)*(NP+NP-M)/2 + 1
        M2 = M1 + NP - M
        MP1 = M + 1
        D1 = D(M)
        D2 = D(MP1)
C
C     Special cases.
C
        IF (D1 .EQ. ZERO .AND. D2 .EQ. ZERO) GO TO 40
        X = RBAR(M1)
        IF (ABS(X) * SQRT(D1) .LT. TOL(MP1)) THEN
          X = ZERO
        END IF
        IF (D1 .EQ. ZERO .OR. X .EQ. ZERO) THEN
          D(M) = D2
          D(MP1) = D1
          RBAR(M1) = ZERO
          DO 10 COL = M+2, NP
            M1 = M1 + 1
            X = RBAR(M1)
            RBAR(M1) = RBAR(M2)
            RBAR(M2) = X
            M2 = M2 + 1
   10     CONTINUE
          X = THETAB(M)
          THETAB(M) = THETAB(MP1)
          THETAB(MP1) = X
          GO TO 40
        ELSE IF (D2 .EQ. ZERO) THEN
          D(M) = D1 * X**2
          RBAR(M1) = ONE / X
          DO 20 COL = M+2, NP
            M1 = M1 + 1
            RBAR(M1) = RBAR(M1) / X
   20     CONTINUE
          THETAB(M) = THETAB(M) / X
          GO TO 40
        END IF
C
C     Planar rotation in regular case.
C
        D1NEW = D2 + D1*X**2
        CBAR = D2 / D1NEW
        SBAR = X * D1 / D1NEW
        D2NEW = D1 * CBAR
        D(M) = D1NEW
        D(MP1) = D2NEW
        RBAR(M1) = SBAR
        DO 30 COL = M+2, NP
          M1 = M1 + 1
          Y = RBAR(M1)
          RBAR(M1) = CBAR*RBAR(M2) + SBAR*Y
          RBAR(M2) = Y - X*RBAR(M2)
          M2 = M2 + 1
   30   CONTINUE
        Y = THETAB(M)
        THETAB(M) = CBAR*THETAB(MP1) + SBAR*Y
        THETAB(MP1) = Y - X*THETAB(MP1)
C
```

```
C     Swap columns M and (M+1) down to row (M-1).
C
   40   IF (M .EQ. 1) GO TO 60
        POS = M
        DO 50 ROW = 1, M-1
          X = RBAR(POS)
          RBAR(POS) = RBAR(POS-1)
          RBAR(POS-1) = X
          POS = POS + NP - ROW - 1
   50   CONTINUE
C
C     Adjust variable order (VORDER), the tolerances (TOL) and
C     the vector of residual sums of squares (RSS).
C
   60   M1 = VORDER(M)
        VORDER(M) = VORDER(MP1)
        VORDER(MP1) = M1
        X = TOL(M)
        TOL(M) = TOL(MP1)
        TOL(MP1) = X
        RSS(M) = RSS(MP1) + D(MP1) * THETAB(MP1)**2
   70 CONTINUE
C
        RETURN
        END
C
        SUBROUTINE REORDR(NP, NRBAR, VORDER, D, RBAR, THETAB, RSS, TOL,
      +      LIST, N, POS1, IER)
C
C     ALGORITHM AS274.12  APPL. STATIST. (1992) VOL 41, NO.2
C
C     Re-order the variables in an orthogonal reduction produced by
C     AS75.1 so that the N variables in LIST start at position POS1,
C     though will not necessarily be in the same order as in LIST.
C     Any variables in VORDER before position POS1 are not moved.
C
C     Auxiliary routine called: VMOVE
C
        INTEGER NP, NRBAR, VORDER(NP), N, LIST(N), POS1, IER
        DOUBLE PRECISION D(NP), RBAR(NRBAR), THETAB(NP), RSS(NP), TOL(NP)
C
C     Local variables.
C
        INTEGER NEXT, I, L, J
C
C     Check N.
C
        IER = 0
        IF (NP .LT. 1) IER = 1
        IF (NRBAR .LT. NP*(NP-1)/2) IER = IER + 2
        IF (N .LT. 1 .OR. N .GE. NP+1-POS1) IER = IER + 4
        IF (IER .NE. 0) RETURN
C
C     Work through VORDER finding variables which are in LIST.
C
        NEXT = POS1
        I = POS1
   10   L = VORDER(I)
        DO 20 J = 1, N
          IF (L .EQ. LIST(J)) GO TO 40
   20   CONTINUE
```

```
   30 I = I + 1
      IF (I .LE. NP) GO TO 10
C
C     If this point is reached, one or more variables in LIST has not
C     been found.
C
      IER = 8
      RETURN
C
C     Variable L is in LIST; move it up to position NEXT if it is not
C     already there.
C
   40 IF (I .GT. NEXT) CALL VMOVE(NP, NRBAR, VORDER, D, RBAR, THETAB,
     +      RSS, I, NEXT, TOL, IER)
      NEXT = NEXT + 1
      IF (NEXT .LT. N+POS1) GO TO 30
C
      RETURN
      END
C
      SUBROUTINE HDIAG(XROW, NP, NRBAR, D, RBAR, TOL, NREQ, HII, WK,
     +     IFAULT)
C
C     ALGORITHM AS274.13  APPL. STATIST. (1992) VOL.41, NO.2
C
      INTEGER NP, NRBAR, NREQ, IFAULT
      DOUBLE PRECISION XROW(NP), D(NP), RBAR(*), TOL(NP), HII, WK(NP)
C
C     Local variables
C
      INTEGER COL, ROW, POS
      DOUBLE PRECISION ZERO, SUM
C
      DATA ZERO /0.0D0/
C
C     Some checks
C
      IFAULT = 0
      IF (NP .LT. 1) IFAULT = 1
      IF (NRBAR .LT. NP*(NP-1)/2) IFAULT = IFAULT + 2
      IF (NREQ .GT. NP) IFAULT = IFAULT + 4
      IF (IFAULT .NE. 0) RETURN
C
C     The elements of XROW.inv(RBAR).sqrt(D) are calculated and stored
C     in WK.
C
      HII = ZERO
      DO 20 COL = 1, NREQ
        IF (SQRT(D(COL)) .LE. TOL(COL)) THEN
          WK(COL) = ZERO
          GO TO 20
        END IF
        POS = COL - 1
        SUM = XROW(COL)
        DO 10 ROW = 1, COL-1
          SUM = SUM - WK(ROW)*RBAR(POS)
          POS = POS + NP - ROW - 1
   10   CONTINUE
        WK(COL) = SUM
        HII = HII + SUM**2 / D(COL)
   20 CONTINUE
```

```
C
      RETURN
      END
```

(File: LSQ.DOC)
The module LSQ is for unconstrained least-squares fitting. It is based upon Applied Statistics algorithm AS 274 (see comments at the start of the module). A planar-rotation algorithm is used to update the QR-factorization. This makes it suitable for updating regressions as more data become available.

By taking advantage of the MODULE facility, it has been possible to remove most of the arguments to routines. Apart from the new function VARPRD, and a back-substitution routine BKSUB2 which it calls, the routines behave as in AS 274.

This release, version 1.00 dated 24 May 1995, has only been tested using one Fortran 90 compiler, Lahey's LF90 versions up to 1.10c. It would be appreciated if users could report their findings with other compilers, or with the various translators to Fortran 77 and C.

The package as posted comprises:

LSQ.DOC	this brief note
LSQ.F90	the least-squares module
DEMO.F90	a simple demonstration program
FUELCONS.DAT	a file of data to be read by the demo program
TEST1.F90	another program which fits a cubic polynomial and has a deliberate singularity

Reference:
Miller, A.J. (1992). Algorithm AS 274: Least squares routines to supplement those of Gentleman. Appl. Statist., vol.41(2), 458-478.

!--
!--

(File: LSQ.F90)

```
MODULE lsq

!     Module for unconstrained linear least-squares calculations.
!     The algorithm is suitable for updating LS calculations as more
!     data are added.   This is sometimes called recursive estimation.
!     Only one dependent variable is allowed.
!     Based upon Applied Statistics algorithm AS 274.
!     Translation from Fortran 77 to Fortran 90 by Alan Miller.
!     A subroutine, VARPRD, has been added for calculating the variances
!     of predicted values, and this uses a subroutine BKSUB2.

!     Version 1.00, 25 May 1995
!     Author: Alan Miller
!             CSIRO Division of Mathematics & Statistics
!             Private Bag 10, Rosebank MDC
!             Clayton 3169, Victoria, Australia
!     Phone: (+61) 3 9545-8036      Fax: (+61) 3 9545-8080
!     e-mail: Alan.Miller @ mel.dms.csiro.au

!     The PUBLIC variables are:
!     lsq_kind = a KIND parameter for the floating-point quantities calculated
!                in this module.   See the more detailed explanation below.
!                This KIND parameter should be used for all floating-point
!                arguments passed to routines in this module.

!     nobs    = the number of observations processed to date.
!     ncol    = the total number of variables, including one for the constant,
```

```
!               if a constant is being fitted.
!     r_dim   = the dimension of array r = ncol*(ncol-1)/2
!     vorder  = an integer vector storing the current order of the variables
!               in the QR-factorization.   The initial order is 0, 1, 2, ...
!               if a constant is being fitted, or 1, 2, ... otherwise.
!     initialized = a logical variable which indicates whether space has
!                   been allocated for various arrays.
!     tol_set = a logical variable which is set when subroutine TOLSET has
!               been called to calculate tolerances for use in testing for
!               singularities.
!     rss_set = a logical variable indicating whether residual sums of squares
!               are available and usable.
!     d()     = array of row multipliers for the Cholesky factorization.
!               The factorization is X = Q.sqrt(D).R where Q is an ortho-
!               normal matrix which is NOT stored, D is a diagonal matrix
!               whose diagonal elements are stored in array d, and R is an
!               upper-triangular matrix with 1's as its diagonal elements.
!     rhs()   = vector of RHS projections (after scaling by sqrt(D)).
!               Thus Q'y = sqrt(D).rhs
!     r()     = the upper-triangular matrix R.   The upper triangle only,
!               excluding the implicit 1's on the diagonal, are stored by
!               rows.
!     tol()   = array of tolerances used in testing for singularities.
!     rss()   = array of residual sums of squares.   rss(i) is the residual
!               sum of squares with the first r variables in the model.
!               By changing the order of variables, the residual sums of
!               squares can be found for all possible subsets of the variables.
!               The residual sum of squares with NO variables in the model,
!               that is the total sum of squares of the y-values, can be
!               calculated as rss(1) + d(1)*rhs(1)^2.   If the first variable
!               is a constant, then rss(1) is the sum of squares of
!               (y - ybar) where ybar is the average value of y.
!     sserr   = residual sum of squares with all of the variables included.
!
!-----------------------------------------------------------------------

!     General declarations

INTEGER                       :: nobs, ncol, r_dim
INTEGER, ALLOCATABLE          :: vorder(:)
LOGICAL                       :: initialized = .false.,                  &
                                 tol_set = .false., rss_set = .false.

!     Note. lsq_kind is being set to give at least 10 decimal digit
!           representation of floating point numbers.   This should be
!           adequate for most problems except the fitting of polynomials.
!           lsq_kind is being set so that the same code can be run on PCs
!           and Unix systems, which will usually represent floating-point
!           numbers in `double precision', and other systems with larger
!           word lengths which will give similar accuracy in `single
!           precision'.

INTEGER, PARAMETER            :: lsq_kind = SELECTED_REAL_KIND(10,70)
REAL (lsq_kind), ALLOCATABLE  :: d(:), rhs(:), r(:), tol(:), rss(:)
REAL (lsq_kind)               :: sserr, zero = 0.D0, one = 1.D0,         &
                                 vsmall = 1.d-69

PUBLIC                        :: lsq_kind, nobs, ncol, r_dim, vorder,    &
                                 initialized, tol_set, rss_set,          &
                                 d, rhs, r, tol, rss, sserr
PRIVATE                       :: zero, one, vsmall
```

```
SAVE                          ! Save everything

CONTAINS

SUBROUTINE startup(nvar, fit_const)

!     Allocates dimensions for arrays and initializes to zero
!     The calling program must set nvar = the number of variables, and
!     fit_const = .true. if a constant is to be included in the model,
!     otherwise fit_const = .false.
!
!------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)           :: nvar
LOGICAL, INTENT(IN)           :: fit_const

!     Local variable
INTEGER                       :: i

nobs = 0
IF (fit_const) THEN
  ncol = nvar + 1
ELSE
  ncol = nvar
END IF

IF (initialized) DEALLOCATE(d, rhs, r, tol, rss, vorder)
r_dim = ncol * (ncol - 1)/2
ALLOCATE( d(ncol), rhs(ncol), r(r_dim), tol(ncol), rss(ncol), vorder(ncol) )

d = zero
rhs = zero
r = zero
sserr = zero

IF (fit_const) THEN
  DO i = 1, ncol
    vorder(i) = i-1
  END DO
ELSE
  DO i = 1, ncol
    vorder(i) = i
  END DO
END IF ! (fit_const)

initialized = .true.
tol_set = .false.
rss_set = .false.

END SUBROUTINE startup



SUBROUTINE includ(weight, xrow, yelem)

!     ALGORITHM AS75.1  APPL. STATIST. (1974) VOL.23, NO. 3

!     Calling this routine updates D, R, RHS and SSERR by the
```

```
!     inclusion of xrow, yelem with the specified weight.

!     *** WARNING  Array XROW is overwritten.

!     N.B. As this routine will be called many times in most applications,
!          checks have been eliminated.
!
!------------------------------------------------------------------------


IMPLICIT NONE
REAL (lsq_kind),INTENT(IN)    :: weight, yelem
REAL (lsq_kind),INTENT(INOUT) :: xrow(ncol)

!     Local variables

INTEGER           :: i, k, nextr
REAL (lsq_kind)   :: w, y, xi, di, wxi, dpi, cbar, sbar, xk

nobs = nobs + 1
w = weight
y = yelem
rss_set = .false.
nextr = 1
DO i = 1, ncol

!     Skip unnecessary transformations.   Test on exact zeroes must be
!     used or stability can be destroyed.

  IF (ABS(w) < vsmall) RETURN
  xi = xrow(i)
  IF (ABS(xi) < vsmall) THEN
    nextr = nextr + ncol - i
  ELSE
    di = d(i)
    wxi = w * xi
    dpi = di + wxi*xi
    cbar = di / dpi
    sbar = wxi / dpi
    w = cbar * w
    d(i) = dpi
    DO k = i+1, ncol
      xk = xrow(k)
      xrow(k) = xk - xi * r(nextr)
      r(nextr) = cbar * r(nextr) + sbar * xk
      nextr = nextr + 1
    END DO
    xk = y
    y = xk - xi * rhs(i)
    rhs(i) = cbar * rhs(i) + sbar * xk
  END IF
END DO ! i = 1, ncol

!     Y * SQRT(W) is now equal to the Brown, Durbin & Evans recursive
!     residual.

sserr = sserr + w * y * y

RETURN
END SUBROUTINE includ
```

```
SUBROUTINE regcf(beta, nreq, ifault)

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Modified version of AS75.4 to calculate regression coefficients
!     for the first NREQ variables, given an orthogonal reduction from
!     AS75.1.
!
!------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)            :: nreq
INTEGER, INTENT(OUT)           :: ifault
REAL (lsq_kind) ,INTENT(OUT)   :: beta(nreq)

!     Local variables

INTEGER             :: i, j, nextr

!     Some checks.

ifault = 0
IF (nreq .LT. 1 .OR. nreq .GT. ncol) ifault = ifault + 4
IF (ifault .NE. 0) RETURN

IF (.NOT. tol_set) CALL tolset

DO i = nreq, 1, -1
  IF (SQRT(d(i)) .LT. tol(i)) THEN
    beta(i) = zero
    d(i) = zero
  ELSE
    beta(i) = rhs(i)
    nextr = (i-1) * (ncol+ncol-i)/2 + 1
    DO j = i+1, nreq
      beta(i) = beta(i) - r(nextr) * beta(j)
      nextr = nextr + 1
    END DO ! j = i+1, nreq
  END IF
END DO ! i = nreq, 1, -1

RETURN
END SUBROUTINE regcf



SUBROUTINE tolset

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Sets up array TOL for testing for zeroes in an orthogonal
!     reduction formed using AS75.1.

!     Local variables.
!
!------------------------------------------------------------------------

IMPLICIT NONE
```

```
!     Local variables

INTEGER           :: col, row, pos
REAL (lsq_kind)   :: eps, ten = 10.0, total, work(ncol)

!     EPS is a machine-dependent constant.

eps = ten * EPSILON(ten)

!     Set tol(i) = sum of absolute values in column I of R after
!     scaling each element by the square root of its row multiplier,
!     multiplied by EPS.

work = SQRT(d)
DO col = 1, ncol
  pos = col - 1
  total = work(col)
  DO row = 1, col-1
    total = total + ABS(r(pos)) * work(row)
    pos = pos + ncol - row - 1
  END DO
  tol(col) = eps * total
END DO

tol_set = .TRUE.
RETURN
END SUBROUTINE tolset




SUBROUTINE sing(lindep, ifault)

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Checks for singularities, reports, and adjusts orthogonal
!     reductions produced by AS75.1.

!     Auxiliary routine called: INCLUD
!
!-----------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(OUT)        :: ifault
LOGICAL, INTENT(OUT)        :: lindep(ncol)

!     Local variables

REAL (lsq_kind)   :: temp, x(ncol), work(ncol), y, weight
INTEGER           :: col, pos, row, pos2

ifault = 0

work = SQRT(d)

DO col = 1, ncol

!     Set elements within R to zero if they are less than tol(col) in
!     absolute value after being scaled by the square root of their row
!     multiplier.
```

```
      temp = tol(col)
      pos = col - 1
      DO row = 1, col-1
        IF (ABS(r(pos)) * work(row) .LT. temp) r(pos) = zero
        pos = pos + ncol - row - 1
      END DO

!     If diagonal element is near zero, set it to zero, set appropriate
!     element of LINDEP, and use INCLUD to augment the projections in
!     the lower rows of the orthogonalization.

      lindep(col) = .FALSE.
      IF (work(col) .LE. temp) THEN
        lindep(col) = .TRUE.
        ifault = ifault - 1
        IF (col .LT. ncol) THEN
          pos2 = pos + ncol - col + 1
          x = zero
          x(col+1:ncol) = r(pos+1:pos2-1)
          y = rhs(col)
          weight = d(col)
          r(pos+1:pos2-1) = zero
          d(col) = zero
          rhs(col) = zero
          CALL includ(weight, x, y)
        ELSE
          sserr = sserr + d(col) * rhs(col)**2
        END IF ! (col .LT. ncol)
      END IF ! (work(col) .LE. temp)
    END DO ! col = 1, ncol
    RETURN
    END SUBROUTINE sing



    SUBROUTINE ss

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Calculates partial residual sums of squares from an orthogonal
!     reduction from AS75.1.
!
!------------------------------------------------------------------------

!     Local variables

    IMPLICIT NONE
    INTEGER           :: i
    REAL (lsq_kind)   :: total

    total = sserr
    rss(ncol) = sserr
    DO i = ncol, 2, -1
      total = total + d(i) * rhs(i)**2
      rss(i-1) = total
    END DO

    rss_set = .TRUE.
    RETURN
    END SUBROUTINE ss
```

```
SUBROUTINE cov(nreq, var, covmat, dimcov, sterr, ifault)

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Calculate covariance matrix for regression coefficients for the
!     first nreq variables, from an orthogonal reduction produced from
!     AS75.1.

!     Auxiliary routine called: INV
!
!------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)          :: nreq, dimcov
INTEGER, INTENT(OUT)         :: ifault
REAL (lsq_kind), INTENT(OUT) :: var, covmat(dimcov), sterr(nreq)

!     Local variables.

INTEGER                      :: dim_rinv, pos, row, start, pos2, col, pos1, k
REAL (lsq_kind)              :: total
REAL (lsq_kind), ALLOCATABLE :: rinv(:)

!     Check that dimension of array covmat is adequate.

IF (dimcov < nreq*(nreq+1)/2) THEN
  ifault = 1
  RETURN
END IF

!     Check for small or zero multipliers on the diagonal.

ifault = 0
DO row = 1, nreq
  IF (ABS(d(row)) .LT. vsmall) ifault = -row
END DO
IF (ifault .NE. 0) RETURN

!     Calculate estimate of the residual variance.

IF (nobs > nreq) THEN
  var = rss(nreq) / (nobs - nreq)
ELSE
  ifault = 2
  RETURN
END IF

dim_rinv = nreq*(nreq-1)/2
ALLOCATE ( rinv(dim_rinv) )

CALL INV(nreq, rinv)
pos = 1
start = 1
DO row = 1, nreq
  pos2 = start
  DO col = row, nreq
    pos1 = start + col - row
    IF (row .EQ. col) THEN
      total = one / d(col)
```

```
      ELSE
        total = rinv(pos1-1) / d(col)
      END IF
      DO K = col+1, nreq
        total = total + rinv(pos1) * rinv(pos2) / d(k)
        pos1 = pos1 + 1
        pos2 = pos2 + 1
      END DO ! K = col+1, nreq
      covmat(pos) = total * var
      IF (row .EQ. col) sterr(row) = SQRT(covmat(pos))
      pos = pos + 1
    END DO ! col = row, nreq
    start = start + nreq - row
  END DO ! row = 1, nreq

  RETURN
  END SUBROUTINE cov



  SUBROUTINE inv(nreq, rinv)

  !     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

  !     Invert first nreq rows and columns of Cholesky factorization
  !     produced by AS 75.1.
  !
  !-----------------------------------------------------------------------

  IMPLICIT NONE
  INTEGER, INTENT(IN)            :: nreq
  REAL (lsq_kind), INTENT(OUT)   :: rinv(*)

  !     Local variables.

  INTEGER           :: pos, row, col, start, K, pos1, pos2
  REAL (lsq_kind)   :: total

  !     Invert R ignoring row multipliers, from the bottom up.

  pos = nreq * (nreq-1)/2
  DO row = nreq-1, 1, -1
    start = (row-1) * (ncol+ncol-row)/2 + 1
    DO col = nreq, row+1, -1
      pos1 = start
      pos2 = pos
      total = zero
      DO K = row+1, col-1
        pos2 = pos2 + nreq - K
        total = total - r(pos1) * rinv(pos2)
        pos1 = pos1 + 1
      END DO ! K = row+1, col-1
      rinv(pos) = total - r(pos1)
      pos = pos - 1
    END DO ! col = nreq, row+1, -1
  END DO ! row = nreq-1, 1, -1

  RETURN
  END SUBROUTINE inv
```

```
SUBROUTINE partial_corr(in, cormat, dimc, ycorr, ifault)

!     Replaces subroutines PCORR and COR of:
!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2

!     Calculate partial correlations after the variables in rows
!     1, 2, ..., IN have been forced into the regression.
!     If IN = 1, and the first row of R represents a constant in the
!     model, then the usual simple correlations are returned.

!     If IN = 0, the value returned in array CORMAT for the correlation
!     of variables Xi & Xj is:
!       sum ( Xi.Xj ) / Sqrt ( sum (Xi^2) . sum (Xj^2) )

!     On return, array CORMAT contains the upper triangle of the matrix of
!     partial correlations stored by rows, excluding the 1's on the diagonal.
!     e.g. if IN = 2, the consecutive elements returned are:
!     (3,4) (3,5) ... (3,ncol), (4,5) (4,6) ... (4,ncol), etc.
!     Array YCORR stores the partial correlations with the Y-variable
!     starting with YCORR(IN+1) = partial correlation with the variable in
!     position (IN+1).
!
!------------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)          :: in, dimc
INTEGER, INTENT(OUT)         :: ifault
REAL (lsq_kind), INTENT(OUT) :: cormat(dimc), ycorr(ncol)

!     Local variables.

INTEGER           :: base_pos, pos, row, col, col1, col2, pos1, pos2
REAL (lsq_kind)   :: rms(in+1:ncol), sumxx, sumxy, sumyy, work(in+1:ncol)

!     Some checks.

ifault = 0
IF (in .LT. 0 .OR. in .GT. ncol-1) ifault = ifault + 4
IF (dimc .LT. (ncol-in)*(ncol-in-1)/2) ifault = ifault + 8
IF (ifault .NE. 0) RETURN

!     Base position for calculating positions of elements in row (IN+1) of R.

base_pos = in*ncol - (in+1)*(in+2)/2

!     Calculate 1/RMS of elements in columns from IN to (ncol-1).

IF (d(in+1) > zero) rms(in+1) = one / SQRT(d(in+1))
DO col = in+2, ncol
  pos = base_pos + col
  sumxx = d(col)
  DO row = in+1, col-1
    sumxx = sumxx + d(row) * r(pos)**2
    pos = pos + ncol - row - 1
  END DO ! row = in+1, col-1
  IF (sumxx > zero) THEN
    rms(col) = one / SQRT(sumxx)
  ELSE
    rms(col) = zero
    ifault = -col
```

```
      END IF ! (sumxx > zero)
    END DO ! col = in+1, ncol-1

    !     Calculate 1/RMS for the Y-variable

    sumyy = sserr
    DO row = in+1, ncol
      sumyy = sumyy + d(row) * rhs(row)**2
    END DO ! row = in+1, ncol
    IF (sumyy > zero) sumyy = one / SQRT(sumyy)

    !     Calculate sums of cross-products.
    !     These are obtained by taking dot products of pairs of columns of R,
    !     but with the product for each row multiplied by the row multiplier
    !     in array D.

    pos = 1
    DO col1 = in+1, ncol
      sumxy = zero
      work(col1+1:ncol) = zero
      pos1 = base_pos + col1
      DO row = in+1, col1-1
        pos2 = pos1 + 1
        DO col2 = col1+1, ncol
          work(col2) = work(col2) + d(row) * r(pos1) * r(pos2)
          pos2 = pos2 + 1
        END DO ! col2 = col1+1, ncol
        sumxy = sumxy + d(row) * r(pos1) * rhs(row)
        pos1 = pos1 + ncol - row - 1
      END DO ! row = in+1, col1-1

    !     Row COL1 has an implicit 1 as its first element (in column COL1)

      pos2 = pos1 + 1
      DO col2 = col1+1, ncol
        work(col2) = work(col2) + d(col1) * r(pos2)
        pos2 = pos2 + 1
        cormat(pos) = work(col2) * rms(col1) * rms(col2)
        pos = pos + 1
      END DO ! col2 = col1+1, ncol
      sumxy = sumxy + d(col1) * rhs(col1)
      ycorr(col1) = sumxy * rms(col1) * sumyy
    END DO ! col1 = in+1, ncol-1

    ycorr(1:in) = zero

    RETURN
    END SUBROUTINE partial_corr




    SUBROUTINE vmove(from, to, ifault)

    !     ALGORITHM AS274 APPL. STATIST. (1992) VOL.41, NO. 2

    !     Move variable from position FROM to position TO in an
    !     orthogonal reduction produced by AS75.1.
    !
    !------------------------------------------------------------------------
```

```
  IMPLICIT NONE
  INTEGER, INTENT(IN)            :: from, to
  INTEGER, INTENT(OUT)           :: ifault

  !     Local variables

  REAL (lsq_kind)   :: d1, d2, X, d1new, d2new, cbar, sbar, Y
  INTEGER           :: m, first, last, inc, m1, m2, mp1, col, pos, row

  !     Check input parameters

  ifault = 0
  IF (from .LT. 1 .OR. from .GT. ncol) ifault = ifault + 4
  IF (to .LT. 1 .OR. to .GT. ncol) ifault = ifault + 8
  IF (ifault .NE. 0) RETURN

  IF (from .EQ. to) RETURN

  IF (.NOT. rss_set) CALL ss

  IF (from .LT. to) THEN
    first = from
    last = to - 1
    inc = 1
  ELSE
    first = from - 1
    last = to
    inc = -1
  END IF

     DO 70 m = first, last, inc

  !     Find addresses of first elements of R in rows M and (M+1).

        m1 = (m-1)*(ncol+ncol-m)/2 + 1
        m2 = m1 + ncol - m
        mp1 = m + 1
        d1 = d(m)
        d2 = d(mp1)

  !     Special cases.

        IF (d1 .LT. vsmall .AND. d2 .LT. vsmall) GO TO 40
        X = r(m1)
        IF (ABS(x) * SQRT(d1) .LT. tol(mp1)) THEN
          X = zero
        END IF
        IF (d1 .LT. vsmall .OR. ABS(x) .LT. vsmall) THEN
          d(m) = d2
          d(mp1) = d1
          r(m1) = zero
          DO col = m+2, ncol
            m1 = m1 + 1
            X = r(m1)
            r(m1) = r(m2)
            r(m2) = x
            m2 = m2 + 1
          END DO ! col = m+2, ncol
          X = rhs(m)
          rhs(m) = rhs(mp1)
          rhs(mp1) = x
```

```
        GO TO 40
      ELSE IF (d2 .LT. vsmall) THEN
        d(m) = d1 * x**2
        r(m1) = one / x
        r(m1+1:m1+ncol-m-1) = r(m1+1:m1+ncol-m-1) / x
        rhs(m) = rhs(m) / x
        GO TO 40
      END IF
!
!     Planar rotation in regular case.
!
      d1new = d2 + d1*x**2
      cbar = d2 / d1new
      sbar = x * d1 / d1new
      d2new = d1 * cbar
      d(m) = d1new
      d(mp1) = d2new
      r(m1) = sbar
      DO col = m+2, ncol
        m1 = m1 + 1
        y = r(m1)
        r(m1) = cbar*r(m2) + sbar*y
        r(m2) = y - x*r(m2)
        m2 = m2 + 1
      END DO ! col = m+2, ncol
      y = rhs(m)
      rhs(m) = cbar*rhs(mp1) + sbar*y
      rhs(mp1) = y - x*rhs(mp1)

!     Swap columns M and (M+1) down to row (M-1).

40    IF (m .EQ. 1) GO TO 60
      pos = m
      DO row = 1, m-1
        x = r(pos)
        r(pos) = r(pos-1)
        r(pos-1) = x
        pos = pos + ncol - row - 1
      END DO ! row = 1, m-1

!     Adjust variable order (VORDER), the tolerances (TOL) and
!     the vector of residual sums of squares (RSS).

60    m1 = vorder(m)
      vorder(m) = vorder(mp1)
      vorder(mp1) = m1
      x = tol(m)
      tol(m) = tol(mp1)
      tol(mp1) = x
      rss(m) = rss(mp1) + d(mp1) * rhs(mp1)**2
70 CONTINUE

RETURN
END SUBROUTINE vmove



SUBROUTINE reordr(list, n, pos1, ifault)

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2
```

```
!     Re-order the variables in an orthogonal reduction produced by
!     AS75.1 so that the N variables in LIST start at position POS1,
!     though will not necessarily be in the same order as in LIST.
!     Any variables in VORDER before position POS1 are not moved.

!     Auxiliary routine called: VMOVE
!
!------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)         :: n, list(n), pos1
INTEGER, INTENT(OUT)        :: ifault

!     Local variables.

INTEGER        :: next, i, l, j

!     Check N.

   ifault = 0
   IF (n .LT. 1 .OR. n .GT. ncol+1-pos1) ifault = ifault + 4
   IF (ifault .NE. 0) RETURN

!     Work through VORDER finding variables which are in LIST.

   next = pos1
   i = pos1
10 l = vorder(i)
   DO j = 1, n
     IF (l .EQ. list(j)) GO TO 40
   END DO
30 i = i + 1
   IF (i .LE. ncol) GO TO 10

!     If this point is reached, one or more variables in LIST has not
!     been found.

   ifault = 8
   RETURN

!     Variable L is in LIST; move it up to position NEXT if it is not
!     already there.

40 IF (i .GT. next) CALL vmove(i, next, ifault)
   next = next + 1
   IF (next .LT. n+pos1) GO TO 30

   RETURN
END SUBROUTINE reordr



SUBROUTINE hdiag(xrow, nreq, hii, ifault)

!     ALGORITHM AS274  APPL. STATIST. (1992) VOL.41, NO. 2
!
!------------------------------------------------------------------------

IMPLICIT NONE
INTEGER, INTENT(IN)          :: nreq
INTEGER, INTENT(OUT)         :: ifault
```

```
  REAL (lsq_kind), INTENT(IN)  :: xrow(ncol)
  REAL (lsq_kind), INTENT(OUT) :: hii

  !     Local variables

  INTEGER             :: col, row, pos
  REAL (lsq_kind)     :: total, wk(ncol)

  !     Some checks

  ifault = 0
  IF (nreq .GT. ncol) ifault = ifault + 4
  IF (ifault .NE. 0) RETURN

  !     The elements of xrow.inv(R).sqrt(D) are calculated and stored
  !     in WK.

  hii = zero
  DO col = 1, nreq
    IF (SQRT(d(col)) .LE. tol(col)) THEN
      wk(col) = zero
    ELSE
      pos = col - 1
      total = xrow(col)
      DO row = 1, col-1
        total = total - wk(row)*r(pos)
        pos = pos + ncol - row - 1
      END DO ! row = 1, col-1
      wk(col) = total
      hii = hii + total**2 / d(col)
    END IF
  END DO ! col = 1, nreq

  RETURN
  END SUBROUTINE hdiag



  REAL (lsq_kind) FUNCTION varprd(x, nreq, var, ifault)

  !     Calculate the variance of x'b where b consists of the first nreq
  !     least-squares regression coefficients.
  !
  !------------------------------------------------------------------------

  IMPLICIT NONE
  INTEGER, INTENT(IN)          :: nreq
  INTEGER, INTENT(OUT)         :: ifault
  REAL (lsq_kind), INTENT(IN)  :: x(*)
  REAL (lsq_kind), INTENT(OUT) :: var

  !     Local variables

  INTEGER           :: row
  REAL (lsq_kind)   :: wk(nreq)

  !     Check input parameter values

  varprd = zero
  ifault = 0
  IF (nreq .LT. 1 .OR. nreq .GT. ncol) ifault = ifault + 4
```

```
  IF (nobs .LE. nreq) ifault = ifault + 8
  IF (ifault .NE. 0) RETURN

  !     Calculate the residual variance estimate.

  var = sserr / (nobs - nreq)

  !     Variance of x'b = var.x'(inv R)(inv D)(inv R')x
  !     First call BKSUB2 to calculate (inv R')x by back-substitution.

  CALL BKSUB2(x, wk, nreq)
  DO row = 1, nreq
    IF(d(row) .GT. tol(row)) varprd = varprd + wk(row)**2 / d(row)
  END DO

  varprd = varprd * var

  RETURN
  END FUNCTION varprd



  SUBROUTINE bksub2(x, b, nreq)

  !     Solve x = R'b for b given x, using only the first nreq rows and
  !     columns of R, and only the first nreq elements of R.
  !
  !-----------------------------------------------------------------------

  IMPLICIT NONE
  INTEGER, INTENT(IN)          :: nreq
  REAL (lsq_kind), INTENT(IN)  :: x(nreq)
  REAL (lsq_kind), INTENT(OUT) :: b(nreq)

  !     Local variables

  INTEGER           :: pos, row, col
  REAL (lsq_kind)   :: temp

  !     Solve by back-substitution, starting from the top.

  DO row = 1, nreq
    pos = row - 1
    temp = x(row)
    DO col = 1, row-1
      temp = temp - r(pos)*b(col)
      pos = pos + ncol - col - 1
    END DO
    b(row) = temp
  END DO

  RETURN
  END SUBROUTINE bksub2


  END MODULE lsq

  !-----------------------------------------------------------------------
  !-----------------------------------------------------------------------

  (File: DEMO.F90)
```

```
PROGRAM demo

!     Program to demonstrate the use of module LSQ for unconstrained
!     linear least-squares regression.
!     The data on fuel consumption are from:
!     Sanford Weisberg `Applied Linear Regression', 2nd edition, 1985,
!     pages 35-36.   Publisher: Wiley

!     The program is designed so that users can easily edit it for their
!     problems.   If you are likely to have more than 500 cases then increase
!     the value of maxcases below.   Similarly, if you may have more than
!     30 predictor variables, change maxvar below.

!     The subscript 0 is used to relate to the constant in the model.
!     Module LSQ treats the constant in the model, if there is one, just as
!     any other variable.   The numbering of all arrays starts from 1 in LSQ.

USE lsq                       ! Must be the first declaration in the program
IMPLICIT NONE

INTEGER, PARAMETER :: maxcases = 500, maxvar = 30,                            &
                      max_cdim = maxvar*(maxvar+1)/2, lout = 6
INTEGER            :: case, nvar, iostatus, nreq, ifault, i, list(maxvar), j, &
                      in
REAL ( lsq_kind )  :: xx(0:maxvar), yy, wt, one = 1.D0, beta(0:maxvar),       &
                      var, covmat(max_cdim), sterr(0:maxvar), hii,            &
                      cormat(max_cdim), ycorr(maxvar)
REAL (KIND(0.E0))  :: x(maxcases, maxvar), y(maxcases), t(0:maxvar), tmin,    &
                      r2, resid(maxcases), std_resid(maxcases), std_err_pred, &
                      fitted, stdev_res
CHARACTER          :: heading*80, vname(0:maxvar)*8, state(50)*2, y_name*8, key
LOGICAL            :: fit_const, lindep(0:maxvar)

WRITE(*, *)'The data on fuel consumption are from:'
WRITE(*, *)'Sanford Weisberg "Applied Linear Regression", 2nd edition, 1985,'
WRITE(*, *)'pages 35-36.   Publisher: Wiley   ISBN: 0-471-87957-6'
WRITE(*, *)

OPEN(8, file='fuelcons.dat', status='old')

!     The first line of this file contains the names of the variables.

READ(8, '(a)') heading

vname(0) = 'Constant'

!     Read the heading to get the variable names & the number of variables

CALL separate_text(heading, vname(1), nvar)

nvar = nvar - 2               ! nvar is the number of variables.
                              ! 1 is subtracted as the first field in this
                              ! file contains the abbreviated state name,
                              ! & 1 is subtracted for the Y-variable.
vname(1:nvar) = vname(2:nvar+1)
y_name = vname(nvar+2)

WRITE(*, *)'No. of variables =', nvar
WRITE(*, *)'Predictor variables are:'
WRITE(*, '(1x, 9a9)') vname(1:nvar)
WRITE(*, *)'Dependent variable is: ', y_name
```

```
fit_const = .true.           ! Change to .false. if fitting a model without
                             ! a constant.

CALL startup(nvar, fit_const)          ! Initializes the QR-factorization

!     Read in the data, one line at a time, and progressively update the
!     QR-factorization.

wt = one
case = 1

DO
  READ(8, *, IOSTAT=iostatus) state(case), x(case, 1:nvar), y(case)
  IF (iostatus > 0) CYCLE              ! Error in data
  IF (iostatus < 0) EXIT               ! End of file

  xx(0) = one                          ! A one is inserted as the first
                                       ! variable if a constant is being fitted.
  xx(1:nvar) = x(case, 1:nvar)         ! New variables and transformed variables
                                       ! will often be generated here.
  yy = y(case)
  CALL includ(wt, xx, yy)
  case = case + 1
END DO

WRITE(*, *)'No. of observations =', nobs

CALL sing(lindep, ifault)              ! Checks for singularities

IF (ifault == 0) THEN
  WRITE(*, *)'QR-factorization is not singular'
ELSE
  DO i = 1, nvar
    IF (lindep(i)) THEN
      WRITE(*, *) vname(i), ' is exactly linearly related to earlier variables'
    END IF
  END DO ! i = 1, nvar
END IF ! (ifault == 0)
WRITE(*, *)

!     Show correlations (IN = 1 for `usual' correlations)

in = 1
CALL partial_corr(in, cormat, max_cdim, ycorr, ifault)
CALL printc(in, cormat, max_cdim, ycorr, vname, y_name, 1, lout, ifault)
WRITE(*, *)
WRITE(*, *)'Press ENTER to continue'
READ(*, '(a)') key

!     Weisberg only uses variables TAX, INC, ROAD and DLIC.   These are
!     currently in positions 2, 4, 5 & 7.
!     We could have set NVAR = 4 and copied only these variables from X to
!     XX, but we will show how regressions can be performed for a subset
!     of the variables in the QR-factorization.   Routine REORDR will be used
!     to re-order the variables.

list(1:4) = (/ 2, 4, 5, 7/)
CALL reordr(list, 4, 2, ifault)        ! Re-order so that the first 4 variables
                                       ! appear in positions 2,3,4 & 5.
                                       ! N.B. Though variables # 2, 4, 5 & 7
```

```
                                        !      will occupy positions 2-5, they
                                        !      may be in any order.
                                        ! The constant remains in position 1.
  WRITE(*, 910) (vname(vorder(1:ncol)))
  910 FORMAT(1x, 'Current order of variables:'/1x, 8a9/)

  !     Calculate regression coefficients of Y against the first variable, which
  !     was just a constant = 1.0, and the next 4 predictors.

  CALL tolset                           ! Calculate tolerances before calling
                                        ! subroutine regcf.
  nreq = 5                              ! i.e. Const, TAX, INC, ROAD, DLIC
  CALL regcf(beta, nreq, ifault)

  CALL ss                               ! Calculate residual sums of squares

  !     Calculate covariance matrix of the regression coefficients & their
  !     standard errors.

  CALL cov(nreq, var, covmat, max_cdim, sterr, ifault)

  !     Calculate t-values

  t(0:nreq-1) = beta(0:nreq-1) / sterr(0:nreq-1)

  !     Output regression table, residual sums of squares, and R-squared.

  WRITE(*, *)
  WRITE(*, *)'Variable   Regn.coeff.   Std.error  t-value   Res.sum of sq.'
  DO i = 0, nreq-1
    WRITE(*, 900) vname(vorder(i+1)), beta(i), sterr(i), t(i), rss(i+1)
    900 FORMAT(1x, a8, 2x, g12.4, 2x, g11.4, 1x, f7.2, 2x, g14.6)
  END DO
  WRITE(*, *)

  !     Now delete the variable with the smallest t-value by moving it
  !     to position 5 and then repeating the calculations for the constant
  !     and the next 3 variables.

  j = 1
  tmin = ABS(t(1))
  DO i = 2, nreq-1
    IF (ABS(t(i)) < tmin) THEN
      j = i
      tmin = ABS(t(i))
    END IF
  END DO
  j = j + 1                    ! Add 1 as the t-array started at subscript 0

  WRITE(*, *)'Removing variable in position', j
  CALL vmove(j, nreq, ifault)
  nreq = nreq - 1
  CALL regcf(beta, nreq, ifault)
  CALL ss
  CALL cov(nreq, var, covmat, max_cdim, sterr, ifault)
  t(0:nreq-1) = beta(0:nreq-1) / sterr(0:nreq-1)

  !     Output regression table, residual sums of squares, and R-squared.

  WRITE(*, *)
  WRITE(*, *)'Variable   Regn.coeff.   Std.error  t-value   Res.sum of sq.'
```

```
  DO i = 0, nreq-1
    WRITE(*, 900) vname(vorder(i+1)), beta(i), sterr(i), t(i), rss(i+1)
  END DO
  WRITE(*, *)
  stdev_res = SQRT( rss(nreq)/(nobs - nreq) )
  r2 = one - rss(nreq) / rss(1)          ! RSS(1) is the rss if only the constant
                                         ! is fitted; RSS(nreq) is the rss after
                                         ! fitting the requested NREQ variables.
  WRITE(*, '(1x, "R^2 =", f8.4, 5x, "Std. devn. of residuals =", g12.4/)')      &
        r2, stdev_res
  WRITE(*, *)'N.B. Some statistical packages wrongly call the standard deviation'
  WRITE(*, *)'     of the residuals the standard error of prediction'
  WRITE(*, *)

  !     Calculate residuals, hii, standardized residuals & standard errors of
  !     prediction.

  WRITE(*, *)'Press ENTER to continue'
  READ(*, '(a)') key

  WRITE(*, *)'State     Actual    Fitted  Residual  Std.resid.  SE(prediction)'
  DO i = 1, nobs
    xx(0) = one
    DO j = 1, nreq-1
      xx(j) = x(i, vorder(j+1))          ! N.B. Regression coefficient j is for
                                         !      the variable vorder(j+1)
    END DO
    fitted = DOT_PRODUCT( beta(0:nreq-1), xx(0:nreq-1) )
    resid(i) = y(i) - fitted
    CALL hdiag(xx, nreq, hii, ifault)
    std_resid(i) = resid(i) / SQRT(var*(one - hii))
    std_err_pred = varprd(xx, nreq, var, ifault)
    WRITE(*, 920) state(i), y(i), fitted, resid(i), std_resid(i), std_err_pred
    920 FORMAT(3x, a2, 2x, 3f10.1, f9.2, 5x, f10.0)

    IF (i .EQ. 24) THEN
      WRITE(*, *)'Press ENTER to continue'
      READ(*, '(a)') key
      WRITE(*, *)'State     Actual    Fitted  Residual  Std.resid.
  SE(prediction)'
    END IF
  END DO ! i = 1, nobs

  END PROGRAM demo



  SUBROUTINE separate_text(text, name, number)

  !     Takes the character string in `text' and separates it into `names'.
  !     This version only allows spaces as delimiters.

  IMPLICIT NONE
  CHARACTER (LEN = *), INTENT(INOUT)  :: text
  CHARACTER (LEN = *), INTENT(OUT)    :: name(*)
  INTEGER, INTENT(OUT)                :: number

  !     Local variables

  INTEGER  :: length, len_name, pos1, pos2, nchars
```

```
  length = LEN_TRIM(text)
  number = 0
  IF (length == 0) RETURN

  len_name = LEN(name(1))
  IF (len_name < 1) RETURN

  pos1 = 1
  DO
    DO                                  ! Remove any leading blanks
      IF (text(pos1:pos1) == ' ') THEN
        text(pos1:) = text(pos1+1:)
        length = length - 1
        CYCLE
      ELSE
        EXIT
      END IF
    END DO
    pos2 = INDEX(text(pos1:length), ' ') ! Find position of next blank
    IF (pos2 > 0) THEN
      pos2 = pos2 + pos1 - 1
    ELSE                                ! Last name, no blank found
      pos2 = length + 1
    END IF
    number = number + 1
    nchars = MIN(len_name, pos2-pos1)
    name(number) = text(pos1:pos1+nchars-1)
    pos1 = pos2 + 1                     ! Move past the blank
    IF (pos1 > length) RETURN
  END DO

  END SUBROUTINE separate_text



  SUBROUTINE printc(in, cormat, dimc, ycorr, vname, yname, iopt, lout, ier)

  !     Print (partial) correlations calculated using partial_corr to unit lout.
  !     If IOPT = 0, print correlations with the Y-variable only.

  USE lsq

  IMPLICIT NONE
  INTEGER, INTENT(IN)           :: in, dimc, iopt, lout
  INTEGER, INTENT(OUT)          :: ier
  REAL ( lsq_kind ), INTENT(IN) :: cormat(dimc), ycorr(ncol)
  CHARACTER, INTENT(IN)         :: vname(0:ncol-1)*8, yname*8

  !     Local variables.

  INTEGER            :: nrows, j1, j2, i1, i2, row, upos, tpos, last
  CHARACTER          :: text*74, empty*65 = ' ', char1*9 = ' 1.0'

  !     Check validity of arguments

  ier = 0
  IF (in .GE. ncol) ier = 1
  IF (ncol .LE. 1) ier = ier + 2
  nrows = ncol - in
  IF (dimc .LE. nrows*(nrows-1)/2) ier = ier + 4
  IF (ier .NE. 0) RETURN
```

```
!     If iopt.NE.0 output heading

IF (iopt .EQ. 0) GO TO 30
WRITE(lout, 900)
900 FORMAT(/5x, 'Correlation matrix')
j1 = in + 1

DO
  j2 = MIN(j1+6, ncol)
  i1 = j1 - in
  i2 = j2 - in
  WRITE(lout, 910) vname(vorder(j1:j2))
  910 FORMAT(11X, 7(a8, 1x))

!     Print correlations for rows 1 to i2, columns i1 to i2.

  DO row = 1, i2
    text = ' ' // vname(vorder(row+in)) // empty
    IF (i1 .GT. row) THEN
      upos = (row-1) * (nrows+nrows-row) /2 + (i1-row)
      last = upos + i2 - i1
      WRITE(text(12:74), '(7(F8.5, 1x))') cormat(upos:last)
    ELSE
      upos = (row-1) * (nrows+nrows-row) /2 + 1
      tpos = 12 + 9*(row-i1)
      text(tpos:tpos+8) = char1
      last = upos + i2 - row - 1
      IF (row .LT. i2) WRITE(text(tpos+9:74), '(6(F8.5, 1X))') cormat(upos:last)
    END IF
    WRITE(lout, '(a)') text
  END DO ! row = 1, i2

!     Move onto the next block of columns.

  j1 = j2 + 1
  IF (j1 .GT. ncol) EXIT
END DO

!     Correlations with the Y-variable.

30 WRITE(lout, 920) yname
   920 FORMAT(/5x, 'Correlations with the dependent variable: ', a)
   j1 = in + 1
DO
  j2 = MIN(j1+7, ncol)
  WRITE(lout, 930) vname(vorder(j1:j2))
  930 FORMAT(/1X, 8(A8, 1X))
  WRITE(lout, 940) ycorr(j1:j2)
  940 FORMAT(1X, 8(F8.5, 1X))
  j1 = j2 + 1
  IF (j1 .GT. ncol) EXIT
END DO

END SUBROUTINE printc

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

(File: FUELCONS.DAT)
State Popn     Tax     NLic  Income  RoadMls FuelCon   DLic  Fuel_Pop
```

ME	1029.	9.00	540.	3.571	1.976	557.	52.5	541.
NH	771.	9.00	441.	4.092	1.250	404.	57.2	524.
VT	462.	9.00	268.	3.865	1.586	259.	58.0	561.
MA	5787.	7.50	3060.	4.870	2.351	2396.	52.9	414.
RI	968.	8.00	527.	4.399	.431	397.	54.4	410.
CN	3082.	10.00	1760.	5.342	1.333	1408.	57.1	457.
NY	18366.	8.00	8278.	5.319	11.868	6312.	45.1	344.
NJ	7367.	8.00	4074.	5.126	2.138	3439.	55.3	467.
PA	11926.	8.00	6312.	4.447	8.577	5528.	52.9	464.
OH	10783.	7.00	5948.	4.512	8.507	5375.	55.2	498.
IN	5291.	8.00	2804.	4.391	5.939	3068.	53.0	580.
IL	11251.	7.50	5903.	5.126	14.186	5301.	52.5	471.
MI	9082.	7.00	5213.	4.817	6.930	4768.	57.4	525.
WI	4520.	7.00	2465.	4.207	6.580	2294.	54.5	508.
MN	3896.	7.00	2368.	4.332	8.159	2204.	60.8	566.
IA	2883.	7.00	1689.	4.318	10.340	1830.	58.6	635.
MO	4753.	7.00	2719.	4.206	8.508	2865.	57.2	603.
ND	632.	7.00	341.	3.718	4.725	451.	54.0	714.
SD	579.	7.00	419.	4.716	5.915	501.	72.4	865.
NE	1525.	8.50	1033.	4.341	6.010	976.	67.7	640.
KS	2258.	7.00	1496.	4.593	7.834	1466.	66.3	649.
DE	565.	8.00	340.	4.983	.602	305.	60.2	540.
MD	4056.	9.00	2073.	4.897	2.449	1883.	51.1	464.
VA	4764.	9.00	2463.	4.258	4.686	2604.	51.7	547.
WV	1781.	8.50	982.	4.574	2.619	819.	55.1	460.
NC	5214.	9.00	2835.	3.721	4.746	2953.	54.4	566.
SC	2665.	8.00	1460.	3.448	5.399	1537.	54.8	577.
GA	4720.	7.50	2731.	3.846	9.061	2979.	57.9	631.
FL	7259.	8.00	4084.	4.188	5.975	4169.	56.3	574.
KY	3299.	9.00	1626.	3.601	4.650	1761.	49.3	534.
TN	4031.	7.00	2088.	3.640	6.905	2301.	51.8	571.
AL	3510.	7.00	1801.	3.333	6.594	1946.	51.3	554.
MS	2263.	8.00	1309.	3.063	6.524	1306.	57.8	577.
AR	1978.	7.50	1081.	3.357	4.121	1242.	54.7	628.
LA	3720.	8.00	1813.	3.528	3.495	1812.	48.7	487.
OK	2634.	6.58	1657.	3.802	7.834	1695.	62.9	644.
TX	11649.	5.00	6595.	4.045	17.782	7451.	56.6	640.
MT	719.	7.00	421.	3.897	6.385	506.	58.6	704.
ID	756.	8.50	501.	3.635	3.274	490.	66.3	648.
WY	345.	7.00	232.	4.345	3.905	334.	67.2	968.
CO	2357.	7.00	1475.	4.449	4.639	1384.	62.6	587.
NM	1065.	7.00	600.	3.656	3.985	744.	56.3	699.
AZ	1945.	7.00	1173.	4.300	3.635	1230.	60.3	632.
UT	1126.	7.00	572.	3.745	2.611	666.	50.8	591.
NV	527.	6.00	354.	5.215	2.302	412.	67.2	782.
WN	3443.	9.00	1966.	4.476	3.942	1757.	57.1	510.
OR	2182.	7.00	1360.	4.296	4.083	1331.	62.3	610.
CA	20468.	7.00	12130.	5.002	9.794	10730.	59.3	524.

```
!-------------------------------------------------------------------------
!-------------------------------------------------------------------------

(File: TEST1.F90)
PROGRAM test1

!     Test vmove.
!     The LS fit to the data is:  Y = 1 + X + X**2 + X**3
!     i.e. all of the regression coefficients should be exactly 1.0.
!     An extra variable equal to X + X**2 is inserted to give a singularity.
```

```
USE lsq
IMPLICIT NONE

REAL ( lsq_kind ) :: x(11,4), y(11), xrow(5), yelem, beta(5), one = 1.D0
INTEGER           :: nvar, i, j, ifault
LOGICAL           :: fit_const, lindep(5)
CHARACTER (LEN=1) :: key

DATA y/65647., 70638., 75889., 81399., 87169., 93202., 99503., &
       106079., 112939., 120094., 127557./

WRITE(*, *)'Fitting nasty cubic'
WRITE(*, *)'1st 4 regression coefficients should equal 1.0'
WRITE(*, *)'Last variable = X + X^2, to introduce a deliberate singularity'
WRITE(*, *)

DO i = 1, 11
  x(i,1) = i + 39
  x(i,2) = x(i,1)**2
  x(i,3) = x(i,1) * x(i,2)
  x(i,4) = x(i,1) + x(i,2)
END DO

!     Use includ to form the orthogonal reduction.

nvar = 4
fit_const = .true.
CALL startup(nvar, fit_const)

DO i = 1, 11
  xrow(1) = one
  DO j = 1, 4
    xrow(j+1) = x(i,j)
  END DO
  yelem = y(i)
  CALL includ(one, xrow, yelem)
END DO

!     CALL tolset to set tolerances, then sing to detect singularities.

CALL tolset
CALL sing(lindep, ifault)
WRITE(*, *)'From routine SING, IFAULT =', ifault
WRITE(*, *)'Array LINDEP:', lindep
WRITE(*, *)
WRITE(*, *)'sserr = ', sserr, '   Should be 286.000'

!     Calculate residual sums of squares (RSS).

      CALL ss

!     Calculate regression coefficients, using vmove to cycle through
!     the order of the variables.

    DO i = 1, 6
      CALL regcf(beta, 5, ifault)
      WRITE(*, 920) vorder
920   FORMAT(1x, 'Variable order:'/ 1x, i10, 4i15)
      WRITE(*, 900) beta
900   FORMAT(' Regn. coeffs.:'/ 4x, 5g15.7)
      WRITE(*, 910) d, r, rhs, rss
```

```
  910   FORMAT(' d: ', 5g15.7/ ' r: ', 4g15.7/19x,3g15.7/34x,2g15.7/ &
           49x,g15.7/ ' rhs: '/4x, 5g15.7/ ' rss: '/4x, 5g15.7//)

  WRITE(*, *)'Press ENTER to continue'
  READ(*, '(a)') key

  !     Move variable in 1st position to the last.

        CALL vmove(1, 5, ifault)
      END DO
  END PROGRAM test1
```

```
      REAL FUNCTION CHI2NC(X, F, THETA, IFAULT)
C
C       ALGORITHM AS 275 APPL.STATIST. (1992), VOL.41, NO.2
C
C       Computes the noncentral chi-square distribution function
C       with positive real degrees of freedom f and nonnegative
C       noncentrality parameter theta
C
      REAL X, F, THETA
      INTEGER IFAULT
C
      INTEGER ITRMAX
      LOGICAL FLAG
      REAL ERRMAX, ZERO, ONE, TWO, LAM, N, U, V, X2, F2, T, TERM,
     *    BOUND, ALOGAM
C
      EXTERNAL ALOGAM
C
      DATA ERRMAX, ITRMAX / 1.0E-6, 50 /
      DATA ZERO, ONE, TWO / 0.0, 1.0, 2.0 /
C
      CHI2NC = X
      IFAULT = 2
      IF (F .LE. ZERO .OR. THETA .LT. ZERO) RETURN
      IFAULT = 3
      IF (X .LT. ZERO) RETURN
      IFAULT = 0
      IF (X .EQ. ZERO) RETURN
      LAM = THETA / TWO
C
C       Evaluate the first term
C
      N = ONE
      U = EXP(-LAM)
      V = U
      X2 = X / TWO
      F2 = F / TWO
      T = X2 ** F2 * EXP(-X2) / EXP(ALOGAM((F2 + ONE), IFAULT))
C
C       There is no need to test IFAULT si
C       already been checked
C
      TERM = V * T
      CHI2NC = TERM
C
C       Check if (f+2n) is greater than x
C
      FLAG = .FALSE.
   10 IF ((F + TWO * N - X) .LE. ZERO) GO TO 30
C
C       Find the error bound and check for convergence
C
      FLAG = .TRUE.
   20 BOUND = T * X / (F + TWO * N - X)
      IF (BOUND .GT. ERRMAX .AND. INT(N) .LE. ITRMAX) GO TO 30
      IF (BOUND .GT. ERRMAX) IFAULT = 1
      RETURN
C
C       Evaluate the next term of the expansion and then the
C       partial sum
C
```

```
      30 U = U * LAM / N
         V = V + U
         T = T * X / (F + TWO * N)
         TERM = V * T
         CHI2NC = CHI2NC + TERM
         N = N + ONE
         IF (FLAG) GO TO 20
         GO TO 10
   C
         END
```

```
      SUBROUTINE CCLASS (DD, DI, N1, N2, NP, KMAX, A1, COND, RAW, DATA,
     *                   SX, SY, TMV, CUT, ICC, PQ, PMIN, LKA, Z, UPMQM,
     *                   ICR, CGFE, IFAULT)
C
C        ALGORITHM AS 276.1 APPL.STATIST. (1992), VOL.41, NO.2
C
C        Obtains the optimal combinatoric classification procedure for
C        the two normal problem
C
      INTEGER DD, DI, N1, N2, NP, KMAX, COND, RAW, ICC(NP), LKA(NP + 2),
     *        ICR(2, NP), IFAULT
      REAL A1, DATA(DD, NP), SX(DI, NP), SY(DI, NP), TMV(DI, 2),
     *     CUT(NP), PQ(2, NP), PMIN, Z(DI, NP + 1), UPMQM(3, NP *
     *     (NP + 1)/2), CGFE(DI, 3 * NP + 1)
C
      EXTERNAL DIAG2, ERPROB, MCE, SWAP
C
      INTEGER HOLD, I, IFLT, IZ, J, JP1, K, MX, MY, NDEN1, NDEN2, N1N2,
     *        N1P1
      REAL ONE, SSQ, STORE, SUM, XYMEAN, ZERO
C
      PARAMETER (ZERO = 0.0E0, ONE = 1.0E0)
C
C        Initialise local variables
C
      IZ = NP + 1
      N1N2 = N1 + N2
      MX = N1N2 + 1
      MY = MX + 1
      N1P1 = N1 + 1
C
C        Check for input failures
C

      IFAULT = 0
      IF (N1 .LT. 0 .OR. N2 .LT. 0 .OR. MY .GT. DD)
     *   IFAULT = 1
      IF (NP .LT. 2 .OR. NP .GT. DI) IFAULT = IFAULT + 2
      IF (KMAX .LT. 1 .OR. KMAX .GT. NP) IFAULT = IFAULT + 4
      IF (A1 .LE. ZERO .OR. A1 .GE. ONE) IFAULT = IFAULT + 8
      IF (COND .LT. 1 .OR. COND .GT. 4) IFAULT = IFAULT + 16
      IF (RAW .LT. 0 .OR. RAW .GT. 1) IFAULT = IFAULT + 32
      IF (RAW .NE. 0 .AND. COND .EQ. 1) IFAULT = IFAULT + 64
      IF (IFAULT .GT. 0) RETURN
C
C        Check for known population parameter
C
      IF (RAW .EQ. 1) THEN
         NDEN1 = N1
         NDEN2 = N2
         IF (COND.NE.2) THEN
C
C        Estimation of individual mean vector
C
              DO 30 J = 1, NP
                    SUM = ZERO
                 DO 10 I = 1, N1
                    SUM = SUM + DATA(I, J)
   10            CONTINUE
                 DATA(MX, J) = SUM / N1
                 SUM = ZERO
```

```
                  DO 20 I = N1P1, N1N2
                     SUM = SUM + DATA(I, J)
   20             CONTINUE
                  DATA(MY, J) = SUM / N2
   30          CONTINUE
C
C        Estimation of covariance matrices
C
               NDEN1 = N1 - 1
               NDEN2 = N2 - 1
            ENDIF
            DO 60 J = 1, NP
               XYMEAN = DATA(MX, J)
               DO 40 I = 1, N1
                  DATA(I, J) = DATA(I, J) - XYMEAN
   40          CONTINUE
               XYMEAN = DATA(MY, J)
               DO 50 I = N1P1, N1N2
                  DATA(I, J) = DATA(I, J) - XYMEAN
   50          CONTINUE
   60       CONTINUE
            DO 100 J = 1, NP
               DO 90 K = 1, J
                  SSQ = ZERO
                  DO 70 I = 1, N1
                     SSQ = SSQ + DATA(I, J) * DATA(I, K)
   70             CONTINUE
                  SX(J, K) = SSQ / NDEN1
                  SSQ = ZERO
                  DO 80 I = N1P1, N1N2
                     SSQ = SSQ + DATA(I, J) * DATA(I, K)
   80             CONTINUE
                  SY(J, K) = SSQ / NDEN2
   90          CONTINUE
  100       CONTINUE
            DO 120 J = 2, NP
               DO 110 K = 1, J - 1
                  SX(K, J) = SX(J, K)
                  SY(K, J) = SY(J, K)
  110          CONTINUE
  120       CONTINUE
            IF (COND .EQ. 3) THEN
C
C        Estimation of common mean vector
C
              DO 130 J = 1, NP
                 DATA(MX, J) = (N1 * DATA(MX, J) + N2 * DATA(MY, J)) /
     *                         N1N2
                 DATA(MY, J) = DATA(MX, J)
   130       CONTINUE
          ENDIF
       ENDIF
C
C         Simultaneous diagonalisation of the covariance matrices
C
       CALL DIAG2 (DI, NP, SX, SY, TMV(1, 2), Z, CGFE(1, 1),
     *    CGFE(1, NP+1), CGFE(1, 2*NP+1), CGFE(1, 3*NP+1), IFLT)
       IF (IFLT .GT. 0) THEN
           IFAULT = 128
           RETURN
       ENDIF
```

```
C
C         Compute population 2 mean vector (transformed)
C
      IF (COND .EQ. 3) THEN
         DO 140 I = 1, NP
            TMV(I, 1) = ZERO
  140    CONTINUE
      ELSE
         DO 160 I = 1, NP
            STORE = ZERO
            DO 150 J = 1, NP
               STORE = Z(J, I) * (DATA(MY, J) - DATA(MX, J)) + STORE
  150       CONTINUE
            TMV(I, 1) = STORE
  160    CONTINUE
      ENDIF
C
C         Compute Z' * XBAR
C
      DO 180 I = 1, NP
         STORE = ZERO
         DO 170 J = 1, NP
            STORE = STORE + Z(J, I) * DATA(MX, J)
  170    CONTINUE
         Z(I, IZ) = STORE
  180 CONTINUE
C
C         Determination of marginal misclassification probabilities
C
      CALL ERPROB (DI, NP, TMV, A1, PQ, ICC, CUT)
C
C          Re-indexing the variables such that the values of p are
C          increasing
C
      DO 210 I = 1, NP - 1
         DO 200 J = 1, NP - I
            JP1 = J + 1
            IF ( (PQ(1, J) .EQ. PQ(1, JP1) .AND. PQ(2, J) .GT.
     *           PQ(2, JP1)) .OR. PQ(1, J) .GT. PQ(1, JP1)) THEN
               CALL SWAP (PQ(1, J), PQ(1, JP1))
               CALL SWAP (PQ(2, J), PQ(2, JP1))
               CALL SWAP (CUT(J), CUT(JP1))
               HOLD = ICC(J)
               ICC(J) = ICC(JP1)
               ICC(JP1) = HOLD
               CALL SWAP (TMV(J, 2), TMV(JP1, 2))
               CALL SWAP (TMV(J, 1), TMV(JP1, 1))
               DO 190 K = 1, NP
                  CALL SWAP (Z(K, J), Z(K, JP1))
  190          CONTINUE
               CALL SWAP (Z(J, IZ), Z(JP1, IZ))
            ENDIF
  200    CONTINUE
  210 CONTINUE
C
C         Determination of the optimal joint classification rule
C
      CALL MCE (NP, KMAX, A1, PQ, PMIN, LKA, UPMQM, ICR, IFAULT)
      END

      SUBROUTINE DIAG2 (DI, NP, SX, SY, D, Z, C, G, F, E, IFAULT)
```

```
C
C       ALGORITHM AS 276.2 APPL.STATIST. (1992), VOL.41, NO.2
C
C       Simultaneously diagonalizes two covariance matrices such that
C       Z'(SX)Z = 1 and Z'(SY)Z = D where D is a diagonal matrix
C
      INTEGER DI, NP, IFAULT
      REAL SX(DI, NP), SY(DI, NP), D(NP), Z(DI, NP), C(DI, NP),
     *     G(DI, NP), F(DI, NP), E(NP)
C
      EXTERNAL TQL2, TRED2
C
      INTRINSIC SQRT
C
      INTEGER I, IFLT, J, K
      REAL ROOTD, SUM, ZERO
      PARAMETER (ZERO = 0.0E0)
C
      IFAULT = 0
C
C       Find G such that G' (SX) G = I
C
      CALL TRED2 (DI, NP, SX, D, E, G)
      CALL TQL2 (DI, NP, D, E, G, IFLT)
      IF (IFLT .NE. 0) THEN
         IFAULT = 1
         RETURN
      ENDIF
      DO 10 I = 1, NP
         IF (D(I) .LE. ZERO) THEN
            IFAULT = 2
            RETURN
         ENDIF
   10 CONTINUE
      DO 30 J = 1, NP
         ROOTD = SQRT(D(J))
         DO 20 I = 1, NP
            G(I, J) = G(I, J) / ROOTD
   20    CONTINUE
   30 CONTINUE
C
C       Create matrix F = G' (SY) G
C
      DO 60 I = 1, NP
         DO 50 J = 1, NP
            SUM = ZERO
            DO 40 K = 1, NP
               SUM = SY(I, K) * G(K, J) + SUM
   40       CONTINUE
            Z(I, J) = SUM
   50    CONTINUE
   60 CONTINUE
      DO 90 I = 1, NP
        DO 80 J = 1, NP
           SUM = ZERO
           DO 70 K = 1, NP
              SUM = Z(K, I) * G(K, J) + SUM
   70       CONTINUE
           F(I, J) = SUM
   80    CONTINUE
   90 CONTINUE
```

```
C
C        Find C such that C' F C = D
C
      CALL TRED2 (DI, NP, F, D, E, C)
      CALL TQL2 (DI, NP, D, E, C, IFLT)
      IF (IFLT .NE. 0) THEN
         IFAULT = 3
         RETURN

      ENDIF
      DO 100 I = 1, NP
         IF (D(I) .LE. ZERO) THEN
            IFAULT = 4
            RETURN
         ENDIF
  100 CONTINUE
C
C        Create matrix Z = GC
C
      DO 130 I = 1, NP
         DO 120 J = 1, NP
            SUM = ZERO
            DO 110 K = 1, NP
               SUM = G(I, K) * C(K, J) + SUM
  110       CONTINUE
            Z(I, J) = SUM
  120    CONTINUE
  130 CONTINUE
      RETURN
      END

      SUBROUTINE ERPROB (DI, N, TMV, A1, PQ, ICC, CUT)
C
C        ALGORITHM AS 276.3 APPL.STATIST. (1992), VOL.41, NO.2
C
C        Gives the classification rules and misclassification
C        probabilities for each of the marginal classification
C        procedures in normal population combinatoric classification
C
      INTEGER DI, N, ICC(N)
      REAL TMV(DI, 2), A1, PQ(2, N), CUT(N)
C
      REAL ALNORM, CHISQ1
C
      EXTERNAL ALNORM, CHISQ1
C
      INTRINSIC LOG, SQRT
C
      INTEGER I
      REAL B, G, H, HALF, ONE, S, T, TWO, VU, ZERO
C
      PARAMETER (ONE = 1.0E0, ZERO = 0.0E0, TWO = 2.0E0, HALF = 0.5E0)
C
C        Misclassification probabilities for cases 1 and 2
C
      B = ((ONE - A1) / A1) ** 2
      DO 10 I = 1, N
         S = TMV(I, 2)
         T = TMV(I, 1)
         CUT(I) = ZERO
         IF (S .NE. ONE) THEN
```

```
               G = S * LOG(B / S) / (ONE - S) + (T / (ONE - S)) ** 2 * S
               IF (G .GT. ZERO) THEN
                  VU = T / (S - ONE)
                  CUT(I) = G
                  ICC(I) = 1
                  PQ(1, I) = ONE - CHISQ1(G, VU)
                  G = G / S
                  VU = (VU + T) / SQRT(S)
                  PQ(2, I) = CHISQ1(G, VU)
                  IF (S .LT. ONE) THEN
                     ICC(I) = 2
                     PQ(1, I) = ONE - PQ(1, I)
                     PQ(2, I) = ONE - PQ(2, I)
                  ENDIF
C
C        Misclassification probabilities for cases 3 and 4
C
               ELSE IF (S .GT. ONE) THEN
                  ICC(I) = 3
                  PQ(1, I) = ONE
                  PQ(2, I) = ZERO
               ELSE
                  ICC(I) = 4
                  PQ(1, I) = ZERO
                  PQ(2, I) = ONE
               ENDIF
C
C        Misclassification probabilities for cases 5 and 6
C
               ELSE IF (T .EQ. ZERO) THEN
                  IF (A1 .LT. HALF) THEN
                     ICC(I) = 5
                     PQ(1, I) = ONE
                     PQ(2, I) = ZERO
                  ELSE
                     ICC(I) = 6
                     PQ(1, I) = ZERO
                     PQ(2, I) = ONE
                  ENDIF
C
C        Misclassification probabilities for cases 7 and 8
C
               ELSE
                  H = T / TWO - LOG(B) / TWO / T
                  CUT(I) = H
                  ICC(I) = 7
                  PQ(1, I) = ALNORM(H, .TRUE.)
                  H = H - T
                  PQ(2, I) = ALNORM(H, .FALSE.)
                  IF (T .LT. ZERO) THEN
                     ICC(I) = 8
                     PQ(1, I) = ONE - PQ(1, I)
                     PQ(2, I) = ONE - PQ(2, I)
                  ENDIF
            ENDIF
   10    CONTINUE
         RETURN
         END

         SUBROUTINE SWAP (X, Y)
C
```

```
C        ALGORITHM AS 276.4 APPL.STATIST. (1992), VOL.41, NO.2
C
C        Interchanges the values of two real variables : X and Y
C
      REAL X, Y
C
      REAL STORE
C
      STORE = X
      X = Y
      Y = STORE
      RETURN
      END

      REAL FUNCTION CHISQ1(G, VU)
C
C        ALGORITHM AS 276.5 APPL.STATIST. (1992), VOL.41, NO.2
C
C        Calculates the probability that a noncentral chi-square
C        variable is less than G (with DF=1 and noncentrality parameter
C        VU**2)
C
      REAL G, VU
C
      REAL ALNORM
      EXTERNAL ALNORM
      INTRINSIC SQRT
C
      REAL X, Y, ZERO
      PARAMETER (ZERO = 0.0E0)
C
      IF (G .GT. ZERO) THEN
         X = SQRT(G) - VU
         Y = -SQRT(G) - VU
         CHISQ1 = ALNORM(X, .FALSE.) - ALNORM(Y, .FALSE.)
         RETURN
      ELSE

         CHISQ1 = ZERO
         RETURN
      ENDIF
      END

C------------------------------------------------------------------------
```

```
      SUBROUTINE MEDIAN(X, Y, N, IWS, XMED, YMED, IFAULT)
C
C        ALGORITHM AS 277.1 APPL.STATIST. (1992) VOL.41, NO.3
C
C        On exit (XMED, YMED) will contain the bivariate median
C        of the points (X(i),Y(i)), i = 1, ... ,N.
C
C     *** WARNING ***
C     The routine as published in the journal contains at least three
C     errors.   This version has been constructed by Alan Miller.   It
C     appears to work correctly but no guarantee is given.
C     N.B. The name of this routine is MEDIAN, not MED as specified in
C     the Structure description given in the journal.
C
C     Auxiliary routines required: M01DAF and M01ZAF from the NAG library.
C
      INTEGER N, IWS(N), IFAULT
      REAL X(N), Y(N), XMED, YMED
C
      EXTERNAL M01DAF, M01ZAF
C
      INTEGER I, J, K, KQ, L, LL, NDIF, NT, NZ, NZERO
      REAL D0, DP, EPS, SMALL, SMALLD, T, TL, TU, W, WW, XKL,
     *      XMAX, XMIN, YKL, YMAX, YMIN
C
      DATA EPS / 1.0E-6 /
C
      IFAULT=0
      CALL M01DAF(X,1,N,'A',IWS,IFAULT)
      CALL M01ZAF(IWS,1,N,IFAULT)
      XMIN=X(IWS(1))
      XMAX=X(IWS(N))
C
      CALL M01DAF(Y,1,N,'A',IWS,IFAULT)
      CALL M01ZAF(IWS,1,N,IFAULT)
      YMIN=Y(IWS(1))
      YMAX=Y(IWS(N))
C
      IFAULT=0
      SMALL=EPS*(XMAX-XMIN)*(YMAX-YMIN)
      SMALLD=SMALL*N
C
      DO 40 NDIF=1,N
         WW=0.0
         DO 30 K=1,N-1
            DO 20 L=K+1,N
               NT=0
               XKL=X(K)-X(L)
               YKL=Y(K)-Y(L)
                 IF(ABS(XKL)+ABS(YKL).LT.SMALL) GO TO 15
               DO 10 I=1,N
                    W=(Y(I)-Y(L))*XKL-(X(I)-X(L))*YKL
                    WW=WW+ABS(W)
                    IF(W.GT.0.0)NT=NT+1
  10         CONTINUE
  15         CONTINUE
               IF((ABS(2*NT-N+2).LE.NDIF).AND.WW.GT.SMALLD) GO TO 50
  20        CONTINUE
  30     CONTINUE
         IF(WW.LE.SMALLD) THEN
C
```

```
C        The data set is completely collinear.
C
            IF (MOD(N,2).EQ.0) THEN
                 XMED=(X(IWS(N/2))+X(IWS(N/2+1)))/2
                 YMED=(Y(IWS(N/2))+Y(IWS(N/2+1)))/2
            ELSE
               XMED=X(IWS((N+1)/2))
               YMED=Y(IWS((N+l)/2))
            END IF
            RETURN
         END IF
C
   40 CONTINUE
C
   50 CALL TUTL(X,Y,N,XMIN,YMNI,XMAX,YMAX,K,L,SMALL,TU,TL)
C
C        Start the search along the line through the points (X(k),Y(k))
C        and (X(l),Y(l)), dividing the N-2 other points evenly.
C
   60 KQ=1
C
C        Search for the minimum of the Oja objective function on the
C        line (X(k),Y(k)) to (X(l),Y(l)).
C
      DO 80 I=1,N-1
           DO 70 J=I+1,N
           WW=(X(K)-X(L))*(Y(J)-Y(I))-(Y(K)-Y(L))*(X(J)-X(I))
           IF(ABS(WW).LT.SMALL) GO TO 70
                T=((X(L)-X(J))*(Y(I)-Y(J))-(Y(L)-Y(J))*(X(I)-X(J)))/WW
                IF(T.LT.TU.AND.T.GT.TL)THEN
                   CALL DER(X,Y,N,K,L,T,IWS,SMALL,NZERO,KQ,DP,D0)
c                  A=DP+D0                      ! Redundant instruction
                   IF(D0.GE.ABS(DP)-SMALLD) GO TO 90
                   IF(DP+D0.LT.0.0)TL=T
                   IF(DP+D0.GT.0.0)TU=T
                END IF
   70    CONTINUE
   80 CONTINUE
C
C        The local minimum on the line (X(k),Y(k)) - (X(l),Y(l))
C        was not found.
C
      IFAULT=IFAULT+2
      IF(IFAULT.GT.6) RETURN
      L=N-1
      K=N
   90 KQ=0
      I=K
      J=L
C
      LL=1
      NZ=NZERO
  100 K=INT(IWS(LL)/REAL(N))
      L=MOD(IWS(LL),N)+1
      WW=(X(K)-X(L))*(Y(J)-Y(I))-(Y(K)-Y(L))*(X(J)-X(I))
      IF(ABS(WW).GE.SMALL) THEN
         T=((X(L)-X(J))*(Y(I)-Y(J))-(Y(L)-Y(J))*(X(I)-X(J)))/WW
         CALL DER(X,Y,N,K,L,T,IWS,SMALL,NZERO,KQ,DP,D0)
         IF (D0.LT.ABS(DP)-SMALL) THEN
             CALL TUTL(X,Y,N,XMIN,YWIN,XMAX,YMAX,K,L,SMALL,TU,TL)
             IF(DP+D0.LT.0.0)THEN
```

```
                TL=T
              ELSE
                TU=T
              END IF
              GO TO 60
           END IF
           LL=LL+1
        END IF
C
        IF(LL.EQ.NZ+1)THEN
           XMED=X(L)+T*(X(K)-X(L))
           YMED=Y(L)+T*(Y(K)-Y(L))
           RETURN
        END IF
C
        GO TO 100
        END
C
        SUBROUTINE DER(X,Y,N,K,L,T,IWS,SMALL,NZERO,KQ,DP,D0)
C
C          ALGORITHM AS 277.2 APPL.STATIST. (1992) VOL.41, NO.3
C
C          DP-D0 and DP+D0 are the directional derivatives of the Oja
C           Objective function just before and after the point
C           (X(l)+t*(X(k)-X(l),Y(l)+t*(Y(k)-Y(l))
C
          INTEGER N, K, L, IWS(N), NZERO, KQ
          REAL X(N), Y(N), T, SMALL, DP, D0
C
        REAL DIF, SGN, SMALLD, TT, WW, XKL, YKL
        INTEGER II, JJ
C
        DATA SMALLD / 1E-6 /
C
        DP=0.0
        D0=0.0
        NZERO=0
        XKL=X(K)-X(L)
        YKL=Y(K)-Y(L)
C
        DO 20 II=1,N-1
            DO 10 JJ=II+1,N
            WW=(Y(JJ)-Y(II))*XKL-(X(JJ)-X(II))*YKL
            IF(ABS(WW).LE.SMALL) GO TO 10
            TT=((X(L)-X(JJ))*(Y(II)-Y(JJ))-(Y(L)-Y(JJ))*(X(II)-X(JJ)))/WW
            DIF=T-TT
            IF(ABS(DIF).LE.SMALLD)THEN
               NZERO=NZERO+1
               NZERO=MIN(NZERO,N)
               IF(KQ.NE.0)IWS(NZERO)=N*II+JJ-1
                  D0=D0+ABS(WW)
               ELSE
                  DP=DP+ABS(WW)*SGN(DIF)
               END IF
   10       CONTINUE
   20   CONTINUE
           RETURN
           END
C
        FUNCTION SGN(X)
C
```

```
C          ALGORITHM AS 277.3 APPL.STATIST. (1992) VOL.41, NO.3
C
      REAL SGN, X
      SGN=1.0
      IF (X.LT.0.0)SGN=-1.0
      RETURN
      END
C
      SUBROUTINE TUTL(X,Y,N,XMIN,YMIN,XMAX,YMAX,K,L,SMALL,TU,TL)
C
C        ALGORITHM AS 277.4 APPL.STATIST. (1992) VOL.41, NO.3
C
C        This subroutine calculates the upper (TU) and lower limit (TL)
C        for parameter T on the line (X(l)+t*(X(k)-X(l)), Y(l)+t*(Y(k)-Y(l))
C        inside the rectangle with vertices (XMIN,YMIN), (XMIN,YMAX),
C        (XMAX,YMIN) and (XMAX,YMAX).
C
        INTEGER N, K, L
        REAL X(N), Y(N), XMIN, YMIN, XMAX, YMAX, SMALL, TU, TL
        REAL T1, T2, T3, T4, VBIG
      DATA VBIG / 1E38 /
      T1=-VBIG
      T2=-T1
      IF(ABS(X(K)-X(L)).GT.SMALL)THEN
         T1=-(X(L)-XMIN)/(X(K)-X(L))
         T2= (XMAX-X(L))/(X(K)-X(L))
      END IF
      T3=-VBIG
      T4=-T3
      IF(ABS(Y(K)-Y(L)).GT.SMALL)THEN
         T3=-(Y(L)-YMIN)/(Y(K)-Y(L))
         T4= (YMAX-Y(L))/(Y(K)-Y(L))
      END IF
      TU=MIN(MAX(T1,T2),MAX(T3,T4))+SMALL
      TL=MAX(MIN(T1,T2),MIN(T3,T4))-SMALL
      RETURN
      END
```

```
      REAL FUNCTION PSI2(X, P, Q, A2, DELTA, MAXITR, IFAULT)
C
C        ALGORITHM AS 278.1 APPL.STATIST. (1992) VOL.41, NO.3
C
C        Calculates the probability that a random variable distributed
C        according to the psi-square distribution with P and Q degrees
C        of freedom and A2 eccentricity parameter, is less than or
C        equal to X
C
      REAL X, P, Q, A2, DELTA
      INTEGER MAXITR, IFAULT
C
      REAL ALNGAM, BETAIX
      EXTERNAL ALNGAM, BETAIX
C
      REAL AQAL, BL, CL, DAX, DJ, DJ1, ERP, ERR, EXPL, G,
     *     GCJ, GCL, HALF, ONE, PQ2, PREC, P2, QQAL, Q2, R,
     *     RL, SUM, SUMC, SUMCP, TWO, V, W, XLOW, XP, Y, YYL,
     *     Z, ZERO, ZM, ZN
      INTEGER IOK, J, JJ, JJJ, JOK, KOK
      LOGICAL LALF
C
      PARAMETER (ZERO=0.0E0, HALF=0.5E0, ONE=1.0E0, TWO=2.0E0)
C
C        machine-dependent constants
C
      PARAMETER (PREC=1.0E-6, XLOW=1.2E-38, EXPL=-87.315E0)
      PARAMETER (XP=XLOW/PREC)
C
      IFAULT = 0
C
C        test for valid input arguments
C
      IF (X .LT. ZERO .OR. A2 .LT. ZERO
     *   .OR. P .LE. ZERO .OR. Q .LE. ZERO
     *   .OR. DELTA .GE. ONE .OR. DELTA .LE. PREC) THEN
         IFAULT = 1
         PSI2 = -ONE
         RETURN
      ENDIF
C
C        define usefull parameters
C
      P2  = P * HALF
      Q2  = Q * HALF
      PQ2 = P2 + Q2
C
C        case A2 = 0
C
      IF (A2 .LT. PREC) THEN
         PSI2 = BETAIX( P*X/(P*X+Q), P2, Q2, IOK )
         IF (IOK .NE. 0) THEN
            IFAULT = 4
            PSI2 = -ONE
         ENDIF
         RETURN
      ENDIF
C
C        case P = 1
C
      IF (ABS(P-ONE) .LT. PREC) THEN
```

```
            DAX  = TWO * SQRT( A2*X )
            PSI2 = HALF * ( SIGN( ONE, X-A2 ) *
     *           BETAIX( (A2+X-DAX)/(A2+X-DAX+Q),HALF,Q2,IOK ) +
     *           BETAIX( (A2+X+DAX)/(A2+X+DAX+Q),HALF,Q2,JOK ) )
            IF (IOK .NE. 0 .OR. JOK .NE. 0) THEN
               IFAULT = 4
               PSI2 = -ONE
            ENDIF
            RETURN
          ENDIF
C
          Y = P*X / (A2+Q+P*X)
C
C         case P = Q and Y = 0.5
C
          IF (ABS(Y-HALF) .LT. PREC .AND. ABS(P-Q) .LT. PREC) THEN
            PSI2 = HALF
            RETURN
          ENDIF
C
C         calculate 1-F(X) or F(X) (LALF is the indicator)
C
          IF (Y .GT. HALF .OR. (Y .EQ. HALF .AND. P .GT. Q) ) THEN
            LALF = .TRUE.
            Z    = ONE - Y
            ZM   = Q2
            ZN   = P2
           ELSE
            LALF = .FALSE.
            Z    = Y
            ZM   = P2
            ZN   = Q2
          ENDIF
C
C         Y near 0 or 1 ?
C
          IF (Z .LT. PREC) THEN
            IF( LALF )THEN
               PSI2 = ONE
             ELSE
               PSI2 = ZERO
            ENDIF
            RETURN
          ENDIF
C
C         General case (iterations)
C         G's are decreasing for J >= JJ = -V/W + 1
C         CL's  are decreasing for J >= JJJ = A2/2 + 1
C         GCJ, SUMC, SUMCP are only used for stopping rule
C         logs are used to avoid underflows
C
          ERR  = DELTA * HALF
          ERP  = ERR / PREC
          YYL  = LOG( Y*(ONE-Y) )
          QQAL = Q2 * LOG( Q/(Q+A2) )
          AQAL = LOG( A2/(Q+A2) )
          V    = PQ2 * Z - ZN
          W    = Z + Z - ONE
C
C         initialize
C
```

```
      BL = ZERO
      CL = QQAL
      G  = BETAIX( Z, ZM, ZN, IOK )
      IF (IOK .NE. 0) THEN
         IFAULT = 4
         PSI2 = -ONE
         RETURN
      ENDIF
      RL = P2 * LOG(Y)  +  Q2 * LOG(ONE-Y)  +  ALNGAM(PQ2,IOK)
     *     - ALNGAM(P2,JOK) - ALNGAM(Q2,KOK) - LOG(P2) - LOG(Q2)
      IF (IOK+JOK+KOK .NE. 0) THEN
         IFAULT = 5
         PSI2 = -ONE
         RETURN
      ENDIF
      IF (RL .GE. EXPL) THEN
         R = EXP( RL )
       ELSE
         R = ZERO
      ENDIF
      SUM   = ZERO
      GCJ   = ZERO
      SUMC  = ZERO
      SUMCP = SUMC
      DJ    = ZERO
C
C        define minimum numbers of iterations (JJ and JJJ)
C
      IF (ZM .GT. ZN .AND. W .NE. ZERO) THEN
         JJ = INT( -V/W ) + 1
       ELSE
         JJ = 0
      ENDIF
      JJJ = INT( A2*HALF ) + 1
C
C        iteration loop
C
      DO 10 J = 0, MAXITR
C
         IF (G .GT. ZERO) THEN
            GCL = LOG(G) + CL
            IF (GCL .GE. EXPL) THEN
               GCL = EXP( GCL )
               SUM = SUM + GCL
               GCJ = GCJ + GCL*DJ
C                 check loss of accuracy
               IF (GCJ .GE. ERP) THEN
                  IFAULT = 3
                  GOTO 20
               ENDIF
            ENDIF
         ENDIF
C
         IF (CL .GE. EXPL) SUMC = SUMC + EXP(CL)
C
C           check accuracy (stopping rule)
C
         IF (J .GE. JJ) THEN
C              XP is used to prevent possible underflow
            IF (GCJ .GE. XP) THEN
               IF (PREC*GCJ+G*(ONE-SUMC) .LT. ERR) GOTO 20
```

```
                ELSE
                 IF (         G*(ONE-SUMC) .LT. ERR) GOTO 20
              ENDIF
              IF (J .GE. JJJ .AND. SUMC .EQ. SUMCP .AND.
     *         ABS(ONE-SUMC) .GE. PREC) THEN
                 IFAULT = 3
                 GOTO 20
              ENDIF
           ENDIF
C
C             prepare next iteration
C
           SUMCP = SUMC
           DJ1   = DJ + ONE
           BL    = BL + LOG( (Q2+DJ)/DJ1 )
           CL    = BL + QQAL + DJ1*AQAL
           G     = G  + (V+W*DJ)*R
           RL    = RL + YYL +
     *             LOG( ((DJ+DJ+PQ2)/(DJ1+P2)) * ((DJ+DJ1+PQ2)/(DJ1+Q2)) )
           IF (RL .GE. EXPL) THEN
              R = EXP( RL )
            ELSE
              R = ZERO
           ENDIF
           DJ = DJ1

      10 CONTINUE
C
C          we get here if maximum number of iterations is reached
C
         IFAULT = 2
C
C          the end
C
      20 CONTINUE
         IF (LALF) THEN
            PSI2 = ONE - SUM
          ELSE
            PSI2 = SUM
         ENDIF
C
         END
         REAL FUNCTION BETAIX(X, P, Q, IFAULT)
C
C          ALGORITHM AS 278.2 APPL.STATIST. (1992) VOL.41, NO.3
C
         REAL X, P, Q
         INTEGER IFAULT
C
         REAL ALNGAM, BETAIN
         EXTERNAL ALNGAM, BETAIN
C
         REAL BETA, EXPL, ZERO
         INTEGER IOK, JOK, KOK
C
         PARAMETER (ZERO=0.0E0)
C
C          machine-dependent constant
C
         PARAMETER (EXPL=-87.315E0)
C
```

```
      BETAIX = ZERO
C
C        calculate beta parameter for betain
C
      BETA = ALNGAM(P,JOK) + ALNGAM(Q,KOK) - ALNGAM(P+Q,IOK)
      IF (IOK+JOK+KOK .NE. 0) THEN
         IFAULT = 5
         RETURN
      ENDIF
      IF (BETA .GE. EXPL) THEN
         BETA =  EXP(BETA)
        ELSE
         IFAULT = 4
         RETURN
      ENDIF
      BETAIX = BETAIN(X, P, Q, BETA, IFAULT)
      END
```

```
      SUBROUTINE DWPVAL (X, IAX, N, M, NCONST, LAG, IALT, E, C, P,
     *                   WS1, WS2, WS3, IFAULT)
C
C     ALGORITHM AS279 APPL. STATIST. (1993) VOL. 42, NO.1
C
C     P-VALUE FOR GENERALIZED/ALTERNATIVE DURBIN-WATSON STATISTIC WITH
C     ARBITRARY LAG BY CHOLESKY TRANSFORMATION METHOD
C
C       ARGUMENT DECLARATIONS
C
      INTEGER N, M, IAX, NCONST, LAG, IFAULT, IALT
      REAL X(IAX,*), E, C, P, WS1(*), WS2(*), WS3(*)
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER J, K, MTOT, MNET, ITAIL
      REAL POINT5, POINT1, ZERO, ONE, E1, E2, E3, F1, F2, U, UJ,
     *     PI, RBD, UMAX, DELTA, ZFLOAT, ZLOG, ZIMAG, ZABS
      COMPLEX CF
C
C       CONSTANTS
C
      DATA ZERO, ONE, POINT5, POINT1 / 0.E0, 1.E0, 5.E-1, 1.E-1 /
      DATA PI /3.14159265358979E0 /
C
C       STANDARD FUNCTIONS
C
      ZFLOAT(J) = FLOAT(J)
      ZLOG(U) = ALOG(U)
      ZIMAG(CF) = AIMAG(CF)
      ZABS(CF) = CABS(CF)
C
C       INITIALIZATIONS
C
      IFAULT = 0
      MTOT = M + NCONST
      IF (N .LE. MTOT) THEN
        IFAULT = -1
        RETURN
      ELSEIF (N .LT. 2 * LAG) THEN
        IFAULT = -3
        RETURN
      ENDIF
      E1 = E * POINT1
      E3 = (E - E1) * POINT1
      E2 = E - E1 - E3
      E3 = E3 * PI
C
C       INTERVAL SIZE AND TRUNCATION LIMIT FOR TRAPEZOIDAL RULE
C
      MNET = MTOT - NCONST
      CALL GKFTRP (N, MNET, NCONST, LAG, IALT, C, E1, E2, DELTA, K,
     *             WS1, WS1(N+1), ITAIL, IFAULT)
      IF (IFAULT .NE. 0) RETURN
      IF (ITAIL .GT. 0) THEN
        P = ONE
        RETURN
      ELSEIF (ITAIL .LT. 0) THEN
        P = ZERO
        RETURN
      ENDIF
```

```
C
C       SCALE FACTOR FOR CHARACTERISTIC FUNCTION
C
      CALL DWSCL (X, IAX, N, M, NCONST, F1, F2, WS2, IFAULT)
      IF (IFAULT .NE. 0) RETURN
C
C       NUMERICAL INTEGRATION
C
      UMAX = ZFLOAT(K) + POINT5
      RBD = ONE
      P = ZERO
      J = 0
   50 IF (RBD .GT. E3 .AND. J .LE. K) THEN
        UJ = ZFLOAT(J) + POINT5
        U = UJ * DELTA
        CALL DWCHF(X, IAX, N, M, NCONST, LAG, IALT, U, C, F1, F2,
     *             CF, WS2, WS3, IFAULT)
        IF (IFAULT .NE. 0) RETURN
        P = P + ZIMAG(CF) / UJ
        RBD = ZABS(CF) * ZLOG(UMAX / UJ)
        J = J + 1
        GOTO 50
      ENDIF
      P = POINT5 - P / PI
      RETURN
      END
C
C
      SUBROUTINE DWCHF (X, IAX, N, M, NCONST, LAG, IALT, T, C, F1,
     *                  F2, CF, S, XT, IFAULT)
C
C     CHARACTERISTIC FUNCTION FOR GENERALIZED/ALTERNATIVE DURBIN-
C     WATSON STATISTIC AND EXACT LBI VERSIONS FOR ARBITRARY LAG
C
C       ARGUMENT DECLARATIONS:
C
      INTEGER IAX, IALT, N, M, NCONST, IFAULT, LAG
      REAL X(IAX,*), T, C, F1, F2
      COMPLEX CF, S(*), XT(LAG+1,*)
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER MTOT, I, J, K, MLAG, II, JJ, KK, NN, N1, IROW, JROW
      REAL ZERO, ONE, TWO, D2
      COMPLEX A1, A2, A, B, LB, ZCMPLX, ZSQRT, D1, LD, LLD
C
C       CONSTANTS
C
      DATA ZERO,ONE,TWO / 0.E0,1.E0,2.E0 /
C
C       STANDARD FUNCTIONS
C
      ZCMPLX(F1,F2) = CMPLX(F1,F2)
      ZSQRT(A) = CSQRT(A)
C
C       INITIALIZATIONS
C
      IFAULT = 0
      CF = F1
      MTOT = M + NCONST
      K = 0
```

```
      DO 20 I = 1,MTOT
      DO 10 J = 1,I
      K = K + 1
      S(K) = ZERO
 10   CONTINUE
 20   CONTINUE
      A1 = ZCMPLX(ONE,TWO * T * (C - ONE))
      A2 = A1 - ZCMPLX(ZERO, TWO * T)
      IF (IALT .NE. 0) A1 = A2
      B = ZCMPLX(ZERO, TWO * T)
      NN = (N - 1)/LAG + 1
      N1 = N - LAG +1
      MLAG = LAG + 1
      IROW = 0
      KK = 0
C
C       CHOLESKY TRANSFORMATION CYCLE
C
      DO 110 II = 1,NN
      DO 100 JJ = 1,LAG
      KK = KK + 1
      IF (KK .LE. N) THEN
        IROW = IROW + 1
        IF (IROW .GT. MLAG) IROW = 1
        IF (II .EQ. 1) THEN
          IF (JJ .EQ. 1) LD = ONE / ZSQRT(A1)
          DO 30 I = 1,M
          XT(IROW,I) = X(KK,I) * LD
 30       CONTINUE
          IF (NCONST .NE. 0) XT(IROW,MTOT) = LD
        ELSE
          IF (JJ .EQ. 1 .OR. KK .EQ. N1) THEN
            IF (JJ .NE. 1 .OR. KK .GT. N1) THEN
              LB = B * LLD
            ELSE
              LB = B * LD
            ENDIF
            LLD = LD
            A = A2
            IF (KK .GE. N1) A = A1
            LD = ONE / ZSQRT(A - LB * LB)
          ENDIF
          JROW = IROW - LAG
          IF (JROW .LE. 0) JROW = JROW + MLAG
          DO 40 I = 1,M
          XT(IROW,I) = LD * (X(KK,I) - LB * XT(JROW,I))
 40       CONTINUE
          IF (NCONST .NE. 0) XT(IROW,MTOT)=LD*(ONE-LB*XT(JROW,MTOT))
        ENDIF
        CF = CF * LD
C
C       UPDATE CROSS-PRODUCT MATRIX
C
        IF (MTOT .GT. 0) THEN
          K = 0
          DO 60 I = 1,MTOT
          DO 50 J = 1,I
          K = K + 1
          S(K) = S(K) + XT(IROW,I) * XT(IROW,J)
 50       CONTINUE
 60       CONTINUE
```

```
          ENDIF
        ENDIF
  100   CONTINUE
  110   CONTINUE
C
C       DETERMINANT OF CROSS-PRODUCT MATRIX
C
        IF (MTOT .GT. 0) THEN
          CALL CCLSKY(S,MTOT,D1,D2,IFAULT)
          CF = CF * D1 * TWO ** (D2 + F2)
        ENDIF
        RETURN
        END
C
C
        SUBROUTINE CCLSKY(S,M,D1,D2,IFAULT)
C
C       CHOLESKY FACTORIZATION IN COMPLEX ARITHMETIC
C
C       ARGUMENT DECLARATIONS:
C
        INTEGER M, IFAULT
        REAL D2
        COMPLEX S(*), D1
C
C       LOCAL VARIABLE DECLARATIONS
C
        INTEGER I, J, JJ, K, L, KK, KI, KJ
        REAL EPS, ZERO, ONE, FOUR, SIXTEN, SIXNTH, ZABS
        COMPLEX SUM, ZSQRT
C
C       CONSTANTS
C
        DATA ZERO,ONE,FOUR,SIXTEN,SIXNTH / 0.E0,1.E0,4.E0,1.6E1,.625E-1 /
        DATA EPS / 1.E-6 /
C
C       STANDARD FUNCTIONS
C
         ZSQRT(SUM) = CSQRT(SUM)
         ZABS(SUM) = CABS(SUM)
C
C       INITIALIZATIONS
C
        IFAULT = 0
        D1 = ONE
        D2 = ZERO
C
C       FACTORIZATION
C
        K = 0
        DO 60 I = 1,M
        KK = 0
        KI = K
        DO 50 J = 1,I
        K = K + 1
        SUM = S(K)
        IF (J .GT. 1) THEN
          JJ = J - 1
          KJ = KI
          DO 10 L = 1,JJ
          KK = KK + 1
```

```
          KJ = KJ + 1
          SUM = SUM - S(KK) * S(KJ)
   10     CONTINUE
        ENDIF
        KK = KK + 1
        IF (J .LT. I) THEN
          S(K) = SUM * S(KK)
        ELSE
          IF (ZABS(SUM) .LT. ZABS(S(K)) * EPS) THEN
            IFAULT = -2
            D1 = ZERO
            RETURN
          ENDIF
          S(K) = ONE / ZSQRT(SUM)
          D1 = D1 * S(K)
   20     IF (ZABS(D1) .LE. ONE) GOTO 30
          D1 = D1 * SIXNTH
          D2 = D2 + FOUR
          GOTO 20
   30     IF (ZABS(D1) .GT. SIXNTH) GOTO 40
          D1 = D1 * SIXTEN
          D2 = D2 - FOUR
          GOTO 30
   40     CONTINUE
        ENDIF
   50   CONTINUE
   60   CONTINUE
        RETURN
        END
C
C
        SUBROUTINE DWSCL (X, IAX, N, M, NCONST, F1, F2, S, IFAULT)
C
C     SCALE FACTOR FOR CHARACTERISTIC FUNCTION
C
C       ARGUMENT DECLARATIONS
C
        INTEGER IAX, N, M, NCONST, IFAULT
        REAL X(IAX,*), F1, F2, S(2,*)
C
C       LOCAL VARIABLE DECLARATIONS:
C
        INTEGER I, J, K, L, MTOT
        REAL EPS, TEMP, ONE, ZERO, ZREAL
        COMPLEX DET
C
C       CONSTANTS
C
        DATA ONE,ZERO / 1.E0,0.E0 /
C
C       STANDARD FUNCTIONS
C
        ZREAL(DET) = REAL(DET)
C
C       INITIALIZATIONS
C
        IFAULT = 0
        K = 0
        MTOT = M + NCONST
        IF (MTOT .EQ. 0) THEN
          F1 = ONE
```

```
          F2 = ZERO
          RETURN
        ENDIF
        DO 20 I = 1,MTOT
        DO 10 J = 1,I
        K = K + 1
        S(1,K) = ZERO
        S(2,K) = ZERO
 10     CONTINUE
 20     CONTINUE
C
C       CROSS-PRODUCT MATRIX
C
        DO 50 L = 1,N
        K = 0
        DO 40 I = 1,MTOT
        TEMP = ONE
        IF (I .LE. M) TEMP = X(L,I)
        DO 30 J = 1,I
        K = K + 1
        IF (J .LE. M) THEN
          S(1,K) = S(1,K) + X(L,J) * TEMP
        ELSE
          S(1,K) = S(1,K) + TEMP
        ENDIF
  30    CONTINUE
  40    CONTINUE
  50    CONTINUE
C
C       SCALE FACTOR: DETERMINANT OF CHOLESKY FACTOR
C
        CALL CCLSKY(S,MTOT,DET,F2,IFAULT)
        IF (IFAULT .NE. 0) RETURN
        F1 = ONE / ZREAL(DET)
        F2 = - F2
        RETURN
        END
C
C
        SUBROUTINE GKFTRP (N, M, NCONST, LAG, IALT, C, E1, E2, DELTA, K,
       *                   WS1, WS2, ITAIL, IFAULT)
C
C       INTERVAL SIZE AND TRUNCATION LIMIT FOR TRAPEZOIDAL RULE
C       EVALUATION OF GIL-PALAEZ FORMULA.
C
C       ARGUMENT DECLARATIONS
C
        INTEGER N, M, NCONST, IALT, K, LAG, ITAIL, IFAULT
        REAL C, E1, E2, DELTA, WS1(*), WS2(*)
C
C       LOCAL VARIABLE DECLARATIONS
C
        INTEGER NNET, I, J, MAX, NG, K1
        REAL D2, ZERO, TWO, E, ZMIN1
C
C       CONSTANTS
C
        DATA MAX / 20 /
        DATA ZERO, TWO / 0.E0, 2.E0 /
C
C       STANDARD FUNCTIONS
```

```
C
      ZMIN1(E1,E2) = AMIN1(E1,E2)
C
C       INITIALIZATIONS
C
      IFAULT = 0
      IF (N .LE. M + NCONST) THEN
        IFAULT = -1
        RETURN
      ENDIF
      IF (IALT .EQ. 0) THEN
        NNET = N - M - NCONST
        K1 = NCONST
      ELSE
        NNET = N - M
        K1 = 0
      ENDIF
C
C       EIGENVALUES OF MATRIX DEFINING STATISTIC
C
      CALL EVALUE (WS1, N, LAG, IALT, C)
C
C       INITIAL CHECK FOR TAILS
C
      IF (WS1(K1 + 1) .GE. 0) THEN
        ITAIL = -1
        RETURN
      ELSEIF (WS1(N) .LE. 0) THEN
        ITAIL = 1
        RETURN
      ENDIF
C
C       INTERVAL SIZE
C
      E = E1 + E2
      IF (C .GT. TWO) THEN
        ITAIL = 1
        J = M + K1
      ELSE
        ITAIL = -1
        J = K1
      ENDIF
      DO 10 I = 1, NNET
      J = J + 1
      WS2(I) = WS1(J)
   10 CONTINUE
      J = ITAIL
      CALL INTBD (WS2, NNET, E, E1, DELTA, ITAIL)
      IF (ITAIL .NE. 0) RETURN
      ITAIL = -J
      IF (ITAIL .EQ. 1) THEN
        J = M + K1
      ELSE
        J = K1
      ENDIF
      DO 20 I = 1, NNET
      J = J + 1
      WS2(I) = WS1(J)
   20 CONTINUE
      CALL INTBD (WS2, NNET, E, E1, D2, ITAIL)
      IF (ITAIL .NE. 0) RETURN
```

```
      DELTA = ZMIN1 (DELTA, D2)
C
C       TRUNCATION POINT FOR TRAPEZOIDAL RULE
C
      CALL TBDWT (WS1, N, M, NCONST, IALT, WS2, NG)
      IF (NG .EQ. 0) THEN
        IFAULT = 2
        RETURN
      ENDIF
      CALL TBND1 (WS2, NG, E2, DELTA, K1, IFAULT)
      IF (IFAULT .NE. 0) K1 = 0
      IFAULT = 0
      CALL TBND2 (WS2, NG, E2, DELTA, K)
      IF (K1 .GT. 0) K = MIN0(K,K1)
      RETURN
      END
C
C
      SUBROUTINE TBDWT (W, N, M, NCONST, IALT, G, NG)
C
C     WEIGHTS (SQUARED) FOR BOUNDING FUNCTION TO WHICH DAVIES' (APPL.
C     STATIST. 29, 1980, 323-333) TRUNCATION RULES ARE TO BE APPLIED.
C
C       ARGUMENT DECLARATIONS:
C
      INTEGER N, M, NCONST, IALT, NG
      REAL W(N), G(N)
C
C       LOCAL VARIABLE DECLARATIONS:
C
      INTEGER I, MTOT, NNET, L1, L2
      REAL ZERO
C
C       CONSTANTS
C
      DATA ZERO / 0.E0 /
C
C       NUMBER OF REGRESSORS, DEGREES OF FREEDOM
C
      MTOT = M + NCONST
      NNET = N - MTOT
C
C       WEIGHTS
C
      IF (IALT .EQ. 0 .AND. NCONST .NE. 0) THEN
        L1 = 1
      ELSE
        L1 = 0
      ENDIF
      L2 = MTOT
      NG = 0
      DO 30 I = 1, NNET
      L1 = L1 + 1
      L2 = L2 + 1
      IF (W(L1) .GT. ZERO) THEN
        NG = NG + 1
        G(NG) = W(L1) ** 2
      ELSEIF (W(L2) .LT. ZERO) THEN
        NG = NG + 1
        G(NG) = W(L2) ** 2
      ENDIF
```

```
   30 CONTINUE
      IF (NG .EQ. N) RETURN
      NNET = NG + 1
      DO 40 I = NNET, N
      G(I) = ZERO
   40 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE EVALUE (W, N, LAG, IALT, C)
C
C     EIGENVALUES OF DEFINING QUADRATIC FORM
C
C       ARGUMENT DECLARATIONS:
C
      INTEGER N, LAG, IALT
      REAL W(N), C
C
C       LOCAL VARIABLE DECLARATIONS:
C
      INTEGER I, J, K, L, NVAL, M1, M2
      REAL PI, D1, D2, TWO, C2, TEMP, ZFLOAT, ZCOS
C
C       STANDARD FUNCTIONS
C
      ZFLOAT(I) = FLOAT(I)
      ZCOS(D1) = COS(D1)
C
C       CONSTANTS
C
      DATA PI / 3.14159265358979E0 /
      DATA TWO / 2.E0 /
      C2 = TWO - C
C
C     FIRST-ORDER CASE
C
      IF (LAG .GT. 1) GOTO 20
      L = IALT
      IF (IALT .EQ. 0) D1 = PI / ZFLOAT(N)
      IF (IALT .NE. 0) D1 = PI / ZFLOAT(N + 1)
      DO 10 I = 1, N
      W(I) = C2 - TWO * ZCOS(L * D1)
      L = L + 1
   10 CONTINUE
      RETURN
C
C     GENERAL-ORDER CASE
C
   20 NVAL = (N - 1) / LAG + 1
      IF (IALT .EQ. 0) D1 = PI / ZFLOAT(NVAL)
      IF (IALT .NE. 0) D1 = PI / ZFLOAT(NVAL + 1)
      M1 = N - (NVAL - 1) * LAG
      M2 = M1 + 1
      IF (M1 .EQ. LAG) GOTO 30
      IF (IALT .EQ. 0) D2 = PI / ZFLOAT(NVAL - 1)
      IF (IALT .NE. 0) D2 = PI / ZFLOAT(NVAL)
   30 K = 0
      L = IALT - 1
      DO 60 I = 1, NVAL
      L = L + 1
```

```
      TEMP = C2 - TWO * ZCOS(L * D1)
      DO 40 J = 1, M1
      K = K + 1
      W(K) = TEMP
   40 CONTINUE
      IF (M1 .EQ. LAG .OR. K .EQ. N)GOTO 60
      TEMP = C2 - TWO * ZCOS(L * D2)
      DO 50 J = M2, LAG
      K = K + 1
      W(K) = TEMP
   50 CONTINUE
   60 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE LMGFEV (U, G, N, PSI, DPSI, D2PSI)
C
C     LOG MOMENT GENERATING FUNCTION FROM EIGENVALUES ADJUSTED FOR
C     ARGUMENT OF DISTRIBUTION FUNCTION, AND DERIVATIVES
C
C       ARGUMENT DECLARATIONS
C
      INTEGER N
      REAL U, G(N), PSI, DPSI, D2PSI
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER I
      REAL U2, F, ZERO, ONE, TWO, ZLOG
C
C       CONSTANTS
C
      DATA ZERO, ONE, TWO / 0.E0, 1.E0, 2.E0/
C
C       STANDARD FUNCTIONS
C
      ZLOG(F) = ALOG(F)
C
C       FUNCTION AND DERIVATIVES
C
      U2 = - TWO * U
      PSI = ZERO
      DPSI = ZERO
      D2PSI = ZERO
      DO 10 I = 1, N
      F = ONE + G(I) * U2
      PSI = PSI + ZLOG(F)
      F = G(I) / F
      DPSI = DPSI + F
      D2PSI = D2PSI + F ** 2
   10 CONTINUE
      PSI = - PSI / TWO
      D2PSI = TWO * D2PSI
      RETURN
      END
C
C
      SUBROUTINE INTBD (G, N, E, ET, DELTA, ITAIL)
C
C     INTERVAL FOR TRAPEZOIDAL RULE INTEGRATION TO LIMIT ERROR TO E
```

```
C     USING MOMENT GENERATING FUNCTION BOUND FOR WEIGHTED SUM OF CHI-
C     SQUARED(1) RANDOM VARIABLES
C
C       ARGUMENT DECLARATIONS
C
      INTEGER N, ITAIL
      REAL DELTA, G(N), E, ET
C
C       LOCAL VARIABLE DECLARATIONS:
C
      INTEGER I
      REAL ZERO, TWO, POINT1, POINT5, TWOPI, ULIM, PSI, DPSI, D2PSI,
     *     HB, U, U1, U2, ELOG, EPS, HM, HSD, SGN, ZABS, ZEXP, ZFLOAT,
     *     ZLOG, ZMAX1, ZMIN1, ZSQRT
C
C       SET CONSTANTS (EPS = LOG(1.1))
C
      DATA ZERO, TWO, POINT1, POINT5 / 0.E0, 2.E0, .1E0, .5E0 /
      DATA TWOPI / 6.28318530717959E0 /
      DATA EPS / .09531E0 /
C
C       STANDARD FUNCTIONS
C
      ZABS(U) = ABS(U)
      ZEXP(U) = EXP(U)
      ZFLOAT(I) = FLOAT(I)
      ZLOG(U) = ALOG(U)
      ZMAX1(U,U1) = AMAX1(U,U1)
      ZMIN1(U,U1) = AMIN1(U,U1)
      ZSQRT(U) = SQRT(U)
C
C       TAIL CHECK
C
      SGN = ZFLOAT(ITAIL)
      IF (ITAIL .EQ. 1) THEN
        IF (G(N) .LE. ZERO) RETURN
        ULIM = POINT5 / G(N)
      ELSE
        IF (G(1) .GE. ZERO) RETURN
        ULIM = POINT5 / G(1)
      ENDIF
      HM = ZERO
      HSD = ZERO
      DO 10 I = 1, N
      HM = HM + G(I)
      HSD = HSD + G(I) ** 2
   10 CONTINUE
C
C       NEWTON SEARCH FOR ZERO LOG(MGF) DERIVATIVE
C
      HSD = ZSQRT(HSD)
      U2 = ZERO
      IF ( SGN * HM .LT. ZERO .AND. - SGN * HM / HSD .GT. TWO) THEN
        DPSI = HM
   20   IF (DPSI * SGN .LT. ZERO) THEN
          U1 = U2
          U2 = POINT5 * (U2 + ULIM)
          CALL LMGFEV (U2, G, N, PSI, DPSI, D2PSI)
          GOTO 20
        ENDIF
   25   IF (DPSI ** 2 .GE. POINT1 * D2PSI) THEN
```

```
            U2 = U2 - DPSI / D2PSI
            CALL LMGFEV (U2, G, N, PSI, DPSI, D2PSI)
          ENDIF
          IF (ZEXP(PSI) .LE. E) RETURN
        ENDIF
C
C       BOUND EQUALITY
C
        ELOG = - ZLOG(ET)
        HB = POINT5
   30   IF (HB .GT. ZERO) THEN
          U1 = U2
          U2 = POINT5 * (U2 + ULIM)
          CALL LMGFEV (U2, G, N, PSI, DPSI, D2PSI)
          HB = PSI - U2 * DPSI + ELOG
          GOTO 30
        ENDIF
C
C       NEWTON/BINARY SEARCH SOLUTION
C
        U = U2
   40   IF (ZABS(HB) .GE. EPS) THEN
          U = U - HB / (U * D2PSI)
          IF (U .LE. ZMIN1(U1,U2) .OR. U .GT. ZMAX1(U1,U2))
     *      U = POINT5 * (U1 + U2)
          CALL LMGFEV (U, G, N, PSI, DPSI, D2PSI)
          HB = PSI - U * DPSI + ELOG
          IF (HB .GT. ZERO) THEN
            U1 = U
          ELSE
            U2 = U
          ENDIF
          GOTO 40
        ENDIF
        IF (ITAIL .EQ. 1 .AND. DPSI .GT. ZERO) THEN
          DELTA = TWOPI / DPSI
          ITAIL = 0
        ELSEIF (ITAIL .EQ. -1 .AND. DPSI .LT. ZERO) THEN
          DELTA = - TWOPI / DPSI
          ITAIL = 0
        ENDIF
        RETURN
        END
C
C
        SUBROUTINE TBND1 (G, N, E, DELTA, K, IFAULT)
C
C       UPPER BOUND ON NUMBER OF TERMS REQUIRED IN TRAPEZOIDAL RULE
C       INTEGRATION USING DAVIES' BOUND EQN (6), APPL. STATIST. 29 (1980)
C       P. 324.  WEIGHTS G(J) ARE SQUARES OF DAVIES' COEFFICIENTS.
C
C       ARGUMENT DECLARATIONS
C
        INTEGER N, K, IFAULT
        REAL G(N), E, DELTA
C
C       LOCAL VARIABLE DECLARATIONS
C
        INTEGER I, M
        REAL PIBY2, ZERO, ONE, TWO, FOUR, S, F, DF, C, U, ULAST,
     *     FLAST, DFLAST, EPS, POINT5, ZABS, ZFLOAT, ZLOG, ZSQRT
```

```
C
C       CONSTANTS (EPS = 4 * LOG(1.1))
C
      DATA PIBY2 / 1.57079632679490E0 /
      DATA ZERO, ONE, TWO, FOUR, POINT5
     *    / 0.E0, 1.E0, 2.E0, 4.E0, 5.E-1 /
      DATA EPS / .38124E0 /
C
C       STANDARD FUNCTIONS
C
      ZABS(C) = ABS(C)
      ZFLOAT(I) = FLOAT(I)
      ZLOG(C) = ALOG(C)
      ZSQRT(C) = SQRT(C)
C
C       INITIALIZATIONS
C
      IFAULT = 0
      M = 0
      C = ZERO
      DO 10 I = 1, N
      IF (G(I) .GT. ONE) THEN
        M = M + 1
        C = C + ZLOG(G(I))
      ENDIF
   10 CONTINUE
      IF (M .EQ. 0) THEN
        IFAULT = 1
        RETURN
      ENDIF
      S = ZFLOAT(M)
      C = C + FOUR * ZLOG(S * PIBY2 * E)
      I = 0
      U = FOUR * FOUR * DELTA ** 2
C
C       INITIAL ESTIMATE OF BOUND FUNCTION SOLUTION
C
   20 CALL FBND1 (G, N, C, S, U, F, DF)
      IF (F .GT. ZERO .AND. I .EQ. -1) THEN
        U = ULAST
        F = FLAST
        DF = DFLAST
      ELSEIF (F .GT. ZERO) THEN
        I = 1
        U = U * POINT5
        GOTO 20
      ELSEIF (F .LE. ZERO .AND. I .NE. 1) THEN
        I = -1
        ULAST = U
        FLAST = F
        DFLAST = DF
        U = TWO * U
        GOTO 20
      ENDIF
C
C       NEWTON'S METHOD SOLUTION FOR ARGUMENT OF BOUND FUNCTION
C
   30 IF (ZABS(F) .GE. EPS) THEN
        U = U - F / DF
        CALL FBND1 (G, N, C, S, U, F, DF)
        GOTO 30
```

```
      ENDIF
      K = ZSQRT(U / FOUR) / DELTA + POINT5
      RETURN
      END
C
C
      SUBROUTINE FBND1 (G, N, C, S, U, F, DF)
C
C     EVALUATION OF DAVIES' BOUND EQN (6), APPL. STATIST. 29 (1980)
C     P. 324.  WEIGHTS G(J) ARE SQUARES OF DAVIES' COEFFICIENTS, AND F
C     IS - 4 * (LOG(BOUND) - LOG(MAX ERROR)).
C
C       ARGUMENT DECLARATIONS
C
      INTEGER N
      REAL G(N), C, S, U, F, DF
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER J
      REAL ONE, TEMP, ZLOG
C
C       CONSTANTS
C
      DATA ONE / 1.E0 /
C
C       STANDARD FUNCTIONS
C
      ZLOG(C) = ALOG(C)
C
      F = C + S * ZLOG(U)
      DF = S / U
      DO 10 J = 1, N
      IF (G(J) .LE. ONE) THEN
        TEMP = ONE + U * G(J)
        F = F + ZLOG(TEMP)
        DF = DF + G(J) / TEMP
      ENDIF
   10 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE TBND2 (G, N, E, DELTA, K)
C
C     UPPER BOUND ON NUMBER OF TERMS REQUIRED IN TRAPEZOIDAL RULE
C     INTEGRATION USING DAVIES' BOUND EQN (8), APPL. STATIST. 29 (1980)
C     P. 324.  WEIGHTS G(J) ARE SQUARES OF DAVIES' COEFFICIENTS.
C     LEADING CONSTANT IS 2/PI AS IN AKS (1990).
C
C       ARGUMENT DECLARATIONS:
C
      INTEGER N, K
      REAL G(N), E, DELTA
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER I, M
      REAL C1, ZERO, ONE, TWO, FOUR, F, DF, CONST, EPS,
     *     POINT5, U, ULAST, FLAST, DFLAST, ZABS, ZLOG, ZSQRT
C
```

```
C       CONSTANTS (C1 = - 4 * LOG(2/PI), EPS = 4 * LOG(1.1))
C
      DATA C1 / 1.80633E0 /
      DATA ZERO, ONE, TWO, FOUR, POINT5 /
     *     0.E0, 1.E0, 2.E0, 4.E0, 5.E-1 /
      DATA EPS / .38124E0 /
C
C       STANDARD FUNCTIONS
C
      ZABS(U) = ABS(U)
      ZLOG(U) = ALOG(U)
      ZSQRT(U) = SQRT(U)
C
C       INITIAL ESTIMATE OF BOUND FUNCTION SOLUTION
C
      CONST = C1 + FOUR * ZLOG(E)
      I = 0
      U = FOUR * FOUR * DELTA ** 2
   10 CALL FBND2 (G, N, CONST, U, F, DF)
      IF (F .GT. ZERO .AND. I .EQ. -1) THEN
        U = ULAST
        F = FLAST
        DF = DFLAST
      ELSEIF (F .GT. ZERO) THEN
        I = 1
        U = U * POINT5
        GOTO 10
      ELSEIF (F .LE. ZERO .AND. I .NE. 1) THEN
        I = -1
        ULAST = U
        FLAST = F
        DFLAST = DF
        U = TWO * U
        GOTO 10
      ENDIF
C
C       NEWTON'S METHOD SOLUTION FOR ARGUMENT OF BOUND FUNCTION
C
   20 IF (ZABS(F) .GE. EPS) THEN
        U = U - F / DF
        CALL FBND2 (G, N, CONST, U, F, DF)
        GOTO 20
      ENDIF
C
C       NUMBER OF INTERVALS
C
      K = ZSQRT(U / FOUR) / DELTA + POINT5
      RETURN
      END
C
C
      SUBROUTINE FBND2 (G, N, C, U, F, DF)
C
C     EVALUATION OF DAVIES' BOUND EQN (8), APPL. STATIST. 29 (1980)
C     P. 324, WITH CONSTANT 2/PI AS IN AKS (1990).  WEIGHTS G(J) ARE
C     SQUARES OF DAVIES' COEFFICIENTS, AND F IS -4 * (LOG(BOUND)
C     - LOG(MAX ERROR)).
C
C       ARGUMENT DECLARATIONS
C
      INTEGER N
```

```
      REAL G(N), C, U, F, DF
C
C       LOCAL VARIABLE DECLARATIONS
C
      INTEGER J
      REAL ZERO, ONE, TEMP, ZLOG
C
C       CONSTANTS
C
      DATA ZERO, ONE / 0.E0, 1.E0 /
C
C       STANDARD FUNCTIONS
C
      ZLOG(U) = ALOG(U)
C
C       FUNCTION AND DERIVATIVE
C
      F = C
      DF = ZERO
      DO 20 J = 1, N
      TEMP = ONE + G(J) * U
      F = F + ZLOG(TEMP)
      DF = DF + G(J) / TEMP
   20 CONTINUE
      RETURN
      END
```

```
C       .. ULCC Toolpack/1 2.1
      DOUBLE PRECISION FUNCTION BINOM(N, LP1, LP2, LMP1, LMP2, X, R)
C
C        FAST BIVARIATE BINOMIAL DENSITY
C
      DOUBLE PRECISION ALNGAM, LP1, LP2, LMP1, LMP2, ONE, TWO
      INTEGER N, X, R, IER
      DATA ONE / 1.0 / , TWO / 2.0 /
C
      BINOM = DEXP(TWO * ALNGAM(N + ONE, IER) -
     *        (ALNGAM(X + ONE, IER) + ALNGAM(N - X + ONE,
     *        IER) + ALNGAM(R - X + ONE, IER) + ALNGAM(N - (R - X) +
     *        ONE, IER)) + X * LP1 + (R - X) * LP2 + (N - X) * LMP1 +
     *        (N - (R - X)) * LMP2)
      RETURN
      END


      INTEGER FUNCTION XFMAX(N, R, THETA)
C
C        RETURNS X SUCH THAT F(X,R) IS A MAXIMUM
C
      INTEGER N, R
      DOUBLE PRECISION THETA, A, B, C, Q, ONE, HALF, FOUR
      DATA ONE / 1.0 / , TWO / 2.0 / , HALF / 0.5 / , FOUR / 4.0 /
C
      IF (THETA .NE. ONE) THEN
         A = - (ONE - THETA)
         B = - ((R + N + TWO) * THETA + N - R)
         C = (N + ONE) * (R + ONE) * THETA
         Q = -HALF * (B - DSQRT(B * B - (FOUR * A * C)))
         XFMAX = INT(C / Q)
      ELSE
         XFMAX = (R + 1) / 2
      END IF
      RETURN
      END


      INTEGER FUNCTION CV(N, R, ALPHA)
C
C        RETURNS LARGEST X SUCH THAT PROB(X<=X) < ALPHA
C
      INTEGER N, R, XMIN, X, XSTART, IER
      DOUBLE PRECISION ALNGAM, ALPHA, HYPSUM, LNUM, LDEN, PROB, ONE,
     *                 HALF, TWO
      DATA ONE / 1.0 / , HALF / 0.5 / , TWO / 2.0 /
C
      LDEN = ALNGAM(2 * N + ONE, IER) - TWO * ALNGAM(N + ONE, IER) -
     *       (ALNGAM(R + ONE, IER) + ALNGAM(2 * N - R + ONE, IER))
      XSTART = (R - 1) / 2
      X = XSTART + 1
      LNUM = - (ALNGAM(R - X + ONE, IER) +
     *       ALNGAM(N - (R - X) + ONE, IER) + ALNGAM(X + ONE, IER) +
     *       ALNGAM(N - X + ONE, IER))
      PROB = DEXP(LNUM - LDEN)
      HYPSUM = HALF
      IF (MOD(R, 2) .EQ. 0) HYPSUM = HYPSUM + HALF * PROB
      XMIN = 0
      IF (R .GT. N) XMIN = R - N
C
```

```
C        HYPERGEOMETRIC ADJACENCY
C
      DO 10 X = XSTART, XMIN, -1
         PROB = PROB * (X + ONE) * (N - R + X + ONE) /
     *          ((R - X) * ONE * (N - X))
         HYPSUM = HYPSUM + PROB
         IF (HYPSUM .GT. ONE - ALPHA) GO TO 20
   10 CONTINUE
      X = XMIN
C
C        IF NO CRITICAL VALUE, RETURN -1
C
   20 CV = -1
      IF (X - 1 .GE. XMIN) CV = X - 1
      RETURN
      END


      DOUBLE PRECISION FUNCTION FSUM(FROM, TO, INC, F, FLIM, N, THETA,
     *                               R, RCNT)
C
C        SUMS BINOMIAL PROBABILITIES ALONG FIXED R
C
      DOUBLE PRECISION FP, F, FLIM, THETA, ONE, ZERO
      INTEGER X, N, FROM, TO, R, INC, RCNT
      DATA ONE / 1.0 / , ZERO / 0.0 /
C
      FP = ZERO
      RCNT = 0
      X = FROM
   10 IF (F .LT. FLIM) GO TO 20
      RCNT = RCNT + 1
      FP = FP + F
C
C        BINOMIAL ADJACENCY
C
      IF (INC .GT. 0) F = F * THETA * (N - X) * (R - X) /
     *                    ((X + ONE) * (N - R + X + 1))
      IF (INC .LE. 0) F = F * (ONE / THETA) * X * (N - R + X) /
     *                    ((N - X + ONE) * (R - X + 1))
      X = X + INC
      IF (X .NE. TO + INC) GO TO 10
   20 FSUM = FP
      RETURN
      END


      DOUBLE PRECISION FUNCTION FXPOWER(N, P1, P2, ALPHA, EPS, ERROR,
     *                 COUNT, TOTAL, IFAULT)
C
C        POWER FOR FISHER'S EXACT TEST
C
C        RETURN THE POWER OF A LEVEL ALPHA TEST FOR A GIVEN
C        COMMON SAMPLE SIZE AND TRUE PROBABILITIES OF EVENTS P1 AND P2.
C
      DOUBLE PRECISION P1, P2, ALPHA, EPS, ERROR, LP1, LP2, LMP1, LMP2,
     *                 F, FLIM, POWSUM, THETA, FMAX, FSUM, BINOM, ONE,
     *                 ZERO, TWO
      INTEGER XM, N, TOTAL, COUNT, RMAX, XMAX, YMAX, XMIN, XCV, XFM, R,
     *        I, J, RCNT, CV, XFMAX
      DATA ONE / 1.0 / , ZERO / 0.0 / , TWO / 2.0 /
```

```
C
      IFAULT = 0
      ERROR = ZERO
      POWSUM = ZERO
      COUNT = 0
      TOTAL = 0
C
C        CHECK INPUT PARMS
C
      IF (N .LE. 0 .OR. P1 .LE. ZERO .OR. P1 .GE. ONE .OR.
     *    P2 .LE. ZERO .OR. P2 .GE. ONE .OR. ALPHA .LE. ZERO .OR.
     *    ALPHA .GE. ONE .OR. EPS .LE. ZERO .OR. EPS .GE. ONE) THEN
         IFAULT = 1
         RETURN
      END IF
C
      LP1 = DLOG(P1)
      LP2 = DLOG(P2)
      LMP1 = DLOG(ONE - P1)
      LMP2 = DLOG(ONE - P2)
      XMAX = (N + 1) * P1
      YMAX = (N + 1) * P2
      RMAX = XMAX + YMAX
      FMAX = BINOM(N, LP1, LP2, LMP1, LMP2, XMAX, RMAX)
      FLIM = FMAX * EPS
      THETA = P1 * (ONE - P2) / (P2 * (ONE - P1))
      XM = 0
      IF (RMAX .GT. N) XM = RMAX - N
      TOTAL = (N + 1 - RMAX + TWO * (CV(N, RMAX, ALPHA) - XM)) *
     *        (N + 2 - RMAX + TWO * (CV(N, RMAX, ALPHA) - XM)) / 2
      R = RMAX
      I = 0
      J = 1
C
C        MAIN LOOP ON R
C
   10 XMIN = 0
      IF (R .GT. N) XMIN = R - N
      XCV = CV(N, R, ALPHA)
      XFM = XFMAX(N, R, THETA)
      IF (XCV .LE. 0) GO TO 30
      IF (XCV .GT. XFM) GO TO 20
C
C        SUM FROM CRITICAL VALUE DOWN TO MINIMUM
C
      F = BINOM(N, LP1, LP2, LMP1, LMP2, XCV, R)
      POWSUM = POWSUM + FSUM(XCV, XMIN, -1, F, FLIM, N, THETA, R, RCNT)
      IF (RCNT .EQ. 0) GO TO 40
      COUNT = COUNT + RCNT
      GO TO 30
C
C        SUM FROM DENSITY MAX DOWN TO MINIMUM
C
   20 F = BINOM(N, LP1, LP2, LMP1, LMP2, XFM, R)
      POWSUM = POWSUM + FSUM(XFM, XMIN, -1, F, FLIM, N, THETA, R, RCNT)
      IF (RCNT .EQ. 0) GO TO 40
      COUNT = COUNT + RCNT
C
C        SUM FROM DENSITY MAX UP TO CRITICAL VALUE
C
      F = BINOM(N, LP1, LP2, LMP1, LMP2, XFM + 1, R)
```

```
      POWSUM = POWSUM + FSUM(XFM + 1, XCV, 1, F, FLIM, N, THETA, R,
     *         RCNT)
      COUNT = COUNT + RCNT
C
C        WORK OUT FROM RMAX ALTERNATING BETWEEN SMALLER AND LARGER R
C
   30 I = I + 1
      J = -1 * J
      R = R + I * J
      IF (R .GE. 0 .AND. R .LE. 2 * N) GO TO 10
C
C        CALCULATE ERROR BOUND AND RETURN
C
   40 ERROR = ONE - POWSUM
      IF (ONE - POWSUM .GT. (TOTAL - COUNT) * FLIM) ERROR = FLIM *
     *    (TOTAL - COUNT)
      FXPOWER = POWSUM
      RETURN
      END
```

```
      SUBROUTINE RCINT(N, RC, L, X, WT, Y, M, WORK, NS, WORK2)
C
C     ALGORITHM AS 281.1 APPL. STATIST. (1993) VOL.42, NO.1
C
C     Scaling and rounding regression coefficients to integers.
C
C     N.B. No checks of input parameters are made in this routine.
C     N.B. The published routine is in double precision contrary to
C          the rules of the journal.
C
      INTEGER N, L, M, NS
      DOUBLE PRECISION RC(N), X(N,L), WT(L), Y(L), WORK(3,M), WORK2(N)
C
C     Local variables
C
      INTEGER I, J, NRC, JMIN, JMAX, K, NSC, KK
      LOGICAL OK
      DOUBLE PRECISION ZERO, EPS, HALF, BIG, SCMIN, RCSUM, RCT, SCMAX,
     +    WTSUM, YSS, T, SC, RCSS, RCSP, SDP, SD, SDPP, SCP
C     EXTERNAL RCINT2     ! This is a subroutine, not a function!
C
      DATA ZERO,  EPS,    HALF,  BIG
     +    /0.D0, 1.D-08, 0.5D0, 1.D08/
C
C     Identify sums and range of coefficients
C
      NS = 0
      NRC = 0
      SCMIN = BIG
      RCSUM = ZERO
      DO 10 I = 1, N
        RCT = ABS( RC(I) )
        IF ( RCT .EQ. ZERO ) GO TO 10
        NRC = NRC + 1
        RCSUM = RCSUM + RCT
        IF ( RCT .LT. SCMIN ) SCMIN = RCT
   10 CONTINUE
      IF ( NRC .EQ. 0 ) RETURN
C
C     Derive range for scaling factor K.
C
      SCMIN = HALF / SCMIN
      SCMAX = SCMIN + M / RCSUM
C
C     Derive linear predictor, sum of squares and sum of weights.
C
      WTSUM = ZERO
      YSS = ZERO
      DO 30 I = 1, L
        IF (WT(I) .EQ. ZERO ) GO TO 30
        T = ZERO
        DO 20 J = 1, N
          T = T + RC(J) * X(J,I)
   20   CONTINUE
        Y(I) = T
        WTSUM = WTSUM + WT(I)
        YSS = YSS + T * T * WT(I)
   30 CONTINUE
      IF ( WTSUM .LE. ZERO .OR. YSS .LE. ZERO ) RETURN
      NSC = 0
      DO 40 I = 1, M
```

```
          WORK(1,I) = ZERO
       40 CONTINUE
C
C     For each coefficient in turn ...
C
      DO 130 I = 1, N
        RCT = ABS( RC(I) )
        IF ( RCT .EQ. ZERO ) GO TO 130
C
C     ... derive scaling factors at integer boundaries ...
C
        JMIN = NINT( RCT * SCMIN + EPS )
        JMAX = NINT( RCT * SCMAX + EPS )
        DO 120 J = JMIN, JMAX
          SC = ( J - HALF ) / RCT
          IF ( SC .LT. SCMIN .OR. SC .GT. SCMAX ) GO TO 120
C
C     ... derive sum of squares and sum of products for each
C     scaling factor.
C
          RCSS = ZERO
          RCSP = ZERO
          DO 50 K = 1, N
            WORK2(K) = NINT( RC(K) * SC + SIGN( EPS, RC(K) ) )
       50   CONTINUE
C
          DO 70 K = 1, L
            IF ( WT(K) .EQ. ZERO ) GO TO 70
            T = ZERO
            DO 60 KK = 1, N
              T = T + WORK2(KK) * X(KK,K)
       60     CONTINUE
            RCSS = RCSS + T * T * WT(K)
            RCSP = RCSP + T * Y(K) * WT(K)
       70   CONTINUE
C
C     Sort by scaling factor.
C
          NRC = 0
       80   NRC = NRC + 1
          IF ( NRC .GT. M ) GO TO 120
          IF ( ABS( SC - WORK(1,NRC) ) .LT. SC * EPS ) GO TO 120
          IF ( NRC .LE. NSC .AND. SC .GT. WORK(1,NRC) ) GO TO 80
          IF ( NSC .LT. M ) NSC = NSC + 1
          IF ( NRC .EQ. NSC ) GO TO 110
          DO 100 K = NSC - 1, NRC, -1
            DO 90 KK = 1, 3
              WORK(KK, K+1) = WORK(KK,K)
       90     CONTINUE
      100   CONTINUE
      110   WORK(1,NRC) = SC
          WORK(2,NRC) = RCSS
          WORK(3,NRC) = RCSP
      120 CONTINUE
      130 CONTINUE
C
C     Identify optimal values for scaling factor.
C
      IF ( NSC .LE. 1 ) RETURN
      SDP = BIG
      DO 150 K = 1, NSC - 1
```

```
        RCSS = WORK(2,K)
        RCSP = WORK(3,K)
        IF ( RCSP .EQ. ZERO ) RETURN
C
C     Adjust scaling factor and derive residual sum of squares.
C
        SC = RCSS / RCSP
        IF ( SC .LT. WORK(1,K) ) THEN
          SC = WORK(1,K)
        ELSE IF ( SC .GT. WORK(1,K+1) ) THEN
          SC = WORK(1,K+1)
        END IF
        SD = YSS - 2 * RCSP / SC + RCSS / ( SC * SC )
        IF ( SD .LT. EPS * YSS ) SD = ZERO
        IF ( K .EQ. 1 .OR. SDP .GT. SD .OR. .NOT. OK
     +      .OR. ( NS .GT. 0 .AND. SDP .GE. SDPP ) ) GO TO 140
        NS = NS + 1
        WORK(1,NS) = SCP
        WORK(2,NS) = SQRT( SDP / WTSUM )
        SDPP = SDP
  140   OK = SD .LT. SDP
        SDP = SD
        SCP = SC
  150 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE RCINT2(N, RC, INT, SCALE)
C
C     ALGORITHM AS 281.2 APPL. STATIST. (1993) VOL.42, NO.1
C
      INTEGER N, INT(N)
      DOUBLE PRECISION RC(N), SCALE
C
C     Local variables
C
      INTEGER I
      DOUBLE PRECISION EPS
      DATA EPS / 1.D-08 /
C
      DO 10 I = 1, N
        INT(I) = NINT( RC(I) * SCALE + SIGN( EPS, RC(I) ) )
   10 CONTINUE
C
      RETURN
      END
```

```
C
C THE FOLLOWING ROUTINES ARE WRITTEN IN DOUBLE PRECISION. IT IS STRONGLY
C RECOMMENDED THAT THEY BE RUN THIS WAY. IF IT IS DESIRED TO RUN THEM IN
C SINGLE PRECISION, ALL "IMPLICIT DOUBLE PRECISION (A-H,O-Z)" LINES SHOULD
C BE DELETED, ALL DOUBLE PRECISION CONSTANTS IN DATA STATEMENTS SHOULD BE
C CONVERTED TO REAL CONSTANTS, AND THE VALUE OF "TOLER" SET IN THE DATA
C STATEMENT IN ROUTINE MVELMS SHOULD BE CHANGED TO "1.E-5"
C
      SUBROUTINE MVELMS (X,Y,NCAS,NPRE,IOPTN,MAXTRY,NCLOW,NCHIGH,
     1 COEFFS,EPRMIN,RESID,ROBDSQ,CVEMIN,DATA,IWORK,WORK,NVDIM,
     2 NODIM,IFAULT)
C
C     ALGORITHM AS 282 APPL.STATIST. (1993), VOL.42, NO.2
C
C     HIGH BREAKDOWN REGRESSION AND MULTIVARIATE ESTIMATION
C
C
C ROUTINE TO CALCULATE LEAST MEDIAN OF SQUARES REGRESSION, MINIMUM VOLUME
C ELLIPSOID, AND ASSOCIATED STATISTICS
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(NODIM,NVDIM), Y(NODIM), COEFFS(NVDIM,*),EPRMIN(*),
     1 RESID(NODIM,*), CVEMIN(*), ROBDSQ(NODIM,*), DATA(0:NVDIM,*),
     2 IWORK(*), WORK(*)
C
C     OPTIONAL ADDITIONAL ARGUMENTS FOR MORE SPECIALIZED USE
C
C     DIMENSION XINV(NVDIM,NVDIM,*),ELSCAL(*),LMSBAS(NVDIM,*),
C    1 MVEBAS(NVDIM,*)
C
      LOGICAL LMS,MVE,ISINT,EXH,EXACT
      DATA TOLER/1.D-9/,BIG/1.D30/,NFRESH/50/,ZERO/0.D0/,HALF/0.5D0/,
     1 ONE/1.D0/,TEN/10.D0/
C
C     EXTRACT THE OPTIONS REQUESTED
C
      ITEMP = IOPTN
      LMS = MOD(ITEMP,2) .EQ. 0
      ITEMP = ITEMP / 2
      MVE = MOD(ITEMP,2) .EQ. 0
      ITEMP = ITEMP / 2
      ISINT = MOD(ITEMP,2) .EQ. 0
      ITEMP = ITEMP / 2
      EXH = MOD(ITEMP,2) .EQ. 0
      IF (ISINT) THEN
        NVAR=NPRE+1
        ADDCON=ONE/FLOAT(NVAR)
        RPP1=SQRT(ADDCON)
        INT=1
      ELSE
        NVAR=NPRE
        ADDCON=ZERO
        RPP1=ONE
        INT=0
      END IF
C
C CHECK FAULT PARAMETERS
C
      IFAULT=0
      IF ((NCLOW.LT.NVAR).OR.(NCLOW.GT.NCHIGH).OR.(NCHIGH.
     1  GT.NCAS)) IFAULT=2
```

```
      IF (NCAS.LT.NVAR) IFAULT=1
      IF ((ISINT).AND.(NVAR.EQ.1).AND.(MVE)) IFAULT=IFAULT+4
      IF (IFAULT.GT.0) GO TO 490
      EXACT=.FALSE.
      J2=NCAS
      J3=2*NCAS
      J4=3*NCAS
      I2=NVAR
      I3=2*NVAR
      I4=3*NVAR
      I5=I4+NCAS
      NVR2=2*NVAR
      DETER=RPP1
      POWMED=FLOAT(NPRE)*HALF
      DO 10 I=1,NVAR
 10   IWORK(I)=NCAS+I-NVAR
      IPTR=1
      NCRANG=NCHIGH-NCLOW+1
      DO 20 I=1,NCRANG
      CVEMIN(I)=BIG
      EPRMIN(I)=BIG
 20   CONTINUE
      LFRESH=0
      NCOUNT=0
C
C TARGET VARIABLE, IF PRESENT, IS TREATED IN STANDARDIZED FORM SO THAT EXACT
C FIT CAN BE DETECTED MORE EASILY
C
      IF (LMS) THEN
        IXLO=0
        DO 30 I=1,NCAS
 30     WORK(I)=Y(I)
C
C THE USER-SUPPLIED SUBROUTINE SORTSUB(RA,N,ILOW) SORTS ENTRIES ILOW+1 THROUGH
C ILOW+N IN ASCENDING ORDER FROM A VECTOR RA
C
        CALL SORTSUB(WORK,NCAS,0)
        YMED=(WORK(NCAS/2+1)+WORK((NCAS+1)/2))*HALF
        DO 40 I=1,NCAS
 40     WORK(I)=ABS(Y(I)-YMED)
        CALL SORTSUB(WORK,NCAS,0)
        YMAD=(WORK(NCAS/2+1)+WORK((NCAS+1)/2))*HALF
        IF (YMAD.EQ.ZERO) YMAD=ONE
      ELSE
        IXLO=1
      END IF
      DETADJ=ZERO
      DO 60 J=1,NPRE
      DO 50 I=1,NCAS
 50   WORK(I)=ABS(X(I,J))
      CALL SORTSUB(WORK,NCAS,0)
      WORK(J4+J+INT)=(WORK(NCAS/2+1)+WORK((NCAS+1)/2))*HALF
      IF (WORK(J4+J+INT).EQ.ZERO) WORK(J4+J+INT)=WORK(NCAS)
      DETADJ=DETADJ+LOG10(WORK(J4+J+INT))
 60   CONTINUE
      IF (ISINT) WORK(J4+1)=ONE
C
C DATA IS TRANSFERRED TO WORKAREA; INITIAL SIMPLEX TABLEAU IS SET UP
C
      CALL REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,NVAR,WORK,
     1  J4,IWORK,I2,I3,I5,0,ISINT,INT,LMS,DETER,LFRESH,TOLER)
```

```
C
C INITIAL BASIS IS SET UP. CHECKS ARE MADE THAT INITIAL BASIS MEMBERS ARE IN
C GENERAL POSITION
C
      IF (EXH) THEN
        IWORK(I2+1)=1
        DO 100 I=1,NVAR
 70     J=IWORK(I2+I)
        DO 80 II=I,NVAR
        IF (ABS(DATA(II,J)).GE.TOLER) THEN
          CALL PIVOT(DATA,IXLO,NVAR,NCAS,IWORK,I5,II,J,DETER,
     1      NVDIM)
          IWORK(I3+I)=J
          IF (II.NE.I) CALL SWAP(DATA,NVAR,NCAS,II,I,NVDIM)
          GO TO 90
        END IF
 80     CONTINUE
        IF (J.EQ.NCAS) THEN
          IFAULT=IFAULT+8
          IF (IFAULT.GE.24) IFAULT=IFAULT-16
          GO TO 490
        ELSE
          IFAULT=16
          IWORK(I2+I)=IWORK(I2+I)+1
          GO TO 70
        END IF
 90     IF (I.LT.NVAR) IWORK(I2+I+1)=IWORK(I2+I)+1
100     CONTINUE
        IPTR=NVAR
        GO TO 230
      ELSE
        DO 110 I=1,NCAS
110     IWORK(I4+I)=I
        CALL PERM(IWORK,I4,NCAS,0)
        INXPTR=0
        DO 140 I=1,NVAR
120     INXPTR=INXPTR+1
        J=IWORK(I4+INXPTR)
        DO 130 II=I,NVAR
        IF (ABS(DATA(II,J)).GE.TOLER) THEN
          CALL PIVOT(DATA,IXLO,NVAR,NCAS,IWORK,I5,II,J,DETER,
     1      NVDIM)
          IWORK(I3+I)=J
          IF (II.NE.I) CALL SWAP(DATA,NVAR,NCAS,II,I,NVDIM)
          IWORK(I2+I)=J
          GO TO 140
        END IF
130     CONTINUE
        IF (INXPTR.EQ.NCAS) THEN
          IFAULT=IFAULT+8
          IF (IFAULT.GE.24) IFAULT=IFAULT-16
          GO TO 490
        ELSE
          IFAULT=16
          GO TO 120
        END IF
140     CONTINUE
        GO TO 230
      END IF

C
```

```
C MAIN ANALYSIS LOOP. GENERATE ALL SUBSETS (IF EXH) OR A SUBSET (IF
C NOT EXH)
C
150   IF (EXH) THEN
C
C IF EXHAUSTIVE, SUCCESSIVE BASES ARE CONSIDERED
C
160     IWORK(I2+IPTR)=IWORK(I2+IPTR)+1
        IF (IWORK(I2+IPTR).GT.IWORK(IPTR)) THEN
          IPTR=IPTR-1
          IF (IPTR.EQ.0) THEN
            GO TO 440
          ELSE
            GO TO 160
          END IF
        ELSE
C
C IF DATA IS NOT IN GENERAL POSITION, NEXT POSITION IN LIST MUST BE FOUND
C
          IF (ABS(DATA(IPTR,IWORK(I2+IPTR))).GE.TOLER) GO TO 210
          IF (LFRESH.GT.NVR2) THEN
            DETER=RPP1
            CALL REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,
     1        NVAR,WORK,J4,IWORK,I2,I3,I5,IPTR-1,ISINT,INT,LMS,DETER,
     2        LFRESH,TOLER)
            IF (ABS(DATA(IPTR,IWORK(I2+IPTR))).GE.TOLER) GO TO 210
            IFAULT=16
            IF (IPTR.EQ.NVAR) THEN
              GO TO 160
            ELSE
              GO TO 170
            END IF
          END IF
          IFAULT=16
          IF (IPTR.EQ.NVAR) GO TO 160
          DETER=RPP1
          CALL REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,
     1      NVAR,WORK,J4,IWORK,I2,I3,I5,IPTR-1,ISINT,INT,LMS,DETER,
     2      LFRESH,TOLER)
170       J=IWORK(I2+IPTR)
          DO 180 II=IPTR,NVAR
          IF (ABS(DATA(II,J)).GE.TOLER) THEN
            CALL PIVOT(DATA,IXLO,NVAR,NCAS,IWORK,I5,II,J,DETER,
     1        NVDIM)
            IWORK(I3+IPTR)=J
            IF (II.NE.IPTR) CALL SWAP(DATA,NVAR,NCAS,II,IPTR,NVDIM)
            GO TO 220
          END IF
180       CONTINUE
          IWORK(I2+IPTR)=IWORK(I2+IPTR)+1
          IF (IWORK(I2+IPTR).GT.IWORK(IPTR)) THEN
            IPTR=IPTR-1
            IF (IPTR.EQ.0) THEN
              GO TO 440
            ELSE
              IWORK(I2+IPTR)=IWORK(I2+IPTR)+1
            END IF
          END IF
          GO TO 170
        END IF
       ELSE
```

```
C
C IF NOT EXHAUSTIVE, THE NEXT ENTRY IN RANDOM PERMUTATION VECTOR IS ENTERED
C INTO THE BASIS
C
        IF (NCOUNT.GT.MAXTRY) GO TO 440
        IPTR=MOD(IPTR,NVAR)+1
190     INXPTR=INXPTR+1
        IF (INXPTR.GT.NCAS) THEN
C
C IF COME TO THE END OF THE PERMUTATION VECTOR, GENERATE A NEW ONE
C
          DO 200 I=1,NVAR
          IWORK(I4+NCAS-NVAR+I)=IWORK(I4+I)
          IWORK(I4+I)=IWORK(I2+I)
200       CONTINUE
          CALL PERM(IWORK,I4,NCAS,NVAR)
          INXPTR=NVAR+1
        END IF
        NCOL=IWORK(I4+INXPTR)
        IF (ABS(DATA(IPTR,NCOL)).LT.TOLER) THEN
          IF (LFRESH.GT.NVR2) THEN
            DETER=RPP1
            CALL REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,
     1        NVAR,WORK,J4,IWORK,I2,I3,I5,IPTR-1,ISINT,INT,LMS,DETER,
     2        LFRESH,TOLER)
            IF (ABS(DATA(IPTR,NCOL)).LT.TOLER) THEN
              IFAULT=16
              GO TO 190
            END IF
          ELSE
            IFAULT=16
            GO TO 190
          END IF
        END IF
        IWORK(I2+IPTR)=NCOL
      END IF
210   NCOUNT=NCOUNT+1
      LFRESH=LFRESH+1
C
C CARRY OUT NEXT PIVOT, THEREBY CREATING A NEW BASIS
C
      CALL PIVOT (DATA,IXLO,NVAR,NCAS,IWORK,I3,IPTR,
     1  IWORK(I2+IPTR),DETER,NVDIM)
C
C RECOMPUTE THE INVERSE BASIS EVERY NFRESH SIMPLEX PIVOTS
C
      IF (LFRESH.GT.NFRESH) THEN
        DETER=RPP1
        CALL REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,
     1    NVAR,WORK,J4,IWORK,I2,I3,I5,NVAR,ISINT,INT,LMS,DETER,LFRESH,
     2    TOLER)
      END IF
220   IF (EXH.AND.(IPTR.LT.NVAR)) THEN
        IPTR=IPTR+1
        IWORK(I2+IPTR)=IWORK(I2+IPTR-1)
        GO TO 150
      END IF
230   IF (LMS) THEN
C
C CHECK TO SEE IF THIS BASIS HAS A SMALLER LMS CRITERION VALUE THAN PREVIOUS
C BASES. IF THERE IS NO INTERCEPT, A PRELIMINARY COMPARISON IS MADE TO SEE IF
```

```
C THIS BASIS COULD BE OPTIMAL.
C
        IF (EXACT) GO TO 340
        IF (ISINT) THEN
          DO 240 J=1,NCAS
240       WORK(J)=DATA(0,J)
        ELSE
          DO 250 J=1,NCAS
250       WORK(J)=ABS(DATA(0,J))
          DO 270 KX=NCLOW,NCHIGH
          KXSTO=KX-NCLOW+1
          TARGET=EPRMIN(KXSTO)
          ICOUNT=0
          DO 260 J=1,NCAS
260       IF (WORK(J).LE.TARGET) ICOUNT=ICOUNT+1
          IF (ICOUNT.GE.KX) GO TO 280
270       CONTINUE
          GO TO 340
        END IF
280     CALL SORTSUB(WORK,NCAS,0)
        DO 330 KX=NCLOW,NCHIGH
        KXSTO=KX-NCLOW+1
        OFFS=ZERO
        IF (ISINT) THEN
C
C CALCULATE PROPER OFFSET FOR INTERCEPT TERM, IF REQUIRED, AND DETERMINE
C CRITERION FOR THIS BASIS
C
          SHORT=BIG
          DO 290 KKX=KX,NCAS
          IF (WORK(KKX)-WORK(KKX-KX+1).LT.SHORT) THEN
            SHORT=WORK(KKX)-WORK(KKX-KX+1)
            CRITER=SHORT*HALF
            OFFS=(WORK(KKX)+WORK(KKX-KX+1))*HALF
          END IF
290       CONTINUE
        ELSE
          CRITER=WORK(KX)
        END IF
C
C UPDATE SUMMARY STATISTICS IF THIS BASIS IS CURRENTLY OPTIMAL
C
        IF (CRITER.LT.EPRMIN(KXSTO)) THEN
          EPRMIN(KXSTO)=CRITER
C         DO 300 J=1,NVAR
C300      LMSBAS(J,KXSTO)=IWORK(I3+J)
          DO 310 J=1,NVAR
310       COEFFS(J,KXSTO)=-DATA(0,NCAS+J)*YMAD
          IF (ISINT) COEFFS(1,KXSTO)=COEFFS(1,KXSTO)+OFFS*YMAD
          DO 320 J=1,NCAS
320       RESID(J,KXSTO)=(DATA(0,J)-OFFS)*YMAD
C
C A CHECK FOR EXACT FIT IS MADE
C
          IF ((CRITER.LT.TOLER).AND.(KX.EQ.NCHIGH)) THEN
            EXACT=.TRUE.
            IF (.NOT.MVE) GO TO 460
          END IF
        END IF
330     CONTINUE
C
```

```
C IF THIS IS A UNIVARIATE LOCATION PROBLEM, NO FURTHER ANALYSIS IS NEEDED
C
        IF ((ISINT).AND.(NVAR.EQ.1)) GO TO 460
      END IF
340   IF (MVE) THEN
C
C CHECK TO SEE IF THIS BASIS HAS A SMALLER MVE CRITERION VALUE THAN PREVIOUS
C BASES. A PRELIMINARY COMPARISON IS MADE TO SEE IF THIS BASIS COULD BE
C OPTIMAL.
C
        BASVOL=LOG10(ABS(DETER))
        DO 360 J=1,NCAS
        WORK(J+J2)=-ADDCON
        DO 350 I=1,NVAR
350     WORK(J+J2)=WORK(J+J2)+DATA(I,J)**2
        IF (WORK(J+J2).GT.ZERO) THEN
          WORK(J+J2)=LOG10(WORK(J+J2))
        ELSE
          WORK(J+J2)=-BIG
        END IF
        WORK(J+J3)=WORK(J+J2)
360     CONTINUE
        DO 380 KX=NCLOW,NCHIGH
        KXSTO=KX-NCLOW+1
        TARGET=(CVEMIN(KXSTO)-BASVOL)/POWMED
        ICOUNT=0
        DO 370 J=1,NCAS
370     IF (WORK(J+J3).LE.TARGET) ICOUNT=ICOUNT+1
        IF (ICOUNT.GE.KX) GO TO 390
380     CONTINUE
        GO TO 150
390     CALL SORTSUB(WORK,NCAS,J3)
        DO 430 KX=NCLOW,NCHIGH
        KXSTO=KX-NCLOW+1
        VOLU=BASVOL+POWMED*WORK(KX+J3)
        IF (VOLU.LT.CVEMIN(KXSTO)) THEN
C
C UPDATE SUMMARY STATISTICS IF THIS BASIS IS CURRENTLY OPTIMAL
C
          CVEMIN(KXSTO)=VOLU
C         ELSCAL(KXSTO)=WORK(KX+J3)
C         DO 400 J=1,NVAR
C400      MVEBAS(J,KXSTO)=IWORK(I3+J)
          DO 410 JJ=1,NCAS
410       ROBDSQ(JJ,KXSTO)=WORK(JJ+J2)-WORK(KX+J3)
C         DO 420 J=1,NVAR
C         DO 420 JJ=1,NVAR
C420      XINV(JJ,J,KXSTO)=DATA(J,JJ+NCAS)/WORK(J4+JJ)
        END IF
430     CONTINUE
      END IF
C
C END OF MAIN PROCESSING LOOP. GO TO BEGINNING AND GENERATE ANOTHER BASIS
C
      GO TO 150
440   IF (MVE) THEN
C
C CONVERT ROBUST DISTANCES AND SCALING BACK TO ORIGINAL SCALE
C
        DO 450 K=1,NCRANG
C       ELSCAL(K)=TEN**ELSCAL(K)
```

```
        CVEMIN(K)=CVEMIN(K)+DETADJ
        DO 450 J=1,NCAS
        IF (ROBDSQ(J,K).GT.(-BIG)) THEN
          ROBDSQ(J,K)=TEN**ROBDSQ(J,K)
        ELSE
          ROBDSQ(J,K)=ZERO
        END IF
450     CONTINUE
      END IF
460   IF (LMS) THEN
        DO 480 K=1,NCRANG
        EPRMIN(K)=EPRMIN(K)*YMAD
        DO 470 I=1,NPRE
470     COEFFS(I+INT,K)=COEFFS(I+INT,K)/WORK(I+J4+INT)
480     CONTINUE
      END IF
490   RETURN
      END
      SUBROUTINE REFRESH(DATA,NODIM,NVDIM,IXLO,X,NCAS,NPRE,Y,YMAD,
     1 NVAR,WORK,J4,IWORK,I2,I3,I5,IUP,ISINT,INT,LMS,DETER,LFRESH,TOLER)
C
C SUBROUTINE TO REFRESH FIRST IUP ENTRIES OF SIMPLEX BASIS
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION DATA(0:NVDIM,*), X(NODIM,NVDIM), Y(NODIM),
     1  WORK(*),IWORK(*)
      LOGICAL ISINT,LMS
      DATA ZERO/0.D0/,ONE/1.D0/
      LFRESH=0
      IF (ISINT) THEN
        DO 10 I=1,NCAS
10      DATA(1,I)=ONE
      END IF
      DO 20 I=1,NCAS
      DO 20 J=1,NPRE
20    DATA(J+INT,I)=X(I,J)/WORK(J4+J+INT)
      DO 40 I=1,NVAR
      DO 30 J=1,NVAR
30    DATA(J,NCAS+I)=ZERO
      DATA(I,NCAS+I)=ONE
40    CONTINUE
      IF (LMS) THEN
        DO 50 I=1,NCAS
50      DATA(0,I)=Y(I)/YMAD
        DO 60 I=1,NVAR
60      DATA(0,NCAS+I)=ZERO
      END IF
      DO 80 I=1,IUP
      J=IWORK(I2+I)
      DO 70 II=I,NVAR
      IF (ABS(DATA(II,J)).GE.TOLER) THEN
        CALL PIVOT(DATA,IXLO,NVAR,NCAS,IWORK,I5,II,J,DETER,NVDIM)
        IF (II.NE.I) CALL SWAP(DATA,NVAR,NCAS,II,I,NVDIM)
        IWORK(I3+I)=J
        GO TO 80
      END IF
70    CONTINUE
80    CONTINUE
      RETURN
      END
      SUBROUTINE PIVOT(X,IXLO,NORD,NCAS,IWORK,I3,NROW,NCOL,DETER,
```

```
     1  NVDIM)
C
C SUBROUTINE TO PIVOT ENTRY CORRESPONDING TO (NROW,NCOL) INTO SIMPLEX TABLEAU
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(0:NVDIM,*),IWORK(*)
      DATA ZERO/0.D0/,ONE/1.D0/
      PIVT=X(NROW,NCOL)
      DETER=DETER*PIVT
      NHIGH=NORD+NCAS
      DO 20 J=1,NHIGH
      IF (J.NE.NCOL) THEN
        FMULT=X(NROW,J)/PIVT
        DO 10 I=IXLO,NORD
 10     IF (I.NE.NROW) X(I,J)=X(I,J)-FMULT*X(I,NCOL)
        X(NROW,J)=FMULT
      END IF
 20   CONTINUE
      DO 30 I=IXLO,NORD
 30   X(I,NCOL)=ZERO
      X(NROW,NCOL)=ONE
      IWORK(I3+NROW)=NCOL
      RETURN
      END
      SUBROUTINE PERM(INDEX,IAA,N,IAB)
C
C SUBROUTINE TO RETURN PSEUDORANDOM PERMUTATION OF N - IAB ENTRIES IN
C VECTOR INDEX STARTING AT IAA+1
C
      DIMENSION INDEX(*)
      L=0
      M=N-IAB
C
C GENERATE A RANDOM DIGIT FROM 1 TO M USING THE PSEUDORANDOM U(0,1) VARIATE
C RANDOM() (SUCH AS FROM ALGORITHM AS 183)
C
 10   J=INT(RANDOM()*FLOAT(M))+1
C
C SWAP ENTRIES IN INDEX CORRESPONDING TO J AND M, OFFSET BY IAB
C
      IF (J.NE.M) THEN
        ITEMP=INDEX(IAA+J+IAB)
        INDEX(IAA+J+IAB)=INDEX(IAA+M+IAB)
        INDEX(IAA+M+IAB)=ITEMP
      END IF
      M=M-1
      IF (M.GT.1) GO TO 10
      RETURN
      END
      SUBROUTINE SWAP(DATA,NVAR,NCAS,IR1,IR2,NVDIM)
C
C SUBROUTINE TO SWAP ROWS IR1 AND IR2 OF MATRIX DATA
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION DATA(0:NVDIM,*)
      NHIGH=NVAR+NCAS
      DO 10 I=1,NHIGH
      TEMP=DATA(IR1,I)
      DATA(IR1,I)=DATA(IR2,I)
      DATA(IR2,I)=TEMP
10    CONTINUE
```

```
      RETURN
      END
```

The file contains the following:

fishfun.c /* C-source of Fisher's symmetry test function + driver program */

groupfun.c /* C source of Pitman's 2 sample test function + driver program */

fishint.c /* exact (integer) version of fishfun.c */

groupint.c /* exact (integer) version of groupfun.c */

psydata /* data file of Rutter score data for fishfun */

agedata /* data file of age data for groupfun */

iagedat /* integer version of age data */

```
/* ------------------------------------------------------------------------*/
/* fishfun.c */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define MAXNUM 200 /* max. sample size */
float fisherprob(float *,int,float,float *, long *);

#define NAMLEN 40 /* max. length of file-name */
int compare(int *x,int *y)
        {
        return(*x-*y);
        }
main()
        {
        FILE *fp;
        char filename[NAMLEN]; /* array to hold filename */
        float a[MAXNUM],cv,p,ptol=.01; /* fractional error allowed on prob. */
        int scanval,n;
        long binsreq;

        puts("\nEnter input file name: ");
        scanf("%40s",filename); /* get file name (MAX 40 CHARS) */

        if ((fp = fopen(filename,"r")) == 0 ) /* open file */
                {
                puts ("Can't open input file"); /* unsuccessful */
                return;
                }

        n=0;
        while((scanval=fscanf(fp,"%f",&a[n])) != EOF && scanval != NULL )
                {
                printf("number %d is %f\n",n,a[n]);
                ++n;
                }
        close(fp);
        printf("read in %d numbers\n",n);

        p=fisherprob(a,n,ptol,&cv,&binsreq);
        if(p<0.)
                {
                printf("estimating probability to a CV of %f requires %ld
```

```
bins-sorry",ptol,binsreq);
                return;
                }
        printf("calculation used %ld histogram bins\n",binsreq);
        printf("1-tailed probability is %f\n",p);
        printf("Coefficient of variation on estimate of p is %f%%\n",cv);
        /* can repeat with ptol less if cv is too large */
        }

float fisherprob(float a[],int n,float ptol,float *cv,long *binsreq)
/* This function returns the 1-tailed probability from
Fisher's symmetry test, the permutation, i.e. distribution-free
analogue of the paired t-test. */

        {
        int numberpos=0,numberneg=0, *positive, *x,smallnum,topbin,
        m, bin,i,ipossum=0,inegsum=0,s,n0,n0dash,ndash,ntop;
        float possum=0,negsum=0,scale,sum,p,factor,mu,sigsq,term1,term2,tail,
*b;

        *binsreq=n;
        x=calloc(n,sizeof(int));
        positive=calloc(n,sizeof(int)); /* allocate work arrays */
        if(!x || !positive)return(-1.); /* and return -ve p-value if not
possible */

        for(i=0,mu=0.,sigsq=0.;i<n;i++){
                sigsq+=a[i]*a[i]; /* accumulate permutation variance */
                if(a[i]>=0)
                        {
                        ++numberpos; /* count number of +ve values */
                        possum+=a[i]; /* and accumulate their sum */
                        positive[i]=1; /* flag value as +ve */
                        }
                else
                                {
                        ++numberneg;
                        negsum-=a[i];
                        a[i]=-a[i];
                        positive[i]=0;
                        }
                }

        mu=(possum+negsum)/2.;
        sigsq/=4.; /* permutation mean and variance */
        sum=(possum<negsum)?possum:negsum; /* choose smaller sum as test
statistic */
        m=(possum<negsum)?numberpos:numberneg;
        factor=sqrt((n-(mu-sum)*(mu-sum)/sigsq)/48.); /*for error calculation */
        term1=mu*(mu-sum)/(ptol*sigsq);
        term2=2./ptol;
        bin=factor*((term1>term2)?term1:term2);
        /* no. of bins needed for required error on p*/

        smallnum=(numberpos<numberneg)?numberpos:numberneg;
        /* number in smaller group */
        if(smallnum>0)
                scale=bin/sum; /* scale sample values so that smaller

sum is in bin number bin */
        else
```

```
                scale=1.; /* if only one sign present calculation is trivial.
However,

just go through the usual motions */
        for (i=0;i<n;++i){
                x[i]=a[i]*scale+0.5; /* rescale sample values. Adding 0.5 rounds
to nearest integer rather than truncating*/
                if(positive[i])ipossum+=x[i];
                else
                        inegsum+=x[i];
                }
        bin=(ipossum<inegsum)?ipossum:inegsum; /* rescale number of bins */

        *binsreq=bin+2;
        b=calloc(bin+2,sizeof(float)); /* allocate work array */
        if(!b)
                return(-1.); /* and return if not enough memory */

        qsort(x,n,sizeof(int),compare); /* sort sample values */
        for(i=0;i<n;++i)
                x[i]=(x[i]<bin+1)?x[i]:bin+1; /* any sample values
                                                 greater than bin are put into the
overflow bin */
        /*      for(i=0;i<=bin+1;i++)b[i]=0.0; */
        ++b[0]; /* start histogram off - entries at zero and x0 */
        ++b[x[0]];
        n0=x[0];
        for(s=1;s<n;s++)
                {
                n0dash=n0+x[s];
                if(n0dash>bin)
                        {
                        b[bin+1]*=2.; /* just double entries in overflow bin */

for(ndash=bin;ndash>=bin-x[s]+1;ndash--)b[bin+1]+=b[ndash];
                        }/* translate histogram */
                ntop=(bin>n0dash)?n0dash:bin;
                for(ndash=ntop;ndash>=x[s];ndash--)b[ndash]+=b[ndash-x[s]];
                n0=n0dash;
                }

        tail=0.; /* count entries in tail */
        for(i=0;i<=bin;++i)
                tail+=b[i];
        p=tail/(tail+b[bin+1]);
        *cv=100*b[bin]*factor/tail;
        free(b); /* free memory used for work arrays */
        free(x);
        free(positive);
        return(p);


        }

/* ------------------------------------------------------------------------*/
/* groupfun.c */

#include <stdio.h> /* for reading data file */
#include <math.h> /* for square root */
#include <stdlib.h>
```

```
#define MAXNUM 160 /* max. sample size */

#define LEN 40 /* max. length of filename */
int compare(int *x,int *y)
        {
        return(*x-*y);
        }
main()
        {
        FILE *fp;
        int n,firstgroup[MAXNUM],scanval;
        float a[MAXNUM],p,cv,ptol=.05; /* fractional error required on prob. */
        long binsreq;
        float pitmanprob(float *,int *, int, float, float *, long *);
        char filename[LEN]; /* array to hold filename */

        puts("\nEnter input file name: ");
        scanf("%40s",filename); /* get file name (MAX 40 CHARS) */

        if ((fp = fopen(filename,"r")) == 0 ) /* open file */
                {
                puts ("Can't open input file"); /* unsuccessful */
                return;
                }

        n=0;
        while((scanval=fscanf(fp,"%f%d",&a[n],&firstgroup[n])) != EOF && scanval
!= NULL)
                {
                printf("number %d is %f group %d\n",n,a[n],firstgroup[n]);
                ++n;
                if(n>=MAXNUM)
                        {
                        printf("sample size too large. Maximum is %d\n",MAXNUM);
                        return;
                        }
                }
        close(fp);
        printf("read in %d numbers\n",n);

        p=pitmanprob(a,firstgroup,n,ptol,&cv,&binsreq); /* find tail probability
*/
        if(p<0.){
                printf("%ld array elements requested, which is too
many.\n",binsreq);
                return;
                }
        else    {
                printf("%ld array elements used\n",binsreq);
                printf("1-tailed probability is %f\n",p);
                printf("CV of estimated probability is %f %%",cv);
                }
        }

float pitmanprob(float a[],int firstgroup[],int n,float ptol,float *cv,long
*binsreq)
/* returns 1-tail probability from Pitman's 2-sample permutation test,
the distribution-free analogue of the grouped t-test */

        {
        int numberone=0,numbertwo=0,
```

```
        smallnum,k,j,smallgroup,ntop,kbot,topbin,
        bin,i,isum=0,s,n0,n0dash,ndash,m, *dope, *x;
        float onesum=0,twosum=0,scale,sum,p,tail,term1,term2,
        minnum,maxnum,onemean,twomean,mu,sigsq,em,en,factor, *b;

        *binsreq=n;
        dope=calloc(n,sizeof(int));
        x=calloc(n,sizeof(int)); /* allocate work area */
        if(!dope || !x)return(-1.); /* return -ve p value if not possible */

        for(i=0;i<n;i++){
                if(i == 0){
                        minnum=a[i]; /* find max. and min. sample values */
                        maxnum=a[i];
                        }
                else    {
                        minnum=(a[i]>minnum)?minnum:a[i];
                        maxnum=(a[i]>maxnum)?a[i]:maxnum;
                        }
                if(firstgroup[i] ==1)
                        {
                        ++numberone; /* count numbers of in each group */
                        onesum+=a[i]; /* and their sums */
                        }
                else
                                {
                        ++numbertwo;
                        twosum+=a[i];
                        }
                }

        onemean=onesum/numberone; /* find means for each group */
        twomean=twosum/numbertwo;
        smallgroup=(onemean>twomean)?2:1;
        for(i=0;i<n;i++)a[i]-=minnum; /* translate sample values to be >= 0 */
        smallnum=(numberone<numbertwo)?1:2;
        onesum-=numberone*minnum; /* and modify sums and maximum value */
        twosum-=numbertwo*minnum;
        maxnum-=minnum;

        if(smallgroup != smallnum)
                { /* if necessary, reflect values so smaller group has smaller
mean */
                for(i=0;i<n;i++)a[i]=maxnum-a[i];
                onesum=numberone*maxnum-onesum;
                twosum=numbertwo*maxnum-twosum;
                }
        sum=(smallnum == 1)?onesum:twosum;
        m=(numberone<numbertwo)?numberone:numbertwo; /* number in smaller group
*/

        em=m;
        en=n;
        for(mu=0.,sigsq=0.,i=0;i<n;i++){
                mu+=a[i];
                sigsq+=a[i]*a[i];
                for(j=i+1;j<n;j++)sigsq=sigsq-2.*a[i]*a[j]/(en-1.);
                }
        mu=(em/en)*mu;
        sigsq=(em/en)*(1.-(em/en))*sigsq; /* permutation mean and variance */
```

```
factor=sqrt((em/en)*(1.-(em/en))*(en+(mu-sum)*(mu-sum)*(en+1.)/(en*sigsq))/12.);
        term1=mu*(mu-sum)/(ptol*sigsq);
        term2=2./ptol;
        bin=factor*((term1>term2)?term1:term2); /* number of histogram bins
needed */

        scale=bin/sum; /* scale sample values so that sum of smaller sample
                                                                    is in bin
number bin */
        for (i=0;i<n;i++){
                x[i]=a[i]*scale+0.5; /* rescaling sample values.
                                                        Adding 0.5 rounds to
nearest bin rather than truncating */
                if(firstgroup[i]==smallnum)isum+=x[i];
                }
        bin=isum; /* find exact histogram bin for start of reject H0 region */


        *binsreq=(long)(bin+2)*m;
        b=calloc(*binsreq,sizeof(float)); /* allocate and zero workspace */
        if(!b)
                return(-1.); /* not enough workspace */
        qsort(x,n,sizeof(int),compare); /* sort sample */
        for(i=0;i<n;i++)
                x[i]=(x[i]<bin+1)?x[i]:bin+1; /* any sample values greater than
                        the test statistic "bin" will cause b[>bin] to be
occupied and are set
                        to bin+1, the overflow bin NOW rather than later so
as not to cause integer
                        overflow */

        for(i=0;i<m;i++)dope[i]=i*(bin+2); /* offsets to mimick 2 dim. array */
        ++b[dope[0]+x[0]]; /* start histogram off */
        /* for 2 dim. array code would be  ++b[0][x[0]];    */

        n0=x[0];
        for(s=1;s<n;s++)
                {
                kbot=(m+s-n>1)?m+s-n:1;
                n0dash=n0+x[s]; /* translate each level by x[s] */
                for(k=(s>(m-1))?(m-1):s;k>=kbot;k--) /* only build up histograms

as far as mth level, which is the one we want */
                        {
                        if(n0dash>bin)for(ndash=bin-x[s]+1;ndash<=bin+1;ndash++)
                                /* translate k-1 th level onto kth level...these
terms go into overflow bin */
                                b[dope[k]+bin+1]+=b[dope[k-1]+ndash];
                        /* for 2 dim. array...    b[k][bin+1]+=b[k-1][ndash]; */
                        ntop=(bin>n0dash)?n0dash:bin;
                        for(ndash=x[s];ndash<=ntop;ndash++){
                                /* these terms just translate along, but may go
into overflow bin */
                                b[dope[k]+ndash]+=b[dope[k-1]+ndash-x[s]];
                                /* for 2 dim. array...
b[k][ndash]+=b[k-1][ndash-x[s]]; */
                                }
                        }

                ++b[dope[0]+x[s]]; /* add new term to lowest level */
                /* for 2 dim. array...  ++b[0][x[s]]; */
```

```
                n0=n0dash;
                }

          tail=0.; /* find tail probability */
          for(i=0;i<=bin;i++)
                /* for 2 dim. array...   tail+=b[m-1][i]; */
                tail+=b[dope[m-1]+i];

          *cv=100.*b[dope[m-1]+bin]*factor/tail; /* find coefficient of variation
*/
          /* for 2d array...   *cv=100.*b[m-1][bin]*factor/tail; */
          free(x); /* free off workspace */
          free(dope);
          free(b);
          return(tail/(tail+b[dope[m-1]+bin+1]));
          }

/* ---------------------------------------------------------------------*/
/* fishint.c */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define MAXNUM 200 /* max. sample size */
float fishprob();

#define NAMLEN 40 /* max. length of file-name */
int comp(int *x,int *y)
{
return(*x-*y);
}
main()
      {
      FILE *fp;
      char filename[NAMLEN]; /* array to hold filename */
      float cv,p;
      int scanval,n,a[MAXNUM];
      long binsreq;

      puts("\nEnter input file name: ");
      scanf("%40s",filename); /* get file name (MAX 40 CHARS) */

      if ((fp = fopen(filename,"r")) == 0 ) /* open file */
            {
            puts ("Can't open input file"); /* unsuccessful */
            return;
            }

      n=0;
      while((scanval=fscanf(fp,"%d",&a[n])) != EOF && scanval != NULL )
            {
            printf("number %d is %d\n",n,a[n]);
            ++n;
            }
      close(fp);
      printf("read in %d numbers\n",n);

      p=fishprob(a,n,&binsreq);
      if(p<0.)
            {
```

```
            printf("calculating probability requires %ld bins-sorry",binsreq);
            return;
            }
      printf("calculation used %ld histogram bins\n",binsreq);
      printf("1-tailed probability is %f\n",p);
      }

float fishprob(int a[],int n,long *binsreq)
/* This function returns the 1-tailed probability from
Fisher's symmetry test, the permutation, i.e. distribution-free
analogue of the paired t-test. */

      {
      long bin,i,s,n0,n0dash,ndash,ntop;
      long possum=0,negsum=0,sum;
      float p,tail;
float *b;

      for(i=0;i<n;i++){
            if(a[i]>=0)
                  {
                  possum+=a[i]; /* and accumulate their sum */
                  }
            else
                        {
                  negsum-=a[i];
                  a[i]=-a[i];
                  }
            }

      sum=(possum<negsum)?possum:negsum; /* choose smaller sum as test statistic
*/
      bin=sum;
      /* no. of bins needed */
b=calloc(bin+2,sizeof(float));
      *binsreq=bin+2;
      if(!b)
            return(-1.);

      qsort(a,n,sizeof(int),comp);
      for(i=0;i<n;++i)
            a[i]=(a[i]<bin+1)?a[i]:bin+1; /* any sample values
                              greater than bin are put into the overflow bin */
      for(i=0;i<=bin+1;i++)b[i]=0.0;
      ++b[0]; /* start histogram off - entries at zero and x0 */
      ++b[a[0]];
      n0=a[0];
      for(s=1;s<n;s++)
            {
            n0dash=n0+a[s];
            if(n0dash>bin)
                  {
                  b[bin+1]=2*b[bin+1]; /* just double entries in overflow bin */
                  for(ndash=bin;ndash>=bin-a[s]+1;ndash--)b[bin+1]+=b[ndash];
                  }/* translate histogram */
            ntop=(bin>n0dash)?n0dash:bin;
            for(ndash=ntop;ndash>=a[s];ndash--)b[ndash]+=b[ndash-a[s]];
            n0=n0dash;
            }

      tail=0.; /* count entries in tail */
```

```
        for(i=0;i<=bin;++i)
              tail+=b[i];
        p=tail/(tail+b[bin+1]);
        free(b); /* free work space array */
        return(p);


        }

/* ------------------------------------------------------------------------*/
/* groupint.c */

#include <stdio.h> /* for reading data file */
#include <math.h> /* for square root */
#include <stdlib.h>

#define MAXNUM 200 /* max. sample size */

#define LEN 40 /* max. length of filename */
int comp(int *x,int *y)
{
return(*x-*y);
}
void main()
        {
        FILE *fp;
        int n,firstgroup[MAXNUM],scanval,x[MAXNUM];
        long binsreq;
        float p;
        float grouprob();
        char filename[LEN]; /* array to hold filename */

        puts("\nEnter input file name: ");
        scanf("%40s",filename); /* get file name (MAX 40 CHARS) */

        if ((fp = fopen(filename,"r")) == 0 ) /* open file */
              {
              puts ("Can't open input file"); /* unsuccessful */
              return;
              }

        n=0;
        while((scanval=fscanf(fp,"%d%d",&x[n],&firstgroup[n])) != EOF && scanval
!= NULL)
              {
              printf("number %d is %d group %d\n",n,x[n],firstgroup[n]);
              ++n;
              if(n>=MAXNUM)
                    {
                    printf("sample size too large. Maximum is %d\n",MAXNUM);
                    return;
                    }
              }
        close(fp);
        printf("read in %d numbers\n",n);

        p=grouprob(firstgroup,x,n,&binsreq); /* find tail probability */
        if(p<0.){
              printf("%ld array elements requested, which is too
many.\n",binsreq);
              return;
```

```
            }
      else  {
            printf("%ld array elements used\n",binsreq);
            printf("1-tailed probability is %f\n",p);
            }
      }

float grouprob(int firstgroup[],int x[],int n,long *binsreq)
/* returns 1-tail probability from Pitman's 2-sample permutation test. */

      {
      int numberone=0,numbertwo=0,
      smallnum,k,j,smallgroup,ntop,kbot,topbin,
      bin,i,s,n0,n0dash,ndash,m,onesum=0,twosum=0,sum,minnum,maxnum;
      float p,tail,onemean,twomean;
int *dope;
float *b;
*binsreq=n;
dope=calloc(n,sizeof(int));
if(!dope)return(-1.);
      for(i=0;i<n;i++){
            if(i == 0){
                  minnum=x[i]; /* find max. and min. sample values */
                  maxnum=x[i];
                  }
            else  {
                  minnum=(x[i]>minnum)?minnum:x[i];
                  maxnum=(x[i]>maxnum)?x[i]:maxnum;
                  }
            if(firstgroup[i] ==1)
                  {
                  ++numberone; /* count numbers of in each group */
                  onesum+=x[i]; /* and their sums */
                  }
            else
                        {
                  ++numbertwo;
                  twosum+=x[i];
                  }
            }

      onemean=onesum/numberone; /* find means for each group */
      twomean=twosum/numbertwo;
      smallgroup=(onemean>twomean)?2:1;
      for(i=0;i<n;i++)x[i]-=minnum; /* translate sample values to be >= 0 */
      smallnum=(numberone<numbertwo)?1:2;
      onesum-=numberone*minnum; /* and modify sums and maximum value */
      twosum-=numbertwo*minnum;
      maxnum-=minnum;

      if(smallgroup != smallnum)
            { /* if necessary, reflect values so smaller group has smaller mean
*/
            for(i=0;i<n;i++)x[i]=maxnum-x[i];
            onesum=numberone*maxnum-onesum;
            twosum=numbertwo*maxnum-twosum;
            }
      sum=(smallnum == 1)?onesum:twosum;
      m=(numberone<numbertwo)?numberone:numbertwo; /* number in smaller group */

bin=sum;
```

```
      qsort(x,n,sizeof(int),comp);
      for(i=0;i<n;i++)
            x[i]=(x[i]<bin+1)?x[i]:bin+1; /* any sample values greater than
            the test statistic "bin" will cause b[>bin] to be occupied and are
set
            to bin+1, the overflow bin NOW rather than later so as not to cause
integer
            overflow */
      *binsreq=(bin+2)*m;
b=calloc(*binsreq,sizeof(float));
      if(!b)
            return(-1.); /* not enough workspace */
/*      for(i=0;i<*binsreq;i++)
            b[i]=0.0;  initialise work array */
      for(i=0;i<m;i++)dope[i]=i*(bin+2); /* offsets to mimic 2 dim. array */
      ++b[dope[0]+x[0]]; /* start histogram off */
      /* for 2 dim. array code would be  ++b[0][x[0]];    */

      n0=x[0];
      for(s=1;s<n;s++)
            {
            kbot=(m+s-n>1)?m+s-n:1;
            n0dash=n0+x[s]; /* translate each level by x[s] */
            for(k=(s>(m-1))?(m-1):s;k>=kbot;k--) /* only build up histograms
                                    as far as mth level, which is the one we
want */
                  {
                  if(n0dash>bin)for(ndash=bin-x[s]+1;ndash<=bin+1;ndash++)
                        /* translate k-1 th level onto kth level...these terms
go into overflow bin */
                        b[dope[k]+bin+1]+=b[dope[k-1]+ndash];
                  /* for 2 dim. array...    b[k][bin+1]+=b[k-1][ndash]; */
                  ntop=(bin>n0dash)?n0dash:bin;
                  for(ndash=x[s];ndash<=ntop;ndash++){
                        /* these terms just translate along, but may go into
overflow bin */
                        b[dope[k]+ndash]+=b[dope[k-1]+ndash-x[s]];
                        /* for 2 dim. array...  b[k][ndash]+=b[k-1][ndash-x[s]];
*/
                        }
                  }

            ++b[dope[0]+x[s]]; /* add new term to lowest level */
            /* for 2 dim. array...  ++b[0][x[s]]; */
            n0=n0dash;
            }

      tail=0.; /* find tail probability */
      for(i=0;i<=bin;i++)
            /* for 2 dim. array...   tail+=b[m-1][i]; */
            tail+=b[dope[m-1]+i];

      return(tail/(tail+b[dope[m-1]+bin+1]));
      }

/* ---------------------------------------------------------------------*/
/* psydata */

-1 -2 1 -7 -5 0 3 0 -6 -1 -5 -1 -1 5 -1 -8 0 5 -2 -2 0 -4 -4 2
-1 -1 -1 4 0 1 -2 -2 -2 1 0 -4 -5 0 -5 -1 5 2 0
```

```
/* ------------------------------------------------------------------------*/
/* agedata */

1.5  2
2.5 2
1 2
6 2
7 2
2 2
9 2
8 1
5 1
6 1
5 1
6 1
7 1
4 1
9 2
6 2
9 2
7 2
4 2
4 2
5 1
5 1
4 1
7 1
9 1
6 1
6 1
6 1
5 2
3 2
1 2
3 2
5 1
4.5 1
6 1
4 1
6 1
4.5 1
5 1
9 1
5 2
3 2
3.5 2
5 2
7 2
3 2
3.5 2
7 2
3 2

/* ------------------------------------------------------------------------*/
/* iagedat */

3  2
5 2
2 2
12 2
```

```
14 2
4 2
18 2
16 1
10 1
12 1
10 1
12 1
14 1
8 1
18 2
12 2
18 2
14 2
8 2
8 2
10 1
10 1
8 1
14 1
18 1
12 1
12 1
12 1
10 2
6 2
2 2
6 2
10 1
9 1
12 1
8 1
12 1
9 1
10 1
18 1
10 2
6 2
7 2
10 2
14 2
6 2
7 2
14 2
6 2
```

```
      Program driver
C
C             Program to test and illustrate algorithm BEST.  Program
C             DRIVER is not necessary for using BEST.
C
C             At the prompt, input values for p, q, and x.  The value of
C             p is the dimension of the covariance matrix and the value
C             of q is the degrees of freedom.  The program outputs an
C             approximation to Pr(GG < x), where GG is the Greenhouse-
C             Geiser epsilon.
C
C             The program is intended to be run interactively.  It expects
C             that unit 5 is input and unit 6 is output.
C
      real x,best,pval
      integer ip,iq
      write(6,10)
 10   format(/,' To exit program, type Control C',/)
 20   write(6,30)
 30   format(/,' Input p, q, and x')
      read(5,*,err=50) ip,iq,x
      pval=best(x,ip,iq,ifault)
      write(6,40) pval,ifault
 40   format(' Pr(GG < x) = ',f7.5,', Ifault = ',i1)
      goto 20
 50   stop
      end



      real function best(x,ip,iq,ifault)
C
C      ALGORITHM AS 284, APPL. STATIST. (1993), VOL. 42, NO. 3
C
C             Computes The Null Distribution Function of The Test
C             Statistic For The Locally Best Invariant Test of
C             Sphericity And Additivity
C
      real x,p,q,a,b,t,s1,s2,s3,s4,s5,zero,one,two,three,four,epsi,
     $ delta(4),fact(0:6),pocham,zreal,alogam,betain,beta,small,lag,
     $ nbit,base
      intrinsic abs,exp,log,sign,sqrt
C
C             Fixed Constants
C
      data zero,one,two,three,four,small,(fact(i),i=0,6) /0.0e0,
     $ 1.0e0, 2.0e0, 3.0e0, 4.0e0, 1.0e-2, 1.0e0, 1.0e0, 2.0e0,
     $ 6.0e0, 2.4e1, 1.2e2, 7.2e2/
C
C             Machine Dependent Constants
C
      data base,nbit/2.0e0, 23.0e0/
      zreal(i)=real(i)
      epsi=one/base**(nbit-one)
C
C             Check Validity of Input Arguments
C
      best=one
      ifault=4
      if(ip.le.1.or.iq.lt.ip) return
      p=zreal(ip)
```

```
      q=zreal(iq)
      ifault=5
      if(x.ge.one-epsi) return
      best=zero
      if(x*p.le.one+epsi) return
C
C             Use Chebyshev's Inequality to Check for Probabilities
C             Near 0 or 1
C
      ifault=3
      t=(one/x-one)/(p-one)
      a=(p+two)*(p*(p*q-q+two*four)-three*four)/(four*p*(q+two))
      b=(q-one)*a*p/(p+two)
      s1=a/(a+b)-t
      s2=(log(a)+log(b)-log(a+b+one))/two-log(a+b)-log(small)
      if(exp(s2).le.abs(s1)) then
         if(s1.lt.zero) then
            return
         else
            best=one
            return
         endif
      endif
      ifault=0
C
C             Use Exact Beta(1,(q-1)/2) Distribution if p = 2
C
      if(ip.eq.2) then
          best=exp((q-one)*log(one-t)/two)
C
C             Check Magnitude of Higher Order Terms.
C
      else
          call moment(p,q,delta)
          s1=abs(delta(1))+abs(delta(2))+abs(delta(3))+abs(delta(4))
          if(s1.ge.epsi**(one/three)) then
C
C             Use Exact Distribution from John (1972) if p = 3
C             and Magnitude of Higher Order Terms is Large
C
              if(ip.eq.3) then
                  a=q/two
                  b=q-one
C
C             Compute log Beta Function.  No Need to Check Error
C             Indicator, it is Known to be 0.
C
                  beta=alogam(a,ier)+alogam(b,ier)-alogam(a+b,ier)
                  s1=sqrt(t)
                  s2=(one+two*s1)/three
                  best=betain(s2,a,b,beta,ier)
     $             -three*exp(a*log(s2)+b*log(one-s2)-beta)
                  if(t.lt.one/four-epsi) then
                      s3=(one-two*s1)/three
C
C             Compute Probability.  No Need to Check Error
C             Indicator, it is Known to be 0.
C
                      best=best-betain(s3,a,b,beta,ier)
     $                 +three*exp(a*log(s3)+b*log(one-s3)-beta)
                  end if
```

```
                    best=one-best
C
C               Use Six moment Jacobi Polynomial Expansion if p > 3
C               And Magnitude of Higher Order Terms is Large
C
                else
                    ifault=1
C
C               Compute log Beta Function And Probability.  No Need to
C               Check Error Indicator, it is Known to be 0.
C
                    beta=alogam(a,ier)+alogam(b,ier)-alogam(a+b,ier)
                    best=betain(t,a,b,beta,ier)
                    do 30 n=3,6
                        s1=pocham(a,n-1)+pocham(a+b+zreal(n),n-1)
     $                   -pocham(a+b,n)+pocham(a+b+zreal(n-1),n-1)
     $                   -pocham(b,n)-log(fact(n))
                        s2=zero
                        do 10 i=0,n-3
                            s3=pocham(zreal(n-i+1),i)-log(fact(i))
     $                      +log(a+zreal(n-1))
     $                      -pocham(a+b+zreal(2*n-i-1),i)
     $                      +pocham(a+b+zreal(n-i),i)
     $                      +log(abs(delta(n-i-2)))
     $                      +log(a+b+zreal(2*n-2))
                            s2=s2+exp(s3)*sign(one,delta(n-i-2))
     $                      *(-one)**i
 10                     continue
                        if(abs(s2).le.epsi) goto 30
                        s3=zero
                        do 20 i=1,n
                            s4=pocham(zreal(n-i+1),i)-log(fact(i))
     $                      -pocham(a+b+zreal(2*n-i-1),i)
     $                      +pocham(a+b+zreal(n-i),i)
                            do 20 j=1,i
                                s5=pocham(a+b,n-j)-pocham(a,n-j+1)
     $                          +(a+zreal(n-j))*log(t)+b*log(one-t)
     $                          -beta
                                s3=s3+exp(s5+s4+log(a+b+zreal(2*n-1)))
     $                          *(-one)**i
 20                     continue
                        if(abs(s3).le.epsi) goto 30
                        s4=exp(s1+log(abs(s2))+log(abs(s3)))
                        best=best+s4*sign(one,s2*s3)
 30                 continue
                    best=one-best
                end if
C
C           Use Three moment Laguerre Polynomial Approximation if
C           Magnitude of Higher Order Terms is Small
C
            else
                best=lag(x,ip,iq,ifault)
            end if
        end if
        if(best.lt.zero) best=zero
        if(best.gt.one) best=one
        return
        end
```

```
      real function lag(x,ip,iq,ifault)
C
C             Computes Three moment Laguerre Polynomial Approximation
C             to the Distribution Function
C
      real x,p,q,pq,a,b,s1,s2,t,delta,gammad,alogam,zreal,
     $ zero,one,two,three,four,six,cn(9),cd(20)
      intrinsic abs,exp,log,sign
C
C             Fixed Constants
C
      data zero,one,two,three,four,six/ 0.0e0, 1.0e0, 2.0e0, 3.0e0,
     $ 4.0e0, 6.0e0/
      data (cn(i),i=1,9)/ 0.0e0, 1.2e2, 1.8e1, 4.8e1, -1.2e1, -3.0e0,
     $ 4.0e1, 0.0e0, -1.0e0/
      data (cd(i),i=1,20)/ 1.152e3, 3.84e2, -1.84e2, 1.2e1, -1.152e3,
     $ -6.72e2, 6.4e1, 8.0e0, -2.88e2, -1.92e2, 5.4e1, 9.0e0, 2.88e2,
     $ 2.96e2, 4.8e1, 2.0e0, 6.4e1, 8.8e1, 1.8e1, 1.0e0/
      zreal(i)=real(i)
      ifault=2
      p=zreal(ip)
      q=zreal(iq)
      pq=p*q
      a=two*four
      b=-log(a/pq+one)-log((a+two)/pq+one)-three*log(pq)
     $ +log(q-one)+log(q+two)+log(zreal(32))
      b=-exp(b)
C
C             Compute Standardized Third moment
C
      n1=0
      n2=0
      s1=zero
      do 20 j=0,2
          a=zero
          do 10 k=0,2
              n1=n1+1
              a=a/pq+cn(n1)
 10       continue
          s1=s1/p+a
 20   continue
      s2=zero
      do 40 j=0,4
          a=zero
          do 30 k=0,3
              n2=n2+1
              a=a/pq+cd(n2)
 30       continue
          s2=s2/p+a
 40   continue
      delta=b*s1/s2
      a=(p-one)*(p+two)*(p*q+four)*(p*q+six)
     $ /(four*(q-one)*(q+two)*p**2)
      b=(p*q+two)*a/(p+two)
      t=b*(one/x-one)/(p-one)
C
C             Compute Log Gamma Function.  No Need to Check Error
C             Indicator, it is Known to be 0.
C
      s1=a*log(t)-t-alogam(a,ier)+log(abs(delta))-log(six)
      s2=exp(s1+two*log(abs(t-a-two)))-exp(s1+log(abs(a-two)))
```

```
C
C             Compute Probability.  No Need to Check Error
C             Indicator, it is Known to be 0.
C
      lag=one-gammad(t,a,ier)+s2*sign(one,delta)
      return
      end

      real function pocham(x,i)
C
C             Compute Natural Log of Pochhammer's Symbol: (x) sub i
C
      real x,zreal,zero
      intrinsic log
      data zero /0.0e0/
      zreal(i)=real(i)
      pocham=zero
      if(i.eq.0) return
      do 10 j=1,i
 10   pocham=pocham+log(x+zreal(i)-zreal(j))
      return
      end

      Subroutine moment(p,q,delta)
      real p,q,pq,num,den,s,zero,one,two,three,four,cn(240),cd(154),
     $ g(4),f(4),delta(4)
      intrinsic exp,log
C
C             Compute Standardized moments of LBI Statistic
C
C             Fixed Constants
C
      data zero,one,two,three,four /0.0e0, 1.0e0, 2.0e0, 3.0e0, 4.0e0/
      data (g(i),i=1,4) /3.2e1, 1.28e2, 6.4e1, 1.28e2/
      data (cn(i),i=1,110)/ 0.0e0, -2.8e1, 0.0e0, 1.0e1, -5.0e0, -1.0e0,
     $ 0.0e0, -1.12896e5, 1.94784e5, -3.3888e4, -3.2448e4, 0.0e0,
     $ 4.032e4, -9.7968e4, 7.6856e4, 2.096e3, -1.732e4, -1.008e3,
     $ 3.552e3, -1.4604e4, 1.002e4, 2.42e3, -3.268e3, -3.28e2, 3.28e2,
     $ -4.34e2, 1.65e2, 3.19e2, -2.45e2, -3.7e1, -8.2e1, 1.16e2, -2.9e1,
     $ 1.0e0, -5.0e0, -1.0e0, 0.0e0, -3.90168576e9, 1.4764032e10,
     $ -1.7053949952e10, 3.842187264e9, 3.411984384e9, -9.30594816e8,
     $ -3.29416704e8, 0.0e0, 1.3934592e9, -5.74580736e9, 1.010539008e10,
     $ -8.198281728e9, 7.0906368e8, 2.382832128e9, -4.64467968e8,
     $ -2.78255616e8, -9.068544e6, -3.787776e7, -5.95800576e8,
     $ 1.800523008e9, -1.554594816e9, -2.6491904e7, 6.44580096e8,
     $ -9.4626048e7, -9.609088e7, -5.374464e6, -1.525248e6, 4.829952e6,
     $ 7.0553536e7, -9.968272e7, -1.3282944e7, 8.1219456e7, -8.673152e6,
     $ -1.7555424e7, -1.35904e6, -6.517632e6, 1.8864096e7, -2.4434256e7,
     $ 6.285264e6, 2.517872e6, 3.552544e6, -3.14064e5, -1.801424e6,
     $ -1.7992e5, 2.71712e5, 1.948336e6, -4.017808e6, 1.359176e6,
     $ 8.19112e5, -2.2064e5, -1.072e4, -9.9608e4, -1.2696e4, -9.7584e4,
     $ 2.53584e5, -2.20896e5, 1.557e4, 7.9326e4, -2.446e4, -2.58e3,
     $ -2.55e3, -4.1e2, 5.056e3, -1.1944e4, 8.55e3, -2.077e3, 8.75e2,
     $ -3.7e2, -6.0e1, -2.5e1, -5.0e0, 0.0e0, -4.12018016256e13/
      data (cn(i),i=111,170) / 2.544631676928e14, -5.8340956962816e14,
     $ 5.799309115392e14, -1.667991011328e14, -1.0232596463616e14,
     $ 5.784648155136e13, 9.4170710016e12, -5.5084843008e12,
     $ -1.03842054144e12, 0.0e0, 1.4714929152e13, -8.270448427008e13,
     $ 2.1422392786944e14, -3.36219910668288e14, 2.93880843804672e14,
     $ -6.9645704552448e13, -7.5702487277568e13, 3.8893641228288e13,
     $ 9.646000324608e12, -4.669805494272e12, -1.170392481792e12,
```

```
$ -2.569273344e10, -4.8813539328e12, 1.241719676928e13,
$ 4.57903300608e12, -4.737808889856e13, 6.1560777191424e13,
$ -1.9353568569344e13, -2.04090638848e13, 1.1918790731776e13,
$ 3.827214399488e12, -1.7948152832e12, -5.66963986432e11,
$ -2.1138112512e10, 1.2911837184e11, 2.002896101376e12,
$ -5.52099796992e12, 2.399627067392e12, 4.85868078592e12,
$ -4.4841321088e12, -2.005616851968e12, 2.151339017216e12,
$ 7.65947608064e11, -4.0405611776e11, -1.56711717888e11,
$ -7.879186432e9, -7.0632824832e10, 3.56417660928e11,
$ -1.269488833536e12, 1.782649662464e12, -3.94358012672e11,
$ -8.37563249024e11, 1.6834389056e11, 2.62308470784e11,
$ 7.3064108032e10, -5.86671328e10, -2.7335101312e10,
$ -1.713773312e9, 3.708598272e10, -5.6949588992e10/
 data (cn(i),i=171,240)/ -1.01034381696e11, 3.1574732672e11,
$ -1.58358678688e11, -1.11234267456e11, 7.1553216224e10,
$ 2.4591867072e10, -2.46103264e8, -5.746261696e9, -3.116678112e9,
$ -2.36430272e8, -2.898390528e9, 3.32693408e9, -1.5851460992e10,
$ 3.3324926128e10, -1.7291739616e10, -1.069476608e10,
$ 8.859817856e9, 2.11305632e9, -8.7746096e8, -3.95701248e8,
$ -2.31185472e8, -2.1039408e7, 4.6900352e8, -3.39609568e8,
$ -8.90319408e8, 1.59414452e9, -7.83590964e8, -5.89984244e8,
$ 5.16508044e8, 1.60988444e8, -9.91687e7, -2.0192284e7,
$ -1.070398e7, -1.17714e6, -7.802352e7, 1.48137968e8, -2.5871288e7,
$ -7.2898636e7, 2.0969796e7, -5.79076e5, 5.97858e6, 8.300068e6,
$ -4.904884e6, -7.84316e5, -2.86284e5, -3.8408e4, 4.14784e6,
$ -1.4920968e7, 2.0030724e7, -1.0498098e7, -8.32893e5, 2.605797e6,
$ -5.28245e5, 1.13753e5, -9.6255e4, -1.6865e4, -4.115e3, -6.75e2,
$ -8.3304e4, 2.76992e5, -3.30074e5, 1.73416e5, -5.7129e4, 2.7445e4,
$ -6.681e3, 1.85e2, -6.75e2, -1.45e2, -2.5e1, -5.0e0/
 data (cd(i),i=1,124)/ 5.76e2, -7.68e2, -1.44e2, 2.56e2, 6.4e1,
$ -1.92e2, 6.4e1, 4.8e1, 4.8e1, 1.6e1, 1.2e1, 8.0e0, 9.0e0, 2.0e0,
$ 1.0e0, 1.3824e4, -3.456e4, 1.344e4, 1.632e4, -4.48e3, -3.84e3,
$ -5.12e2, -1.0368e4, 1.4016e4, 2.528e3, -2.352e3, -2.096e3,
$ -1.152e3, -1.92e2, 2.208e3, -5.92e2, -4.56e2, -8.6e2, -3.6e2,
$ -1.08e2, -2.4e1, -1.2e2, -9.2e1, -1.1e2, -3.7e1, -2.1e1, -3.0e0,
$ -1.0e0, 3.31776e5, -1.327104e6, 1.456128e6, 1.8432e5, -8.0256e5,
$ -6.144e4, 1.61792e5, 4.9152e4, 4.096e3, -4.42368e5, 1.179648e6,
$ -5.28384e5, -4.66944e5, 4.7104e4, 1.08544e5, 6.912e4, 2.048e4,
$ 2.048e3, 1.98144e5, -2.79552e5, -4.6336e4, 1.6896e4, 5.5136e4,
$ 3.7824e4, 1.2128e4, 3.072e3, 3.84e2, -3.3792e4, 8.192e3, 6.4e3,
$ 1.6896e4, 8.064e3, 3.712e3, 1.056e3, 1.92e2, 3.2e1, 1.68e3,
$ 1.408e3, 1.752e3, 7.2e2, 4.41e2, 1.0e2, 3.8e1, 4.0e0, 1.0e0,
$ 7.962624e6, -4.644864e7, 9.068544e7, -4.902912e7, -3.9020544e7,
$ 3.236352e7, 1.3006848e7, -5.44768e6, -3.35872e6, -5.7344e5,
$ -3.2768e4, -1.65888e7, 7.133184e7, -8.626176e7, -8.41728e5,
$ 3.8708736e7, 5.992192e6, -5.542144e6, -4.4032e6, -1.6896e6,
$ -3.072e5, -2.048e4, 1.271808e7, -3.621888e7, 1.8376704e7,
$ 1.2434432e7, 1.097984e6, -3.187328e6, -3.21408e6, -1.3088e6,
$ -3.4944e5, -6.4e4, -5.12e3, -4.3776e6, 6.517248e6, 9.78944e5/
 data (cd(i),i=125,154)/ 8.0128e4, -1.564096e6, -1.15376e6,
$ -5.1376e5, -1.768e5, -3.824e4, -6.4e3, -6.4e2, 6.48576e5,
$ -1.53792e5, -1.22944e5, -3.89792e5, -2.0356e5, -1.1454e5,
$ -3.78e4, -1.06e4, -2.28e3, -3.0e2, -4.0e1, -3.024e4, -2.7024e4,
$ -3.4624e4, -1.612e4, -1.041e4, -2.961e3, -1.225e3, -2.1e2,
$ -6.0e1, -5.0e0, -1.0e0/
 pq=p*q
 s=two*four
 f(1)=log(q-one)+log(q+two)+log(p-two)-log(pq)
$ -log(p-one)-log(pq+s)-log(pq+s+two)
 do 10 i=2,4
     s=s+four
```

```
          f(i)=f(i-1)+three*log(p)+two*log(q)
     $     -log(p-one)-log(pq+s)-log(pq+s+two)
 10   continue
      do 20 i=1,4
 20   f(i)=exp(f(i)+log(g(i)))*(-one)**i
      n1=0
      n2=0
      do 70 ii=1,4
          i=ii+2
          num=zero
          ind=3*(i-3)+1
          do 40 j=0,ind
              s=zero
              do 30 k=0,ind+1
                  n1=n1+1
                  s=s/p+cn(n1)
 30           continue
              num=num/pq+s
 40       continue
          den=zero
          ind=i-1
          do 60 j=0,ind
              s=zero
              do 50 k=0,2*ind
                  n2=n2+1
                  s=s/p+cd(n2)
 50           continue
              den=den/pq+s
 60       continue
          delta(ii)=f(ii)*num/den
 70   continue
      return
      end

C
C
C     ****************************************************************
C     *                                                              *
C     *    The remaining subprograms are included for referees use   *
C     *    only.  They are not intended for publication.             *
C     *                                                              *
C     ****************************************************************
C
C
      real function betain(x,p,q,beta,ifault)
C
C         Algorithm AS 63 Appl. Statist. (1973) Vol. 22, p. 409
C
C         Computes Incomplete Beta Function Ratio for Arguments
C         x between zero and one, p and q positive.
C         Log of Complete Beta Function, Beta, is Assumed to
C         be Known.
C
      logical index
      real acu,ai,beta,cx,one,p,pp,psq,q,qq,rx
      real temp,term,x,xx,zero,zabs,zexp,zlog
C
C         Define Accuracy and Initialize
C
      data acu,one,zero/1.0e-7, 1.0e0, 0.0e0/
      zabs(X)=abs(X)
```

```
      zexp(X)=exp(X)
      zlog(X)=alog(X)
      betain=x
C
C         Test for Admissibility of Arguments
C
      Ifault=1
      If(p.le.zero.or.q.le.zero) return
      Ifault=2
      If(x.lt.zero.or.x.gt.one) return
      Ifault=0
      if(x.eq.0.or.x.eq.one) return
C
C         Change Tail If Necessary and Determine s
C
      psq=p+q
      cx=one-x
      if(p.ge.psq*x) goto 1
      xx=cx
      cx=x
      pp=q
      qq=p
      index=.true.
      goto 2
 1    xx=x
      pp=p
      qq=q
      index=.false.
 2    term=one
      ai=one
      betain=one
      ns=qq+cx*psq
C
C         Use Reduction Formulae of Soper
C
      rx=xx/cx
 3    temp=qq-ai
      if(ns.eq.0) rx=xx
 4    term=term*temp*rx/(pp+ai)
      betain=betain+term
      temp=zabs(term)
      if(temp.le.acu.and.temp.le.acu*betain) goto 5
      ai=ai+one
      ns=ns-1
      if(ns.ge.0) goto 3
      temp=psq
      psq=psq+one
      goto 4
C
C         Calculate Result
C
 5    betain=betain*zexp(pp*zlog(xx)+(qq-one)*zlog(cx)-beta)/pp
      if(index) betain=one-betain
      return
      end

      real function gammad(x,p,ifault)
C
C          Algorithm AS239 Appl. Statist. (1988) Vol. 37, No. 3
C
C          Computation of the Incomplete Gamma Integral
```

```
C
      integer ifault
      real pn1,pn2,pn3,pn4,pn5,pn6,x,tol,oflo,xbig,arg,c,rn,p,a,
     $ b,one,zero,alogam,an,two,elimit,plimit,alnorm,three,nine
      parameter (zero=0.0e0, one=1.0e0, two=2.0e0, oflo=1.3e19,
     $ three=3.0e0, nine=9.0e0, tol=1.0e-7, plimit=1.0e3,
     $ xbig=1.0e6, elimit=-8.8e1)
      intrinsic abs,log,exp,sqrt,min
      external alogam,alnorm
C
      gammad=zero
C
C        Check that we have valid values for X and P
C
      if(p.le.zero.or.x.lt.zero) then
      ifault=1
      return
      endif
      ifault=0
      if(x.eq.zero) then
      gammad=zero
      return
      endif
C
C         Use a Normal Approximation if P .gt. PLIMIT
C
      if(p.gt.plimit) then
      pn1=three*sqrt(p)*((x/p)**(one/three)+one/(nine*p)-one)
      gammad=alnorm(pn1,.false.)
      return
      endif
C
C         If X is extremely large compared to P then set GAMMAD to ONE
C
      if(x.gt.xbig) then
      gammad=one
      return
      endif
C
      if(x.le.one.or.x.lt.p) then
C
C        Use Pearson's series expansion (Note that P is not large enough
C        to force overflow in ALOGAM)  No need to test IFAULT on exit
C        since P .gt. xero
C
      arg=p*log(x)-x-alogam(p+one,ifault)
      c=one
      gammad=one
      a=p
 40   a=a+one
      c=c*x/a
      gammad=gammad+c
      if(c.gt.tol) goto 40
      arg=arg+log(gammad)
      gammad=zero
      if(arg.ge.elimit) gammad=exp(arg)
C
      else
C
C        Use a continued fraction expansion
C
```

```
      arg=p*log(x)-x-alogam(p,ifault)
      a=one-p
      b=a+x+one
      c=zero
      pn1=one
      pn2=x
      pn3=x+one
      pn4=x*b
      gammad=pn3/pn4
 60   a=a+one
      b=b+two
      c=c+one
      an=a*c
      pn5=b*pn3-an*pn1
      pn6=b*pn4-an*pn2
      if(abs(pn6).gt.zero) then
      rn=pn5/pn6
      if(abs(gammad-rn).le.min(tol,tol*rn)) goto 80
      gammad=rn
      endif
C
      pn1=pn3
      pn2=pn4
      pn3=pn5
      pn4=pn6
      if(abs(pn5).ge.oflo) then
C
C         Re-scale terms in continued fraction if terms are large
C
      pn1=pn1/oflo
      pn2=pn2/oflo
      pn3=pn3/oflo
      pn4=pn4/oflo
      endif
      goto 60
 80   arg=arg+log(gammad)
      gammad=one
      if(arg.ge.elimit) gammad=one-exp(arg)
      endif
C
      return
      end

      real function alnorm(x,upper)
C
C        Algorithm AS 66 Appl. Statist. (1973) Vol. 22, p. 424
C
C        Evaluates the tail of the standardized normal curve
C        from x to infinity if .upper. is true or from minus
C        infinity to x if upper is .false.
C
      real ltone,utzero,zero,half,one,con,a1,a2,a3,a4,a5,a6,a7,
     $ b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,x,y,z,zexp
      logical upper,up
C
C        Ltone and utzero must be set to suit the particular computer
C        (See introductory text)
C
      data ltone,utzero/5.6e0, 1.2e1/
      data zero,half,one,con/ 0.0e0, 5.0e-1, 1.0e0, 1.28e0/
      data a1,a2,a3,a4,a5,a6,a7/ 3.98942280444e-1, 3.99903438504e-1,
```

```
     $ 5.75885480458e0, 2.98213557808e1, 2.62433121679e0,
     $ 4.86959930692e1, 5.92885724438e0/
      data b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12/ 3.98942280385e-1,
     $ 3.8052e-8, 1.00000615302e0, 3.98064794e-4, 1.98615381364e0,
     $ 1.51679116635e-1, 5.29330324926e0, 4.8385912808e0,
     $ 1.51508972451e1, 7.42380924027e-1, 3.0789933034e1,
     $ 3.99019417011e0/
C
      zexp(z)=exp(z)
C
      up=upper
      z=x
      if(z.ge.zero) goto 10
      up=.not. up
      z=-z
 10   if(z.le.ltone.or.up.and.z.le.utzero) goto 20
      alnorm=zero
      goto 40
 20   y=half*z*z
      if(z.gt.con) goto 30
C
      alnorm=half-z*(a1-a2*y/(y+a3-a4/(y+a5+a6/(y+a7))))
      goto 40
C
 30   alnorm=b1*zexp(-y)/(z-b2+b3/(z+b4+b5/(z-b6+b7/
     $ (z+b8-b9/(z+b10+b11/(z+b12))))))
C
 40   if(.not.up) alnorm=one-alnorm
      return
      end
      real function alogam(x,ifault)
C
C            Algorithm ACM 291, Comm. ACM. (1966) Vol. 9, p. 684
C
C            Evaluates Natural Logarithm of Gamma(x)
C
      real a1,a2,a3,a4,a5,f,x,y,z,zlog,half,zero,one,seven
C
C         The Following Constants are Alog(2pi)/2,
C         1/1680, 1/1260, 1/360, and 1/12
C
      data a1,a2,a3,a4,a5/ 9.18938533204673e-1, 5.95238095238e-4,
     $ 7.93650793651e-4, 2.777777777778e-3,
     $ 8.3333333333333e-2/
      data half,zero,one,seven/ 0.5, 0.0, 1.0, 7.0/
C
      zlog(f)=alog(f)
C
      alogam=zero
      ifault=1
      if(x.le.zero) return
      ifault=0
      y=x
      f=zero
      if(y.ge.seven) goto 30
      f=y
 10   y=y+one
      if(y.ge.seven) goto 20
      f=f*y
      goto 10
 20   f=-zlog(f)
```

```
   30   z=one/(y*y)
        alogam=f+(y-half)*zlog(y)-y+a1
       $ +(((-a2*z+a3)*z-a4)*z+a5)/y
        return
        end
```

ALGORITHM AS 285

This file contains in order:
1. Some notes supplied by the author of the algorithm.
2. A MAKEFILE.
3. A driver program.
4. The algorithm.
5. User functions for various regions of integration - a cube, an ellipsoid, a rectangle, a sphere, a convex polynomial and a star shape.
6. The input data file FORT.1 needed by the driver program.

The sections are separated by the following:

```
C------------------------------------------------------------------------
```

1. Some notes supplied by the author of the algorithm.

The subroutines in this file compute the probability that a multivariate normal random vector falls within a prescribed region A. A is input to the program via a user-defined function subprogram.

The function should begin as follows:

```
      real function f(n,v)
      integer n
      real v(n)

      lines calculating f, which should take the value:
            = 0 if v is on the boundary of A
            < 0 if v is inside A
            > 0 if v is outside A
```

The following functions in this directory may be used to describe the boundary. Each must be altered to provide the specific boundary (or, the covariance matrix input into the driving program may be scaled appropriately).

sphere	A is sphere of radius sqrt(radsq)
cube	A is a cube with infinity-norm cubsz
rect	x is in A if a(i) < x(i) < b(i) for all i
ellipsoid	The boundary of A is x'ax + bx - c = 0
convpoly	A is a convex polytope with faces described by the inequalities a(1,i)*x(1) + a(2,i)*x(2) +...+ a(n,i)*x(n) <= 1 for i=1,...,p.
pentagram	A is a five-pointed star

Once an appropriate boundary function is selected or written, the program should be reloaded. This may be done quickly by editing the file "Makefile" so that the filename listed after "FNCN =" is the object file of the boundary function, e.g.

```
      FNCN = sphere.o
```

To load the file, type

```
      make FFLAGS=-O

and type

      mulnor

to execute the program.

All input to the driving program should be in file fort.1, of the form

n          order of covariance matrix
stdev      desired standard deviation of estimate
rhigh      radius of sphere circumscribing region A
init       initial number of simulations to be run
ix,iy,iz   three integers between 1 and 50000 to initialize
           random normal generator
covar      lower triangle of the symmetric covariance matrix,
           stored row-wise as a one-dimensional array.

C------------------------------------------------------------------------

2. A MAKEFILE

FNCN = cube.o

MULNOR =  mulnor.o chol.o zerovc.o chiprb.o orthp.o

RANDOM = rnortm.o normal.o alnorm.o random.o ppnd.o

AUX =  drive.o tdiag.o lrvt.o

OBJECTS = $(MULNOR) $(RANDOM) $(AUX) $(FNCN)

mulnor: $(OBJECTS)
      f77  $(OBJECTS) -O0  -o mulnor

clean:
      rm *.o

C------------------------------------------------------------------------

3. A driver program.

C  DRIVING PROGRAM
C
      REAL COVAR(1275),WORK(50,52),T(1275)
      INTEGER ISEED,N,IFAULT,J,IX,IY,IZ,ITER,INIT
      REAL F,SDEV,PROB,RHIGH,ETA
      COMMON /RAND/ IX,IY,IZ
      EXTERNAL F
      DATA ETA /1.0E-5/
C
C READ IN COVARIANCE AND INITIALIZE RANDOM NORMAL GENERATOR.
C
C With many Fortran compilers, you will need to insert a line such as:
C     OPEN(1, file='FORT.1', status='old')
C before reading from unit 1.
C
      READ (1,*) N,SDEV,RHIGH,INIT,IX,IY,IZ
      READ (1,*) (COVAR(J),J=1,N*(N+1)/2)
C
```

```
C  FIND SMALLEST NONZERO EIGENVALUE OF COVARIANCE MATRIX
C
      NN = N*(N+1)/2
      CALL CHOL(COVAR,N,NN,T,NULLTY,IFAULT)
      CALL TDIAG(N,ETA,COVAR,WORK(1,1),WORK(1,2),WORK(1,3),IFAULT)
      CALL LRVT(N,ETA,WORK(1,1),WORK(1,2),WORK(1,3),IFAULT)
      RHIGH = RHIGH/SQRT(WORK(NULLTY+1,1))
C
      CALL MULNOR(COVAR,N,F,SDEV,RHIGH,INIT,ISEED,T,WORK,PROB,
     *    ITER,IFAULT)
      PRINT *, 'ESTIMATED PROBABILITY IS', PROB,'WITH IFAULT = ',IFAULT
      PRINT *, 'ITER = ',ITER
      STOP
      END

      SUBROUTINE TDIAG(N,TOL,A,D,E,Z,IFAULT)
C
      REAL A(*),D(N),E(N),F,G,H,HH,TOL,Z(N,N),ZERO,
     *            ONE,ZSQRT
      INTEGER I,J,K,L,N,IFAULT,I1,J1
C
      DATA ZERO, ONE /0.0, 1.0/
C
      ZSQRT(H) = SQRT(H)
C
      IFAULT = 1
      IF (N .LE. 1) RETURN
      IFAULT = 0
      K = 0
      DO 10 I=1,N
      DO 10 J=1,I
      K = K+1
10    Z(I,J) = A(K)
      I=N
      DO 70 I1=2,N
      L = I-2
      F = Z(I,I-1)
      G=ZERO
      IF (L .LT. 1) GOTO 25
      DO 20 K=1,L
20    G=G+Z(I,K)**2
25    H=G+F*F
C
      IF (G .GT. TOL) GOTO 30
      E(I)=F
      D(I)=ZERO
      GOTO 65
30    L=L+1
      G=ZSQRT(H)
      IF (F .GE. ZERO) G=-G
      E(I) = G
      H = H-F*G
      Z(I,I-1)=F-G
      F=ZERO
      DO 50 J=1,L
      Z(J,I) = Z(I,J)/H
      G = ZERO
C
      DO 40 K=1,J
40    G = G+Z(J,K) *Z(I,K)
      IF (J .GE. L)GOTO 47
```

```
      J1  = J+1
      DO 45 K=J1,L
45    G=G + Z(K,J)*Z(I,K)
C
47    E(J)=G/H
      F = F+G*Z(J,I)
50    CONTINUE
C
      HH = F/(H+H)
C
      DO 60 J=1,L
      F=Z(I,J)
      G=E(J)-HH*F
      E(J)=G
      DO 60 K=1,J
      Z(J,K) = Z(J,K) - F*E(K) - G*Z(I,K)
60    CONTINUE
      D(I)=H
65    I=I-1
70    CONTINUE
      D(1) = ZERO
      E(1) = ZERO
C
      DO 110 I=1,N
      L=I-1
      IF (D(I) .EQ. ZERO .OR. L .EQ. 0) GO TO 100
      DO 90 J=1,L
      G=ZERO
      DO 80 K=1,L
80    G=G + Z(I,K)*Z(K,I)
      DO 90 K=1,L
      Z(K,J) = Z(K,J) - G*Z(K,I)
90    CONTINUE
100   D(I) = Z(I,I)
      Z(I,I) = ONE
      IF (L .EQ. 0) GO TO 110
      DO 105 J=1,L
      Z(I,J) = ZERO
      Z(J,I) = ZERO
105   CONTINUE
110   CONTINUE
      RETURN
      END


      SUBROUTINE LRVT(N,PRECIS,D,E,Z,IFAULT)
C
      REAL B,C,D(N),E(N),F,G,H,P,PR,PRECIS,R,S,
     *          Z(N,N),ZERO,ONE,TWO,ZABS,ZSQRT
      INTEGER I,J,K,L,N,IFAULT,MITS,N1,JJ,M1,M,I1
C
      DATA MITS,ZERO,ONE,TWO /30,0.0,1.0,2.0/
      ZABS(B) = ABS(B)
      ZSQRT(B) = SQRT(B)
C
      IFAULT = 2
      IF (N .LE. 1) RETURN
      IFAULT = 1
      N1 = N-1
      DO 10 I=2,N
10    E(I-1) = E(I)
```

```
      E(N) = ZERO
      B = ZERO
      F = ZERO
      DO 90 L=1,N
      JJ=0
      H = PRECIS * (ZABS(D(L)) + ZABS(E(L)))
      IF (B .LT. H) B = H
C
      DO 20 M1=L,N
      M=M1
      IF (ZABS(E(M)) .LE. B) GOTO 30
20    CONTINUE
30    IF (M .EQ. L) GOTO 85
40    IF (JJ .EQ. MITS) RETURN
      JJ = JJ+1
C
      P = (D(L+1) - D(L))/(TWO*E(L))
      R = ZSQRT(P*P + ONE)
      PR = P + R
      IF (P .LT. ZERO) PR = P - R
      H = D(L) - E(L)/PR
      DO 50 I=L,N
50    D(I) = D(I)-H
      F=F+H
C
      P=D(M)
      C=ONE
      S=ZERO
      M1=M-1
      I=M
      DO 80 I1=L,M1
      J=I
      I=I-1
      G=C*E(I)
      H=C*P
      IF(ZABS(P) .GE. ZABS(E(I))) GOTO 60
      C = P/E(I)
      R = ZSQRT(C*C+ONE)
      E(J) = S*E(I)*R
      S = ONE/R
      C = C/R
      GOTO 70
60    C=E(I)/P
      R=ZSQRT(C*C+ONE)
      E(J) = S*P*R
      S=C/R
      C=ONE/R
70    P=C*D(I)-S*G
      D(J)=H+S*(C*G+S*D(I))
C
      DO 80 K=1,N
      H=Z(K,J)
      Z(K,J)=S*Z(K,I) + C*H
      Z(K,I) = C*Z(K,I) - S*H
80    CONTINUE
      E(L)=S*P
      D(L)=C*P
      IF (ZABS(E(L)) .GT. B)GOTO 40
85    D(L) = D(L) + F
90    CONTINUE
C
```

```
      DO 120 I=1,N1
      K=I
      P=D(I)
      I1=I+1
      DO 100 J=I1,N
      IF (D(J) .GE. P) GOTO 100
      K=J
      P=D(J)
100   CONTINUE
      IF (K .EQ. I) GOTO 120
      D(K) = D(I)
      D(I) = P
      DO 110 J=1,N
      P=Z(J,I)
      Z(J,I)=Z(J,K)
      Z(J,K)=P
110   CONTINUE
120   CONTINUE
      IFAULT=0
      RETURN
      END


C------------------------------------------------------------------------------

4. The algorithm.

      SUBROUTINE MULNOR(COVAR,N,F,SDEV,RHIGH,INIT,ISEED,T,WORK,PROB,
     *   ITER,IFAULT)
C
C     ALGORITHM AS285 APPL. STATIST. (1993) VOL.42, NO.3
C
C  FINDS THE PROBABILITY THAT A NORMALLY DISTRIBUTED RANDOM
C  N-VECTOR WITH MEAN 0 AND COVARIANCE COVAR FALLS
C  IN AREA ENCLOSED BY THE EXTERNAL USER-DEFINED FUNCTION F.
C
      INTEGER N,ISEED,IFAULT,INIT,INIT1,
     *          K,II,KK,Q,ITER,NN,NULLTY,I,J,MAXITR
      REAL F,SDEV,RHIGH,PROB,P,VAR,ZERO,COX,
     *       COVAR(*), WORK(N,*),T(*)
C
      EXTERNAL F
      DATA ZERO,MAXITR /0.0,10000/
C
      IFAULT = 0
      INIT1 = INIT
      IF (INIT .LT. 25) INIT1 = 25
      IF (INIT .GT. 1000) INIT1 = 1000
      COX = 1.0 + 2.0/INIT1
C
C  FIND CHOLESKY DECOMPOSITION OF COVARIANCE MATRIX
C
      NN = N*(N+1)/2
      CALL CHOL(COVAR,N,NN,T,NULLTY,IFAULT)
      Q = N - NULLTY
      IFAULT = IFAULT*IFAULT
      IF (IFAULT .NE. 0) RETURN
C
C  TRANSPOSE CHOLESKY FACTOR, OMITTING COLUMNS OF ZEROES.
C  RESULTING VECTOR IS PACKED FORM OF T'J,
C  WRITTEN ROWWISE BEGINNING WITH LAST ROW.
```

```
C
      DO 10 I = 1,N
      DO 10 J = 1,I
         WORK(J,I) = ZERO
10    CONTINUE
      II = 0
      DO 20 I = 1,N
      DO 20 J = 1,I
         II = II + 1
         WORK(I,J) = T(II)
20    CONTINUE
      J = 0
      DO 40 I = N,1,-1
         IF (J .EQ. NULLTY) GO TO 50
         IF (WORK(I,I) .EQ. ZERO) THEN
            J = J+1
            DO 30 K = I+1,N
            DO 30 KK = I, N-J
              WORK(K,KK) = WORK(K,KK+1)
30          CONTINUE
         END IF
40    CONTINUE
50    CONTINUE
      K = 0
      DO 60 J = N,1,-1
      DO 60 I = 1,MIN0(J,Q)
         K = K + 1
         T(K) = WORK(J,I)
60    CONTINUE
C
C  TAKE PILOT SAMPLE; ESTIMATE VARIANCE OF PROB FROM ORTHP.
C
      PROB = ZERO
      VAR = ZERO
      DO 70 K=1,INIT1
         CALL ORTHP(F,RHIGH,N,Q,T,WORK,P,ISEED,IFAULT)
         IF (IFAULT .NE. 0) RETURN
         PROB = PROB + P
         VAR = VAR + P*P
70    CONTINUE
      PROB = PROB/INIT1
      VAR = VAR/INIT1 - PROB*PROB
C
      ITER = INT(COX*VAR/(SDEV*SDEV)) + 1
      IF (ITER .GT. MAXITR) IFAULT = 5
C
      DO 80 K=INIT1+1,ITER
         CALL ORTHP(F,RHIGH,N,Q,T,WORK,P,ISEED,IFAULT)
         IF (IFAULT .NE. 0) RETURN
         PROB = PROB + (P - PROB)/K
80    CONTINUE
C
      RETURN
      END

      SUBROUTINE CHOL(A,N,NN,U,NULLTY,IFAULT)
      REAL A(NN),U(NN),ETA,ETA2,X,W,ZERO,ZABS,ZSQRT
      INTEGER N,NN,NULLTY,IFAULT
      INTEGER J,K,II,ICOL,L,KK,M,IROW,I
C
      DATA ETA,ZERO /1.0E-05,0.0/
```

```
      ZABS(X) = ABS(X)
      ZSQRT(X) = SQRT(X)
C
      IFAULT = 1
      IF (N .LE. 0) RETURN
      IFAULT = 3
      IF (NN .NE. N*(N+1)/2) RETURN
      IFAULT = 2
      NULLTY = 0
      J = 1
      K = 0
      ETA2 = ETA*ETA
      II = 0
      DO 80 ICOL = 1,N
      II = II + ICOL
      X = ETA2 * A(II)
      L = 0
      KK = 0
      DO 40 IROW = 1, ICOL
      KK = KK + IROW
      K = K+1
      W = A(K)
      M = J
      DO 10 I = 1, IROW
      L = L + 1
      IF (I .EQ. IROW) GO TO 20
      W = W - U(L) * U(M)
      M = M + 1
10    CONTINUE
20    IF (IROW .EQ. ICOL) GO TO 50
      IF (U(L) .EQ. ZERO) GO TO 30
      U(K) = W / U(L)
      GO TO 40
30    IF (W * W .GT. ZABS(X * A(KK))) RETURN
      U(K) = ZERO
40    CONTINUE
50    IF (ZABS(W) .LE. ZABS(ETA * A(K))) GO TO 60
      IF (W .LT. ZERO) RETURN
      U(K) = ZSQRT(W)
      GO TO 70
60    U(K) = ZERO
      NULLTY = NULLTY + 1
70    J = J + ICOL
80    CONTINUE
      IFAULT = 0
      RETURN
      END

      SUBROUTINE ORTHP(F,RHIGH,N,Q,T,Z,PROB,ISEED,IFAULT)
C
C  FINDS THE PSEUDO-MONTE CARLO ESTIMATE OF PROB USING AN
C  ORTHONORMAL SYSTEM OF VECTORS.
C
      INTEGER N,ISEED,IFAULT,Q,I,J,K,JJ,II,QP1,QP2
      REAL T(*),RHIGH,Z(N,*),PROB,S,A,B,R,RT2RCP,ZERO,
     *           F,CHIPRB,FAB
C
      EXTERNAL F
      DATA ZERO,RT2RCP /0.0,0.707106781/
C
      PROB = ZERO
```

```
      QP1 = Q+1
      QP2 = Q+2
C
C  PUT RANDOMLY ORIENTED ORTHONORMAL SYSTEM OF Q-VECTORS IN Z.
C
      CALL RNORTM(N,Q, QP1, 1, 1, Z(1,QP1), 1, Z(1,QP1), ISEED, Z,
     *   Z(1,QP1), FAB, Z(1,QP2), Z(1,QP2), IFAULT)
C
C  REPLACE EACH COLUMN OF Z BY TRANS(CHOLESKY FACTOR)*J*COLUMN OF Z.
C
      DO 30 I=1,Q
        JJ = 0
        DO 20 J = N,1,-1
           S = ZERO
           DO 10 K=1,MIN0(J,Q)
              JJ = JJ + 1
              S = S + T(JJ)*Z(K,I)
10         CONTINUE
           Z(J,I) = S
20      CONTINUE
30    CONTINUE
C
C  ESTIMATE THE PROBABILITY THAT N(0,COVAR) IS WITHIN THE
C  BOUNDARY GIVEN BY F.
C
      DO 40 I = 1,Q
         A = ZERO
         B = RHIGH
         CALL ZEROVC (F,R,A,B,N,Z(1,I),Z(1,QP2),IFAULT)
         PROB = PROB - CHIPRB(R*R,Q)
40    CONTINUE
C
      DO 60 I = 1,Q
         A = ZERO
         B = RHIGH
         DO 50 K = 1,N
           Z(K,QP1) = -Z(K,I)
50       CONTINUE
         CALL ZEROVC (F,R,A,B,N,Z(1,QP1),Z(1,QP2),IFAULT)
         PROB = PROB - CHIPRB(R*R,Q)
60    CONTINUE
C
      DO 80 I = 1,Q-1
      DO 80 J = I+1,Q
      DO 80 JJ= -1,1,2
      DO 80 II = -1,1,2
          DO 70 K=1,N
            Z(K,QP1) = RT2RCP*(Z(K,I)*II + Z(K,J)*JJ)
70        CONTINUE
          A = ZERO
          B = RHIGH
          CALL ZEROVC (F,R,A,B,N,Z(1,QP1),Z(1,QP2),IFAULT)
          PROB = PROB - CHIPRB(R*R,Q)
80    CONTINUE
      PROB = PROB/(2.0*Q*Q) + 1.0
C
      RETURN
      END

      SUBROUTINE RNORTM(LDA,N, NP1, IBCONF, NB, B, NDBI, DBI, ISEED, A,
     *   CHISQ, FAB, U, W, IFAULT)
```

```
C
C      ALGORITHM AS 127  APPL. STATIST.  (1978) VOL.27, NO.2
C
C      RNORTM GENERATES ORTHOGONAL MATRICES A FROM A DISTRIBUTION
C      WITH DENSITY FUNCTION DEFINED ON THE GROUP OF ORTHOGONAL
C      MATRICES.  WHEN THE INPUT PARAMETER IBCONF = 1 THE
C      MATRICES ARE GENERATED FROM THE INVARIANT HAAR MEASURE.
C
      REAL B(NB), DBI(NDBI), A(LDA,N), CHISQ(N), U(NP1), W(N), FAB, UL,
     *   WL, WIL, WWIL, T2, HV, HW, ZERO, ONE
      INTEGER N, NP1, IBCONF, NB, NDBI, ISEED, IFAULT, I, J,IBSUB,L,NP1L
      INTEGER LP1,LDA
C
      DATA ZERO /0.0/, ONE /1.0/
C
C      STATEMENT FUNCTIONS FOR DOUBLE PRECISION
C      DOUBLE PRECISION SQRT, SIGN, ABS
C      SQRT(UL) = DSQRT(UL)
C      SIGN(UL, WL) = DSIGN(UL, WL)
C      ABS(UL) = DABS(UL)
C
C      SECTION 1 INITIALIZATION
C
      IFAULT = 0
      GO TO 4
C
C      ERROR RETURNS
C
2     IFAULT = 2
      RETURN
C
C      INITIALIZE DENSITY, H(STORED IN A), AND IBSUB
C
4     FAB = ONE
      DO 100 I=1,N
      DO 50 J=1,N
50    A(I,J) = ZERO
      A(I,I) = ONE
100   CONTINUE
      IBSUB = 0
C
C      CONSTRUCT  A  ONE COLUMN AT A TIME
C
      DO 315 L=1,N
      NP1L = NP1 - L
C
C      SECTION 2
C      CONSTRUCT U(L...N) UNIFORMLY DISTRIBUTED ON THE SURFACE OF THE
C      (N+1-L)-SPHERE OF RADIUS UL = SQRT(CHISQ(L)).
C
      CALL NORMAL(U(L),NP1L+1,ISEED, CHISQ(L))
      UL = SQRT(CHISQ(L))
      IF (UL .EQ. ZERO) GO TO 2
      IF (L .EQ. N) GO TO 301
C
C      SECTION 3
C      CALCULATE  W = B*U  AND STORE ITS LENGTH IN WL.
C      SIGN OF WL PREVENTS LOSS OF SIGNIFICANCE IN WWIL.
C
C      B IS IDENTITY
C
```

```
      WL = SIGN(UL,-U(L))
      DO 219 J=L,N
219   W(J) = U(J)
C
C      SECTION 4
C      PROJECT  W  ONTO ORTHOGOANL COMPLEMENT OF FIRST L-1
C      COLUMNS OF A, NORMALIZE, AND STORE IN L COLUMN OF A.
C      A(L) = H(L) * W * WIL
C      CALCULATE PROJECTION MATRIX H(L) FOR ORTHOGOANL
C      COMPLEMENT OF FIRST L COLUMNS OF A AND STORE IN LAST
C      N-L COLUMNS OF A.
C
      LP1 = L+1
      WIL = ONE/WL
      WWIL = ONE/(WL - W(L))
      DO 250 I=1,N
      T2 = ZERO
      DO 230 J=L,N
230   T2 = T2 + A(I,J) * W(J)
      HV = T2 * WIL
      HW = (HV - A(I,L)) * WWIL
      A(I,L) = HV
      DO 240 J = LP1, N
240   A(I,J) = A(I,J) - HW*W(J)
250   CONTINUE
C
C      END OF L COLUMN OF MATRIX A
C
C      SECTION 5
C      WHEN L = N ONLY SIGN NEEDS TO BE CHOSEN
C      REMEMBER MATRIX H(N) IS STORED IN N COLUMN OF A.
C
301   IF (U(L) .GT. ZERO) GOTO 315
      DO 310 I=1,N
310   A(I,L) = -A(I,L)
C
C      FAB = ABS(B(LAST) * DBI(N)*FAB)
C
315   CONTINUE
320   FAB = ABS(FAB)
C
      RETURN
      END

      SUBROUTINE ZEROVC (F,R,R0,R1,N,V,Y,IFAULT)
C
C  CALCULATES THE SOLUTION R TO F(R*V) = 0.
C
      INTEGER N,IFAULT,I,J,NITER
      REAL F,R,R0,R1,ETA,V(N),Y(N),F0,F1,FR,HALF,ZABS,X
      EXTERNAL F
      DATA HALF, ETA /0.5,1.0E-05/ NITER /500/
      ZABS(X) = ABS(X)
C
      DO 10 I=1,N
         Y(I) = R0*V(I)
10    CONTINUE
      F0 = F(N,Y)
      DO 20 I=1,N
         Y(I) = R1*V(I)
20    CONTINUE
```

```
      F1 = F(N,Y)
      IF (SIGN(HALF,F1) .EQ. SIGN(HALF,F0)) THEN
       IFAULT = 3
       RETURN
      END IF
       DO 40 J = 1,NITER
         R = R1 - F1*(R1-R0)/(F1-F0)
         DO 30 I=1,N
               Y(I) = R*V(I)
30       CONTINUE
         FR = F(N,Y)
         IF (ZABS(R-R1) .LT. ETA) GO TO 50
         IF (SIGN(HALF,FR) .NE. SIGN(HALF,F1)) THEN
            R0=R1
            F0=F1
         ELSE
            F0 = F0 * HALF
         END IF
         R1 = R
         F1 = FR
40     CONTINUE
       IFAULT = 6
50    CONTINUE
      RETURN
      END

      REAL FUNCTION CHIPRB(X,DF)
C
C  FINDS THE PROBABILITY THAT A CHI-SQUARED RANDOM VARIABLE
C  WITH DF DEGREES OF FREEDOM EXCEEDS X.
C  ADAPTED FROM HILL AND PIKE (1967) ALG. 299, CHI-SQUARED INTEGRAL,
C  COMM. ACM, VOL.10, 243-244.
C
      REAL X,ZERFP1,ZSQRT,SQRT2,
     *     A,SQRTA,Y,C,E,Z,S,BIG,SMALL,RTPIRP,ZERO,ONE,HALF,TWO
      INTEGER DF,ODD,N2,I
      REAL ALNORM
C     REAL ERF
C
      DATA RTPIRP /0.564189583547756/, SQRT2 /1.41421356237309/
      DATA BIG/88.03/,SMALL/-85.2/,ZERO/0.0/,ONE /1.0/,HALF/0.5/,
     *     TWO /2.0/
      ZSQRT(A) = SQRT(A)
      ZERFP1(A) = TWO*ALNORM(SQRT2*A,.FALSE.)
C
C  NOTE: IF THE FUNCTION "ERF" IS AVAILABLE, THE ABOVE STATEMENT
C  FUNCTION MAY BE REPLACED BY
C     ZERFP1(A) = ERF(A) + ONE
C
      S = ZERO
      IF (X .GT. BIG) GO TO 50
      S = ONE
      IF (X .LT. SMALL) GO TO 50
C
      A = HALF*X
      Y = EXP(-A)
      ODD = MOD(DF,2)
      IF (ODD .EQ. 0) THEN
         S = Y
         E = ONE
         Z = ZERO
```

```
      ELSE
         SQRTA = ZSQRT(A)
         S = ZERFP1(-SQRTA)
         E = RTPIRP/SQRTA
         Z = -HALF
      END IF
      IF (DF .LT. 3) GO TO 50
C
      N2 = DF/2 - 1 + ODD
      C = ZERO
      DO 40 I = 1,N2
      Z = Z + 1
      E = E*(A/Z)
      C = C + E
40    CONTINUE
      S = S + C*Y
C
50    CONTINUE
      CHIPRB = S
      END

      SUBROUTINE NORMAL(U,NP1,ISEED, CHISQ)
C
C  GENERATES AN (NP1-1)-VECTOR OF INDEPENDENT NORMAL VARIATES IN U.
C  ON OUTPUT, CHISQ CONTAINS THE SQUARED LENGTH OF THE VECTOR.
C
      REAL U(NP1)
      INTEGER ISEED,NP1,I,IFAULT,IX,IY,IZ
      REAL RANDOM,PPND,CHISQ,ZERO
C
      COMMON /RAND/ IX,IY,IZ
      DATA ZERO /0.0/
C
      CHISQ = ZERO
      DO 10 I = 1, NP1-1
      U(I) = PPND(RANDOM(ISEED),IFAULT)
      CHISQ = CHISQ + U(I)*U(I)
10    CONTINUE
      RETURN
      END

      REAL FUNCTION PPND(P, IFAULT)
C
C  ALGORITHM AS 111  APPL. STATIST. (1977), VOL.26, NO.1
C
C  PRODUCES NORMAL DEVIATE CORRESPONDING TO LOWER TAIL AREA OF P
C  REAL VERSION FOR EPS = 2 **(-31)
C  THE HASH SUMS ARE THE SUMS OF THE MODULI OF THE COEFFICIENTS
C  THEY HAVE NO INHERENT MEANINGS BUT ARE INCLUDED FOR USE IN
C  CHECKING TRANSCRIPTIONS
C  STANDARD FUNCTIONS ABS, ALOG AND SQRT ARE USED
C
      REAL ZERO, SPLIT, HALF, ONE
      REAL A0, A1, A2, A3, B1, B2, B3, B4, C0, C1, C2, C3, D1, D2
      REAL P, Q, R
      INTEGER IFAULT
      DATA ZERO /0.0E0/, HALF/0.5E0/, ONE/1.0E0/
      DATA SPLIT /0.42E0/
      DATA A0 / 2.50662823884E0/
      DATA A1 / -18.61500062529E0/
      DATA A2 / 41.39119773534E0/
```

```
      DATA A3 / -25.44106049637E0/
      DATA B1 / -8.47351093090E0/
      DATA B2 / 23.08336743743E0/
      DATA B3 / -21.06224101826E0/
      DATA B4 / 3.13082909833E0/
      DATA C0 / -2.78718931138E0/
      DATA C1 / -2.29796479134E0/
      DATA C2 / 4.85014127135E0/
      DATA C3 / 2.32121276858E0/
      DATA D1 / 3.54388924762E0/
      DATA D2 / 1.63706781897E0/
C
      IFAULT = 0
      Q = P - HALF
      IF (ABS(Q) .GT. SPLIT) GOTO 1
      R = Q*Q
      PPND = Q * (((A3*R + A2)*R + A1) * R + A0) /
     *  ((((B4*R + B3)*R + B2) * R + B1) * R + ONE)
      RETURN
1     R = P
      IF (Q .GT. ZERO)R = ONE - P
      IF (R .LE. ZERO) GOTO 2
      R = SQRT(-ALOG(R))
      PPND = (((C3 * R + C2) * R + C1) * R + C0)/
     *  ((D2*R + D1) * R + ONE)
      IF (Q .LT. ZERO) PPND = -PPND
      RETURN
2     IFAULT = 1
      PPND = ZERO
      RETURN
      END

      REAL FUNCTION RANDOM(L)
C
C  ALGORITHM AS 183  APPL. STATIST. (1982) VOL. 31, NO. 2
C
C  RETURNS A PSEUDO-RANDOM NUMBER RECTANGULARLY DISTRIBUTED
C  BETWEEN 0 AND 1.
C
C  IX, IY, AND IZ SHOULD BE SET TO INTEGER VALUES BETWEEN
C  1 AND 30000 BEFORE FIRST ENTRY
C
C  INTEGER ARITHMETIC UP TO 30323 IS REQUIRED
C
      INTEGER IX, IY, IZ, L
      COMMON /RAND/ IX,IY,IZ
      IX = 171 * MOD(IX, 177) - 2 * (IX / 177)
      IY = 172 * MOD(IY, 176) - 35 * (IY / 176)
      IZ = 170 * MOD(IZ,178) - 63 * (IZ / 178)
C
      IF (IX .LT. 0) IX = IX + 30269
      IF (IY .LT. 0) IY = IY + 30307
      IF (IZ .LT. 0) IZ = IZ + 30323
C
      RANDOM = MOD(FLOAT(IX) / 30269.0 + FLOAT(IY) / 30307.0 +
     *             FLOAT(IZ) / 30323.0, 1.0)
      RETURN
      END

      REAL FUNCTION ALNORM(X,UPPER)
C
```

```
C  ALGORITHM AS 66  APPL. STATIST. (1973) VOL. 22, NO. 3
C
C  EVALUATES THE TAIL AREA OF THE STANDARDIZED NORMAL CURVE
C  FROM X TO INFINITY IF UPPER IS .TRUE. OR
C  FROM MINUS INFINITY TO X IF UPPER IS .FALSE.
C
      REAL LTONE, UTZERO, ZERO, HALF, ONE, CON, Z, Y, X
      LOGICAL UPPER, UP
C
C  LTONE AND UTZERO MUST BE SET TO SUIT THE PARTICULAR COMPUTER
C  (SEE INTRODUCTORY TEXT)
C
      DATA LTONE, UTZERO /7.0, 18.66/
      DATA ZERO, HALF, ONE, CON /0.0,0.5,1.0,1.28/
      UP = UPPER
      Z = X
      IF (Z .GE. ZERO) GOTO 10
      UP = .NOT. UP
      Z = -Z
10    IF (Z .LE. LTONE .OR. UP .AND. Z .LE. UTZERO) GOTO 20
      ALNORM = ZERO
      GOTO 40
20    Y = HALF *Z * Z
      IF (Z .GT. CON) GOTO 30
C
      ALNORM = HALF - Z*(0.398942280444 - 0.399903438504 * Y /
     1  (Y + 5.75885480458 - 29.8213557808 /
     2  (Y + 2.62433121679 + 48.6959930692 /
     3  (Y + 5.92885724438))))
      GO TO 40
C
30    ALNORM = 0.398942280385 * EXP(-Y) /
     1  (Z - 3.8052E-08 + 1.00000615302 /
     2  (Z + 3.98064794E-4 + 1.98615381364 /
     3  (Z - 0.151679116635 + 5.29330324926 /
     4  (Z + 4.8385912808 - 15.1508972451 /
     5  (Z + 0.742380924027 + 30.789933034 / (Z + 3.99019417011))))))
C
40    IF (.NOT. UP) ALNORM = ONE - ALNORM
      RETURN
      END

C-----------------------------------------------------------------------
```

5. Various user functions for different shapes of region of integration.

```
      REAL FUNCTION F(N,V)
      REAL V(N),CUBSZ
      INTEGER N,I
      DATA CUBSZ /3.00/
C
C  BOUNDARY OF REGION IS AN N-DIMENSIONAL CUBE.
C  CUBSZ IS THE DISTANCE FROM THE ORIGIN TO THE SIDE OF THE CUBE
C
      F = 0.0
      DO 20 I = 1,N
      F = AMAX1(F,ABS(V(I)))
20    CONTINUE
      F = F - CUBSZ
      RETURN
      END
```

```
      REAL FUNCTION F(N,V)
      REAL V(N)
      INTEGER N
      REAL A(200),B(10),C
      INTEGER I,J,K
C
C  THE BOUNDARY IS AN ELLIPSOID OF THE FORM X'AX + BX + C = 0
C  THE UPPER TRIANGLE OF A IS STORED COLUMN-WISE IN PACKED FORM.
C  IN THIS EXAMPLE, A = INVERSE (1  .3), B = 0, C = -1.
C                               (.3  1)
C
      DATA A(1),A(2),A(3) /1.0989011,-.32967033,1.0989011/
     *    B(1),B(2) /0.0,0.0/ C /-1.0/
C
      F = C
      K = 0
      DO 20 I = 1,N
      K = K + 1
      F = F + (V(I)**2)*A(K)
      DO 10 J = I+1,N
      K = K + 1
      F = F + 2.0*V(I)*V(J)*A(K)
10    CONTINUE
20    CONTINUE
      DO 30 I = 1,N
      F = F + B(I)*V(I)
30    CONTINUE
      RETURN
      END


      REAL FUNCTION F(N,V)
      REAL V(N)
      INTEGER N,I
      REAL TEMP,A(30),B(30)
C
C  RECTANGLE IS GIVEN BY A < V < B FOR EACH COORDINATE.
C  THIS SUBROUTINE MAY ALSO BE USED TO CALCULATE THE
C  CDF BY MAKING THE APPROPRIATE SIDES OF THE RECTANGLE
C  SUFFICIENTLY LARGE.
C  THIS EXAMPLE IS THE FOUR-DIMENSIONAL RECTANGLE DEFINED
C  BY  -INFINITY (ALMOST) < V(I) < 2 FOR EACH I.
C
      DATA A(1),A(2),A(3),A(4)
     *  /-20.0,-20.0,-20.0,-20.0/,
     *       B(1),B(2),B(3),B(4)
     *  /2.0,2.0,2.0,2.0/
C
      F = -1.0
      DO 10 I=1,N
         IF (V(I) .GT. 0.0) THEN
            TEMP = V(I) - B(I)
         ELSE
            TEMP = A(I) - V(I)
         END IF
         F  = AMAX1(F,TEMP)
10    CONTINUE
      RETURN
      END
```

```
      REAL FUNCTION F(N,V)
      REAL V(N),RADSQ
      INTEGER N,I
      DATA RADSQ /4.0/
C
C  BOUNDARY IS SPHERE OF RADIUS SQRT(RADSQ).
C
      F = 0.0
      DO 20 I = 1,N
      F = F + V(I)*V(I)
20    CONTINUE
      F = F - RADSQ
      RETURN
      END


      REAL FUNCTION F(N,V)
      REAL V(N)
      INTEGER N
      INTEGER I,J,P
      REAL TEMP, A(2,6)
C
C  A CONVEX POLYTOPE WITH P FACES IS DEFINED BY THE INEQUALITIES
C  A(1,I)*V(1) + A(2,I)*V(2) +...+ A(N,I)*V(N) <= 1
C  FOR I=1,...,P.
C
C  THE FIGURE IN THIS DEMONSTRATION PROGRAM IS THE
C  REGULAR PENTAGON WITH DISTANCE FROM CENTER TO POINT = 1.
C
      DATA P/5/
     *       A(1,1),A(2,1) /0.726542528, -1.0/
     *       A(1,2),A(2,2) /1.175570504, 0.381966011/
     *       A(1,3),A(2,3) /0.0, 1.236067978/
     *       A(1,4),A(2,4) /-0.726542528, -1.0/
     *       A(1,5),A(2,5) /-1.175570504, 0.381966011/
C
      F = -1.0
      DO 10 J=1,N
         F = F + A(J,1)*V(J)
10    CONTINUE
      DO 30 I=2,P
        TEMP = -1.0
        DO 20 J=1,N
           TEMP = TEMP + A(J,I)*V(J)
20      CONTINUE
        F = AMAX1(F,TEMP)
30    CONTINUE
      RETURN
      END


      REAL FUNCTION F(N,V)
      REAL V(N)
      INTEGER I,J,P,INDEX,N
      REAL A(2,6),TANG(5),X(2),TANGNT
C
C  THE FIGURE IN THIS DEMONSTRATION PROGRAM IS THE
C  REGULAR PENTAGRAM WITH DISTANCE FROM CENTER TO POINT = 2.618.
C
```

```
      DATA P/5/
     *       A(1,3),A(2,3) /0.726542528, -1.0/
     *       A(1,2),A(2,2) /1.175570504, 0.381966011/
     *       A(1,4),A(2,4) /0.0, 1.236067978/
     *       A(1,1),A(2,1) /-0.726542528, -1.0/
     *       A(1,5),A(2,5) /1.175570504, 0.381966011/
     *           TANG(1) / -1.376381921/
     *           TANG(2) / -0.3249196962329/
     *           TANG(3) / 0.3249196962329/
     *           TANG(4) / 1.376381921/
C
      X(1) = ABS(V(1))
      X(2) = V(2)
      INDEX = P
      IF (X(1) .EQ. 0.0) THEN
         INDEX = 3
      ELSE
         TANGNT = X(2)/X(1)
         DO 10 I=1,P
            IF (TANGNT .LE. TANG(I)) THEN
            INDEX = I
            GO TO 20
            END IF
10       CONTINUE
20       CONTINUE
      END IF
      F = -1.0
      DO 30 J=1,2
         F = F + A(J,INDEX)*X(J)
30    CONTINUE
      RETURN
      END

C-------------------------------------------------------------------------------
```

6. The data input file FORT.1 needed by the driver program.

```
7   .0005   107.0   100   81501   19837   39470
  1
  0
  1
  0
  0
  1
  0
  0
  0
  1
  0
  0
  0
  0
  1
  0
  0
  0
  0
  0
  1
  0
```

0
0
0
0
0
1
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
1
0
0
0
0

0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
```

```
C                   EVM PROGRAMS WITH EXAMPLES
C
C
C CONTENTS
C
C This file gives the full computer code, in FORTRAN 77, for the algorithm
C described in "Parameter Estimation in the Error-in- Variables Model", by
C Park M. Reilly, S.E. Keeler, and H.V. Reilly.
C
C It is given in two forms, first for the case where there is any number of
C functional equations in the model, and then for the special case where there
C is only one.  The special case covers a large fraction of practical
C applications and the applicable form of the algorithm is not given in detail
C in the paper.  The special form executes substantially more quickly than the
C more general one.  Also given are the complete input and output files for an
C example using each of the two forms of the algorithm.  The material is
C organised as follows:
C
C           1. EVM.FOR     The algorithm in general form
C                          for any number of functional equations.
C           2. EVMS.FOR    The special form of the algorithm
C                          for only one functional equation.
C           3. EX1.FOR     Example 1. The complete Fortran
C                          code for the example in the paper.
C                          It has three functional equations.
C           4. EX1.DAT     The data required by Example 1 in the
C                          required format.
C           5. EX1.OUT     Output from Example 1.  Shown as
C                          Table 1 in the paper.
C           6. EX2.FOR     The complete program for Example 2
C                          which has only one functional equation.
C           7. EX2.DAT     The data for Example 2 in the required
C                          format.
C           8. EX2.OUT     Output from Example 2.
```

```
C
C Park Reilly
C
C
C 1. EVM.FOR, THE GENERAL ALGORITHM
C

      SUBROUTINE EVM(B,BV,DATA,EST,F,G1,G2,G3,H,IFAULT,
     *  NDAT,NEQ,NPAR,NVAR,PHI,Q1,Q2,RESID,S,T,THETA,V,XI,Z)
C
C  PARAMETER ESTIMATION FOR THE ERROR-IN-VARIABLES MODEL
C
      REAL B(NEQ,NVAR), BV(NEQ,NVAR), CRIT1, DATA(NDAT,NVAR),
     * EST(NDAT,NVAR), F(NEQ), G1(NPAR,NPAR), G2(NPAR,NPAR),
     * G3(NPAR,NPAR), H(NEQ), ONE, PHI, Q1(NPAR), Q2(NPAR),
     * RESID(NDAT,NVAR), S(NEQ,NEQ), T(NEQ), THETA(NPAR),
     * V(NVAR,NVAR), WDIFF, W1, W2, XI(NVAR), Z(NEQ,NPAR), ZERO
      DIMENSION IFAULT(2)
      PARAMETER (CRIT1=1.E-6, NITER1=50, ZERO=0.E0, ONE=1.E0)
      IFAULT(1) = 0
      IFAULT(2) = 0
      DO 4 ITER = 1,NITER1
C
C  EACH PASS THROUGH THIS LOOP PERFORMS A SINGLE OUTER ITERATION
C
        CALL INNER(B,BV,DATA,EST,F,G1,H,IFAULT,NDAT,
     *  NEQ,NPAR,NVAR,PHI,Q1,RESID,S,T,THETA,V,XI,Z)
C
C  PUT Q1 INTO Q2 AND G1 INTO G2
C
        DO 2 IPAR1 = 1,NPAR
          Q2(IPAR1) = Q1(IPAR1)
          DO 1 IPAR2 = IPAR1,NPAR
1         G2(IPAR1,IPAR2) = G1(IPAR1,IPAR2)
2       CONTINUE
C
C  CALCULATE CORRECTION FOR THETA
C
        CALL CHOL(G2,Q2,NPAR)
        CALL BKSUB(G2,Q2,1,NPAR,1)
C
C CHECK FOR CONVERGENCE
C
        WDIFF = ZERO
        DO 3 IPAR = 1,NPAR
          W1 = Q2(IPAR)
          W2 = THETA(IPAR)
          WDIFF = MAX(WDIFF,ABS(W1/W2))
          THETA(IPAR) = W2-W1
3       CONTINUE
        IF(WDIFF.LE.CRIT1)GO TO 5
4     CONTINUE
      IFAULT(1) = 1
C
C  FILL LOWER TRIANGLE OF G1 AND PREPARE G2
C  FOR THE INVERSION OF G1
C
5     DO 7 IPAR1 = 1,NPAR
        DO 6 IPAR2 = IPAR1,NPAR
          W1 = G1(IPAR1,IPAR2)
          G1(IPAR2,IPAR1) = W1
```

```
            G2(IPAR1,IPAR2) = W1
            G3(IPAR1,IPAR2) = ZERO
6         CONTINUE
7       CONTINUE
C
C  INVERT G1 INTO G3
C
        CALL CHOL(G2,Q2,NPAR)
        DO 8 IPAR1 = 1,NPAR
8       G3(IPAR1,IPAR1) = ONE/G2(IPAR1,IPAR1)
        CALL BKSUB(G2,G3,NPAR,NPAR,3)
        RETURN
        END


        SUBROUTINE INNER(B,BV,DATA,EST,F,G1,H,IFAULT,NDAT,NEQ,
       *    NPAR,NVAR,PHI,Q1,RESID,S,T,THETA,V,XI,Z)
        REAL B(NEQ,NVAR), BV(NEQ,NVAR), CRIT2, DATA(NDAT,NVAR),
       * EST(NDAT,NVAR), F(NEQ), G1(NPAR,NPAR), H(NEQ), HALF,
       * PHI, Q1(NPAR), RESID(NDAT,NVAR), S(NEQ,NEQ),
       * T(NEQ), THETA(NPAR),
       * V(NVAR,NVAR), WDIFF, W1, XI(NVAR), Z(NEQ,NPAR), ZERO
        DIMENSION IFAULT(2)
        PARAMETER (CRIT2=1.E-6, HALF=.5E0, NITER2=20, ZERO=0.E0)
C
C  INITIALIZE PHI, Q1, AND G1
C
        PHI = ZERO
        DO 2 IPAR1 = 1,NPAR
          Q1(IPAR1) = ZERO
          DO 1 IPAR2 = IPAR1,NPAR
1         G1(IPAR1,IPAR2) = ZERO
2       CONTINUE
C
C  PERFORM INNER ITERATION TO CONVERGENCE FOR EACH IDAT
C
        DO 24 IDAT = 1,NDAT
          DO 3 IVAR1 = 1,NVAR
3         XI(IVAR1) = DATA(IDAT,IVAR1)
          DO 16 ITER = 1,NITER2
C
C  THIS LOOP PERFORMS A SINGLE INNER ITERATION
C
            WDIFF = ZERO
            CALL BF(B,F,THETA,XI)
C
C  CALCULATE BV
C
            DO 6 IEQ1 = 1,NEQ
              DO 5 IVAR1 = 1,NVAR
                W1 = ZERO
                DO 4 IVAR2 = 1,NVAR
4               W1 = W1 + B(IEQ1,IVAR2)*V(IVAR1,IVAR2)
                BV(IEQ1,IVAR1) = W1
5             CONTINUE
6           CONTINUE
C
C  CALCULATE BVB' AND STORE IT IN S
C
            DO 9 IEQ1 = 1,NEQ
              DO 8 IEQ2 = IEQ1,NEQ
```

```
                W1 = ZERO
                DO 7 IVAR1 = 1,NVAR
7               W1 = W1 + BV(IEQ1,IVAR1)*B(IEQ2,IVAR1)
                S(IEQ1,IEQ2) = W1
8             CONTINUE
9           CONTINUE
C
C  PUT F+B(X-XI) INTO H
C
            DO 11 IEQ1 = 1,NEQ
              W1 = F(IEQ1)
              DO 10 IVAR1 = 1,NVAR
10            W1 = W1 + B(IEQ1,IVAR1)*(DATA(IDAT,IVAR1)-XI(IVAR1))
              H(IEQ1) = W1
11          CONTINUE
C
C  REPLACE S BY ITS TRIANGULAR FACTORIZATION
C  AND SOLVE ST = H FOR T
C
            CALL CHOL(S,H,NEQ)
            DO 13 IEQ1 = 1,NEQ
              T(IEQ1) = H(IEQ1)
              DO 12 IEQ2 = IEQ1,NEQ
12            S(IEQ2,IEQ1) = S(IEQ1,IEQ2)
13          CONTINUE
            CALL BKSUB(S,T,1,NEQ,1)
C
C  CALCULATE NEW XI AND PUT RELATIVE CHANGE IN XI INTO WDIFF
C
            DO 15 IVAR1 = 1,NVAR
              W1 = DATA(IDAT,IVAR1)
              DO 14 IEQ1 = 1,NEQ
14            W1 = W1 - BV(IEQ1,IVAR1)*T(IEQ1)
              WDIFF = MAX(WDIFF, ABS((W1-XI(IVAR1))/W1))
              XI(IVAR1) = W1
15          CONTINUE
C
C  CHECK FOR CONVERGENCE
C
            IF(WDIFF.LE.CRIT2)GO TO 17
16        CONTINUE
          IFAULT(2) = 1
C
C  ACCUMULATE PHI
C
17        DO 18 IEQ1 = 1,NEQ
18        PHI = PHI + H(IEQ1)**2
C
C  CALCULATE FITTED VALUES AND RESIDUALS
C
          DO 19 IVAR1 = 1,NVAR
            W1 = XI(IVAR1)
            EST(IDAT,IVAR1) = W1
            RESID(IDAT,IVAR1) = DATA(IDAT,IVAR1)-W1
19        CONTINUE
C
C  CALCULATE CONTRIBUTIONS TO Q1 AND G1 AND COMPLETE THE LOOPS
C
          CALL ZED(B,F,THETA,XI,Z)
          CALL BKSUB(S,Z,NPAR,NEQ,2)
          DO 23 IPAR1 = 1,NPAR
```

```
          W1 = Q1(IPAR1)
          DO 20 IEQ1 = 1,NEQ
20        W1 = W1 + Z(IEQ1,IPAR1)*H(IEQ1)
          Q1(IPAR1) = W1
          DO 22 IPAR2 = IPAR1,NPAR
            W1 = G1(IPAR1,IPAR2)
            DO 21 IEQ1 = 1,NEQ
21          W1 = W1 + Z(IEQ1,IPAR1)*Z(IEQ1,IPAR2)
            G1(IPAR1,IPAR2) = W1
22        CONTINUE
23      CONTINUE
24    CONTINUE
      PHI = PHI*HALF
      RETURN
      END


      SUBROUTINE CHOL(A,B,N)
C
C  PERFORM CHOLESKY DECOMPOSITION OF A AND PRELIMINARY TREATMENT OF B
C
      REAL A(N,N),B(N),W1,W2
      DO 5 IR1 = 1,N
        W1 = A(IR1,IR1)
        DO 1 IR2 = 1,IR1-1
1       W1 = W1 - A(IR2,IR1)**2
        W1 = SQRT(W1)
        A(IR1,IR1) = W1
        DO 3 IC1 = IR1+1,N
          W2 = A(IR1,IC1)
          DO 2 IR2 = 1,IR1-1
2         W2 = W2 - A(IR2,IR1)*A(IR2,IC1)
          A(IR1,IC1) = W2/W1
3       CONTINUE
        W2 = B(IR1)
        DO 4 IR2 = 1,IR1-1
4       W2 = W2 - A(IR2,IR1)*B(IR2)
        B(IR1) = W2/W1
5     CONTINUE
      RETURN
      END


      SUBROUTINE BKSUB(A,B,M,N,JOB)
C
C  PERFORM BACK-SOLUTION ACCORDING AS JOB = 1, 2, OR 3
C
      REAL A(N,N),B(N,M),W1
      DO 3 IC1 = M,1,-1
        IF(JOB.EQ.2) THEN
          K1 = 1
          K2 = N
          K3 = 1
        ELSE
          K2 = 1
          K3 = -1
          IF(JOB.EQ.1) THEN
            K1 = N
          ELSE
            K1 = IC1
          END IF
```

```
          END IF
          DO 2 IR1 = K1,K2,K3
            W1 = B(IR1,IC1)
            IF (JOB.EQ.2) THEN
              K2 = IR1-1
            ELSE
              K1 = N
              K2 = IR1+1
            END IF
            DO 1 IR2 = K1,K2,K3
1           W1 = W1 - A(IR1,IR2)*B(IR2,IC1)
            B(IR1,IC1) = W1/A(IR1,IR1)
            IF(JOB.EQ.3) B(IC1,IR1) = B(IR1,IC1)
2         CONTINUE
3       CONTINUE
        RETURN
        END

C-----------------------------------------------------------------------
C
C 2. EVMS.FOR, THE SPECIAL FORM FOR ONLY ONE FUNCTIONAL EQUATION

        SUBROUTINE EVMS(BS,BVS,DATA,EST,G1,G2,G3,IFAULT,
       *    NDAT,NPAR,NVAR,PHI,Q1,Q2,RESID,THETA,V,XI,ZS)
C
C  PARAMETER ESTIMATION FOR THE ERROR-IN-VARIABLES MODEL WITH M = 1
C  91/3/21 PMR
C
        REAL BS(NVAR), BVS(NVAR), BVB, CRIT1, DATA(NDAT,NVAR),
       * EST(NDAT,NVAR), FS, G1(NPAR,NPAR), G2(NPAR,NPAR),
       * G3(NPAR,NPAR), HS, ONE, PHI, Q1(NPAR), Q2(NPAR),
       * RESID(NDAT,NVAR), THETA(NPAR),
       * V(NVAR,NVAR), WDIFF, W1, W2, XI(NVAR), ZS(NPAR), ZERO
        DIMENSION IFAULT(2)
        PARAMETER (CRIT1=1.E-6, NITER1=50, ONE=1.E0, ZERO=0.E0)
C
        IFAULT(1) = 0
        IFAULT(2) = 0
        DO 4 ITER = 1,NITER1
C
C  EACH PASS THROUGH THIS LOOP PERFORMS A SINGLE OUTER ITERATION
C
          CALL INNERS(BS,BVS,DATA,EST,G1,IFAULT,NDAT,
       *      NPAR,NVAR,PHI,Q1,RESID,THETA,V,XI,ZS)
C
C  PUT Q1 INTO Q2 AND G1 INTO G2
C
          DO 2 IPAR1 = 1,NPAR
            Q2(IPAR1) = Q1(IPAR1)
            DO 1 IPAR2 = IPAR1,NPAR
1           G2(IPAR1,IPAR2) = G1(IPAR1,IPAR2)
2         CONTINUE
C
C  CALCULATE CORRECTION FOR THETA
C
          CALL CHOL(G2,Q2,NPAR)
          CALL BKSUB(G2,Q2,1,NPAR,1)
C
C CHECK FOR CONVERGENCE
C
          WDIFF = ZERO
```

```
          DO 3 IPAR = 1,NPAR
            W1 = Q2(IPAR)
            W2 = THETA(IPAR)
            WDIFF = MAX(WDIFF,ABS(W1/W2))
            THETA(IPAR) = W2-W1
3         CONTINUE
          IF(WDIFF.LE.CRIT1) GO TO 5
4       CONTINUE
        IFAULT(1) = 1
C
C  FILL LOWER TRIANGLE OF G1 AND PREPARE G2 FOR THE INVERSION OF G1
C
5       DO 7 IPAR1 = 1,NPAR
          DO 6 IPAR2 = IPAR1,NPAR
            W1 = G1(IPAR1,IPAR2)
            G1(IPAR2,IPAR1) = W1
            G2(IPAR1,IPAR2) = W1
            G3(IPAR1,IPAR2) = ZERO
6         CONTINUE
7       CONTINUE
C
C  INVERT G1 INTO G3
C
        CALL CHOL(G2,Q2,NPAR)
        DO 8 IPAR1 = 1,NPAR
8       G3(IPAR1,IPAR1) = ONE/G2(IPAR1,IPAR1)
        CALL BKSUB(G2,G3,NPAR,NPAR,3)
        RETURN
        END


        SUBROUTINE INNERS(BS,BVS,DATA,EST,G1,IFAULT,NDAT,
       *    NPAR,NVAR,PHI,Q1,RESID,THETA,V,XI,ZS)
C
C  INNER ITERATION FOR M = 1
C
        REAL BS(NVAR),BVS(NVAR),BVB,CRIT2,DATA(NDAT,NVAR),
       * EST(NDAT,NVAR),FS,HALF,G1(NPAR,NPAR),HS,
       * PHI,Q1(NPAR),RESID(NDAT,NVAR),THETA(NPAR),
       * V(NVAR,NVAR),WDIFF,W1,XI(NVAR),ZS(NPAR),ZERO
        DIMENSION IFAULT(2)
        PARAMETER (CRIT2=1.E-5, HALF=.5E0, NITER2=15, ZERO=0.E0)
C
C  INITIALIZE PHI, Q1, AND G1
C
        PHI = ZERO
        DO 2 IPAR1 = 1,NPAR
          Q1(IPAR1) = ZERO
          DO 1 IPAR2 = IPAR1,NPAR
1         G1(IPAR1,IPAR2) = ZERO
2       CONTINUE
C
C  PERFORM INNER ITERATION TO CONVERGENCE FOR EACH IDAT
C
        DO 14 IDAT = 1,NDAT
C
C  INITIALIZE XI
C
          DO 3 IVAR1 = 1,NVAR
3         XI(IVAR1) = DATA(IDAT,IVAR1)
          DO 9 ITER = 1,NITER2
```

```
C
C  THIS LOOP PERFORMS A SINGLE INNER ITERATION
C
          WDIFF = ZERO
          CALL BF(BS,FS,THETA,XI)
C
C  CALCULATE BVS
C
          DO 5 IVAR1 = 1,NVAR
            W1 = ZERO
            DO 4 IVAR2 = 1,NVAR
4           W1 = W1 + BS(IVAR2)*V(IVAR1,IVAR2)

            BVS(IVAR1) = W1
5         CONTINUE
C
C  CALCULATE BVB
C
          BVB = ZERO
          DO 6 IVAR1 = 1,NVAR
6         BVB = BVB + BVS(IVAR1)*BS(IVAR1)
C
C  PUT F+B(X-XI) INTO HS
C
          HS = FS
          DO 7 IVAR1 = 1,NVAR
7         HS = HS + BS(IVAR1)*(DATA(IDAT,IVAR1)-XI(IVAR1))
C
C  CALCULATE NEW XI AND PUT RELATIVE CHANGE IN XI INTO WDIFF
C
          DO 8 IVAR1 = 1,NVAR
            W1 = DATA(IDAT,IVAR1) - BVS(IVAR1)*HS/BVB
            WDIFF = MAX(WDIFF, ABS((W1-XI(IVAR1))/W1))
            XI(IVAR1) = W1
8         CONTINUE
C
C  CHECK FOR CONVERGENCE
C
          IF(WDIFF.LE.CRIT2) GO TO 10
9       CONTINUE
        IFAULT(2) = 1
C
C  ACCUMULATE PHI
C
10      PHI = PHI+HS*HS/BVB
C
C  CALCULATE FITTED VALUES AND RESIDUALS
C
        DO 11 IVAR1 = 1,NVAR
          W1 = XI(IVAR1)
          EST(IDAT,IVAR1) = W1
          RESID(IDAT,IVAR1) = DATA(IDAT,IVAR1) - W1
11      CONTINUE
C
C  CALCULATE CONTRIBUTIONS TO Q1 AND G1 AND COMPLETE THE LOOPS
C
        CALL ZED(BS,FS,THETA,XI,ZS)
        DO 13 IPAR1 = 1,NPAR
          W1 = ZS(IPAR1)
          Q1(IPAR1) = Q1(IPAR1) + W1*HS/BVB
          DO 12 IPAR2 = IPAR1,NPAR
```

```
12        G1(IPAR1,IPAR2) = G1(IPAR1,IPAR2) + W1*ZS(IPAR2)/BVB
13      CONTINUE
14    CONTINUE
      PHI = PHI*HALF
      RETURN
      END


      SUBROUTINE CHOL(A,B,N)
C
C  PERFORM CHOLESKY DECOMPOSITION OF A AND PRELIMINARY TREATMENT
C     OF B
C
      REAL A(N,N),B(N),W1,W2
      DO 5 IR1 = 1,N
        W1 = A(IR1,IR1)
        DO 1 IR2 = 1,IR1-1
1       W1 = W1 - A(IR2,IR1)**2
        W1 = SQRT(W1)
        A(IR1,IR1) = W1
        DO 3 IC1 = IR1+1,N
          W2 = A(IR1,IC1)
          DO 2 IR2 = 1,IR1-1
2         W2 = W2 - A(IR2,IR1)*A(IR2,IC1)
          A(IR1,IC1) = W2/W1
3       CONTINUE
        W2 = B(IR1)
        DO 4 IR2 = 1,IR1-1
4       W2 = W2 - A(IR2,IR1)*B(IR2)
        B(IR1) = W2/W1
5     CONTINUE
      RETURN
      END


      SUBROUTINE BKSUB(A,B,M,N,JOB)
C
C  PERFORM BACK-SOLUTION ACCORDING AS JOB = 1, 2, OR 3
C
      REAL A(N,N),B(N,M),W1
      DO 3 IC1 = M,1,-1
        IF(JOB.EQ.2) THEN
          K1 = 1
          K2 = N
          K3 = 1
        ELSE
          K2 = 1
          K3 = -1
          IF(JOB.EQ.1) THEN
            K1 = N
          ELSE
            K1 = IC1
          END IF
        END IF
        DO 2 IR1 = K1,K2,K3
          W1 = B(IR1,IC1)
          IF (JOB.EQ.2) THEN
            K2 = IR1-1
          ELSE
            K1 = N
            K2 = IR1+1
```

```
           END IF
           DO 1 IR2 = K1,K2,K3
     1     W1 = W1 - A(IR1,IR2)*B(IR2,IC1)
           B(IR1,IC1) = W1/A(IR1,IR1)
           IF(JOB.EQ.3) B(IC1,IR1) = B(IR1,IC1)
     2    CONTINUE
     3   CONTINUE
         RETURN
         END

C-----------------------------------------------------------------------
C
C 3. EX1.FOR, EXAMPLE 1
C
C This example is fully described in the paper. The description is
C repeated here.  The problem is a small artificial one (called the
C New Brunswick problem) in which it is wanted to obtain estimates
C for the elements of the parameter vector ~ in the following three
C functional equations which apply for all i=1,2,...,6
C
C             (see published paper)
C
C The parameters which describe the magnitude of the problem are:
C     the number of data vectors, NDAT=6
C     the number of equations, NEQ=3
C     the number of model parameters to be estimated, NPAR=4
C     the number of measured variables in a data vector, NVAR=5.
C The program parameters CRIT1 and CRIT2 were both set at 1.E-6 while NITER1
C and NITER2 were 50 and 20.  The full Fortran code as used is listed in
C EX1.FOR.  The data in the form required for input to the program are given
C in the file EX1.DAT.  The same data are listed with descriptive headings at
C the beginning of the file EX1.OUT.  The latter part of EX1.OUT shows the
C output from the program.
C
C EX1.FOR

C  TEST PROGRAM FOR EVM USING NEW BRUNSWICK PROBLEM   91/3/27 PMR

       DIMENSION B(3,5), BV(3,5), DATA(6,5), EST(6,5), F(3), G1(4,4),
      * G2(4,4), G3(4,4), H(3), IFAULT(2), Q1(4), Q2(4), RESID(6,5),
      * S(3,3), T(3), THETA(4), V(5,5), XI(5), Z(3,4)
C
       READ(20,*) THETA
       READ(20,*) ((DATA(I,J),J=1,5),I=1,6)
       READ(20,*) V
       WRITE(30,1)
     1 FORMAT(36X,'TABLE 1')
       WRITE(30,2)
     2 FORMAT(//'DATA')
       WRITE(30,3) ((DATA(I,J),J=1,5),I=1,6)
     3 FORMAT(5F9.2)
       WRITE(30,4)
     4 FORMAT(//'V')
       WRITE(30,3) V
       WRITE(30,5) THETA
     5 FORMAT(//'INITIAL PARAMETER ESTIMATES = ',4F9.2)
       CALL EVM(B,BV,DATA,EST,F,G1,G2,G3,H,IFAULT,
      *    6,3,4,5,PHI,Q1,Q2,RESID,S,T,THETA,V,XI,Z)
       WRITE(30,6) THETA
     6 FORMAT(//'FINAL PARAMETER ESTIMATES = ',4F9.6)
       WRITE(30,7) PHI
```

```
7     FORMAT(//'FINAL PHI = ',E13.7)
      WRITE(30,8) Q1
8     FORMAT(//'FINAL Q = ',4E16.7)
      WRITE(30,9)
9     FORMAT(//'ESTIMATED TRUE VALUES OF MEASUREMENTS')
      WRITE(30,10) ((EST(I,J),J=1,5),I=1,6)
10    FORMAT(5F9.4)
      WRITE(30,11)
11    FORMAT(//'RESIDUALS')
      WRITE(30,10) ((RESID(I,J),J=1,5),I=1,6)
      WRITE(30,12)
12    FORMAT(//'EXPECTED INFORMATION MATRIX')
      WRITE(30,13) G1
13    FORMAT(4F9.4)
      WRITE(30,14)
14    FORMAT(//'INVERSE OF EXPECTED INFORMATION MATRIX')
      WRITE(30,13) G3
      WRITE(30,15)
15    FORMAT(//'FAULT STATUS')
      WRITE(30,16) IFAULT
16    FORMAT(2I5)
      STOP
      END


      SUBROUTINE BF(B,F,THETA,XI)
      DIMENSION B(3,5), F(3), THETA(4), XI(5)
C
      F(1) = THETA(1)**2 + THETA(2)*XI(1) + THETA(3)*XI(2)**2 - XI(3)
      F(2) = THETA(2)**2 + (THETA(3)*XI(1))**2
     *       + THETA(4)**3*XI(3)**2 - XI(4)
      F(3) = THETA(3)**2 + (THETA(1)*XI(2))**2 - XI(5)
      B(1,1) = THETA(2)
      B(1,2) = 2.E0*THETA(3)*XI(2)
      B(1,3) = -1.E0
      B(1,4) = 0.E0
      B(1,5) = 0.E0
      B(2,1) = 2.E0*THETA(3)**2*XI(1)
      B(2,2) = 0.E0
      B(2,3) = 2.E0*THETA(4)**3*XI(3)
      B(2,4) = -1.E0
      B(2,5) = 0.E0
      B(3,1) = 0.E0
      B(3,2) = 2.E0*THETA(1)**2*XI(2)
      B(3,3) = 0.E0
      B(3,4) = 0.E0
      B(3,5) = -1.E0
      RETURN
      END


      SUBROUTINE ZED(B,F,THETA,XI,Z)
      DIMENSION B(3,5),F(3),THETA(4),XI(5),Z(3,4)
      Z(1,1) = 2.E0*THETA(1)
      Z(1,2) = XI(1)
      Z(1,3) = XI(2)**2
      Z(1,4) = 0.E0
      Z(2,1) = 0.E0
      Z(2,2) = 2.E0*THETA(2)
      Z(2,3) = 2.E0*THETA(3)*XI(1)**2
      Z(2,4) = 3.E0*(THETA(4)*XI(3))**2
```

```
      Z(3,1) = 2.E0*THETA(1)*XI(2)**2
      Z(3,2) = 0.E0
      Z(3,3) = 2.E0*THETA(3)
      Z(3,4) = 0.E0
      RETURN
      END


      SUBROUTINE EVM(B,BV,DATA,EST,F,G1,G2,G3,H,IFAULT,
     *    NDAT,NEQ,NPAR,NVAR,PHI,Q1,Q2,RESID,S,T,THETA,V,XI,Z)
C
C  PARAMETER ESTIMATION FOR THE ERROR-IN-VARIABLES MODEL
C
      REAL B(NEQ,NVAR), BV(NEQ,NVAR), CRIT1, DATA(NDAT,NVAR),
     * EST(NDAT,NVAR), F(NEQ), G1(NPAR,NPAR), G2(NPAR,NPAR),
     * G3(NPAR,NPAR), H(NEQ), ONE, PHI, Q1(NPAR), Q2(NPAR),
     * RESID(NDAT,NVAR), S(NEQ,NEQ), T(NEQ), THETA(NPAR),
     * V(NVAR,NVAR), WDIFF, W1, W2, XI(NVAR), Z(NEQ,NPAR), ZERO
      DIMENSION IFAULT(2)
      PARAMETER (CRIT1=1.E-6, NITER1=50, ZERO=0.E0, ONE=1.E0)
      IFAULT(1) = 0
      IFAULT(2) = 0
      DO 4 ITER = 1,NITER1
C
C  EACH PASS THROUGH THIS LOOP PERFORMS A SINGLE OUTER ITERATION
C
        CALL INNER(B,BV,DATA,EST,F,G1,H,IFAULT,NDAT,
     *      NEQ,NPAR,NVAR,PHI,Q1,RESID,S,T,THETA,V,XI,Z)
C
C  PUT Q1 INTO Q2 AND G1 INTO G2
C
        DO 2 IPAR1 = 1,NPAR
          Q2(IPAR1) = Q1(IPAR1)
          DO 1 IPAR2 = IPAR1,NPAR
1         G2(IPAR1,IPAR2) = G1(IPAR1,IPAR2)
2       CONTINUE
C
C  CALCULATE CORRECTION FOR THETA
C
        CALL CHOL(G2,Q2,NPAR)
        CALL BKSUB(G2,Q2,1,NPAR,1)
C
C CHECK FOR CONVERGENCE
C
        WDIFF = ZERO
        DO 3 IPAR = 1,NPAR
          W1 = Q2(IPAR)
          W2 = THETA(IPAR)
          WDIFF = MAX(WDIFF,ABS(W1/W2))
          THETA(IPAR) = W2-W1
3       CONTINUE
        IF(WDIFF.LE.CRIT1)GO TO 5
4     CONTINUE
      IFAULT(1) = 1
C
C  FILL LOWER TRIANGLE OF G1 AND PREPARE G2
C  FOR THE INVERSION OF G1
C
5     DO 7 IPAR1 = 1,NPAR
        DO 6 IPAR2 = IPAR1,NPAR
          W1 = G1(IPAR1,IPAR2)
```

```
            G1(IPAR2,IPAR1) = W1
            G2(IPAR1,IPAR2) = W1
            G3(IPAR1,IPAR2) = ZERO
6         CONTINUE
7       CONTINUE
C
C  INVERT G1 INTO G3
C
        CALL CHOL(G2,Q2,NPAR)
        DO 8 IPAR1 = 1,NPAR
8       G3(IPAR1,IPAR1) = ONE/G2(IPAR1,IPAR1)
        CALL BKSUB(G2,G3,NPAR,NPAR,3)
        RETURN
        END


        SUBROUTINE INNER(B,BV,DATA,EST,F,G1,H,IFAULT,NDAT,NEQ,
       * NPAR,NVAR,PHI,Q1,RESID,S,T,THETA,V,XI,Z)
        REAL B(NEQ,NVAR), BV(NEQ,NVAR), CRIT2, DATA(NDAT,NVAR),
       * EST(NDAT,NVAR), F(NEQ), G1(NPAR,NPAR), H(NEQ), HALF,
       * PHI, Q1(NPAR), RESID(NDAT,NVAR), S(NEQ,NEQ),
       * T(NEQ), THETA(NPAR),
       * V(NVAR,NVAR), WDIFF, W1, XI(NVAR), Z(NEQ,NPAR), ZERO
        DIMENSION IFAULT(2)
        PARAMETER (CRIT2=1.E-6, HALF=.5E0, NITER2=20, ZERO=0.E0)
C
C  INITIALIZE PHI, Q1, AND G1
C
        PHI = ZERO
        DO 2 IPAR1 = 1,NPAR
          Q1(IPAR1) = ZERO
          DO 1 IPAR2 = IPAR1,NPAR
1         G1(IPAR1,IPAR2) = ZERO
2       CONTINUE
C
C  PERFORM INNER ITERATION TO CONVERGENCE FOR EACH IDAT
C
        DO 24 IDAT = 1,NDAT
          DO 3 IVAR1 = 1,NVAR
3         XI(IVAR1) = DATA(IDAT,IVAR1)
          DO 16 ITER = 1,NITER2
C
C  THIS LOOP PERFORMS A SINGLE INNER ITERATION
C
            WDIFF = ZERO
            CALL BF(B,F,THETA,XI)
C
C  CALCULATE BV
C
            DO 6 IEQ1 = 1,NEQ
              DO 5 IVAR1 = 1,NVAR
                W1 = ZERO
                DO 4 IVAR2 = 1,NVAR
4               W1 = W1+B(IEQ1,IVAR2)*V(IVAR1,IVAR2)
                BV(IEQ1,IVAR1) = W1
5             CONTINUE
6           CONTINUE
C
C  CALCULATE BVB' AND STORE IT IN S
C
            DO 9 IEQ1 = 1,NEQ
```

```
            DO 8 IEQ2 = IEQ1,NEQ
              W1 = ZERO
              DO 7 IVAR1 = 1,NVAR
7             W1 = W1+BV(IEQ1,IVAR1)*B(IEQ2,IVAR1)
              S(IEQ1,IEQ2) = W1
8           CONTINUE
9         CONTINUE
C
C  PUT F+B(X-XI) INTO H
C
          DO 11 IEQ1 = 1,NEQ
            W1 = F(IEQ1)
            DO 10 IVAR1 = 1,NVAR
10          W1 = W1+B(IEQ1,IVAR1)*(DATA(IDAT,IVAR1)-XI(IVAR1))
            H(IEQ1) = W1
11        CONTINUE
C
C  REPLACE S BY ITS TRIANGULAR FACTORIZATION
C  AND SOLVE ST = H FOR T
C
          CALL CHOL(S,H,NEQ)
          DO 13 IEQ1 = 1,NEQ
            T(IEQ1) = H(IEQ1)
            DO 12 IEQ2 = IEQ1,NEQ
12          S(IEQ2,IEQ1) = S(IEQ1,IEQ2)
13        CONTINUE
          CALL BKSUB(S,T,1,NEQ,1)
C
C  CALCULATE NEW XI AND PUT RELATIVE CHANGE IN XI INTO WDIFF
C
          DO 15 IVAR1 = 1,NVAR
            W1 = DATA(IDAT,IVAR1)
            DO 14 IEQ1 = 1,NEQ
14          W1 = W1 - BV(IEQ1,IVAR1)*T(IEQ1)
            WDIFF = MAX(WDIFF, ABS((W1-XI(IVAR1))/W1))
            XI(IVAR1) = W1
15        CONTINUE
C
C  CHECK FOR CONVERGENCE
C
          IF(WDIFF.LE.CRIT2)GO TO 17
16      CONTINUE
        IFAULT(2) = 1
C
C  ACCUMULATE PHI
C
17      DO 18 IEQ1 = 1,NEQ
18      PHI = PHI + H(IEQ1)**2
C
C  CALCULATE FITTED VALUES AND RESIDUALS
C
        DO 19 IVAR1 = 1,NVAR
          W1 = XI(IVAR1)
          EST(IDAT,IVAR1) = W1
          RESID(IDAT,IVAR1) = DATA(IDAT,IVAR1)-W1
19      CONTINUE
C
C  CALCULATE CONTRIBUTIONS TO Q1 AND G1 AND COMPLETE THE LOOPS
C
        CALL ZED(B,F,THETA,XI,Z)
        CALL BKSUB(S,Z,NPAR,NEQ,2)
```

```
          DO 23 IPAR1 = 1,NPAR
            W1 = Q1(IPAR1)
            DO 20 IEQ1 = 1,NEQ
20          W1 = W1 + Z(IEQ1,IPAR1)*H(IEQ1)
            Q1(IPAR1) = W1
            DO 22 IPAR2 = IPAR1,NPAR
              W1 = G1(IPAR1,IPAR2)
              DO 21 IEQ1 = 1,NEQ
21            W1 = W1 + Z(IEQ1,IPAR1)*Z(IEQ1,IPAR2)
              G1(IPAR1,IPAR2) = W1
22          CONTINUE
23        CONTINUE
24      CONTINUE
        PHI = PHI*HALF
        RETURN
        END


        SUBROUTINE CHOL(A,B,N)
C
C  PERFORM CHOLESKY DECOMPOSITION OF A AND PRELIMINARY TREATMENT
C           OF B
C
        REAL A(N,N),B(N),W1,W2
        DO 5 IR1 = 1,N
          W1 = A(IR1,IR1)
          DO 1 IR2 = 1,IR1-1
1         W1 = W1 - A(IR2,IR1)**2
          W1 = SQRT(W1)
          A(IR1,IR1) = W1
          DO 3 IC1 = IR1+1,N
            W2 = A(IR1,IC1)
            DO 2 IR2 = 1,IR1-1
2           W2 = W2 - A(IR2,IR1)*A(IR2,IC1)
            A(IR1,IC1) = W2/W1
3         CONTINUE
          W2 = B(IR1)
          DO 4 IR2 = 1,IR1-1
4         W2 = W2 - A(IR2,IR1)*B(IR2)
          B(IR1) = W2/W1
5       CONTINUE
        RETURN
        END


        SUBROUTINE BKSUB(A,B,M,N,JOB)
C
C  PERFORM BACK-SOLUTION ACCORDING AS JOB = 1, 2, OR 3
C
        REAL A(N,N),B(N,M),W1
        DO 3 IC1 = M,1,-1
          IF(JOB.EQ.2) THEN
            K1 = 1
            K2 = N
            K3 = 1
          ELSE
            K2 = 1
            K3 = -1
            IF(JOB.EQ.1) THEN
              K1 = N
            ELSE
```

```
            K1 = IC1
          END IF
        END IF
        DO 2 IR1 = K1,K2,K3
          W1 = B(IR1,IC1)
          IF (JOB.EQ.2) THEN
            K2 = IR1-1
          ELSE
            K1 = N
            K2 = IR1+1
          END IF
          DO 1 IR2 = K1,K2,K3
1         W1 = W1 - A(IR1,IR2)*B(IR2,IC1)
          B(IR1,IC1) = W1/A(IR1,IR1)
          IF(JOB.EQ.3) B(IC1,IR1) = B(IR1,IC1)
2       CONTINUE
3     CONTINUE
      RETURN
      END


C-----------------------------------------------------------------------
C
C 4. EX1.DAT

1 1 1 1
-.78 1.39 3.99 11.7 3.44
-1.27 2 5.15 36.58 4.59
2.62 .38 5.77 16.49 .33
2.05 2.92 6.78 75.68 2.74
1.05 2.46 12.85 149.42 7.59
2.96 1.47 3.9 52.11 6.82
1 -.2 .1 .3 0
-.2 1 .2 -.1 0
.1 .2 2 .4 -.2
.3 -.1 .4 5 .4
0 0 -.2 .4 3


C-----------------------------------------------------------------------
C
C 5. EX1.OUT
                                    TABLE 1


DATA
    -0.78     1.39     3.99    11.70     3.44
    -1.27     2.00     5.15    36.58     4.59
     2.62     0.38     5.77    16.49     0.33
     2.05     2.92     6.78    75.68     2.74
     1.05     2.46    12.85   149.42     7.59
     2.96     1.47     3.90    52.11     6.82


V
     1.00    -0.20     0.10     0.30     0.00
    -0.20     1.00     0.20    -0.10     0.00
     0.10     0.20     2.00     0.40    -0.20
     0.30    -0.10     0.40     5.00     0.40
     0.00     0.00    -0.20     0.40     3.00
```

```
INITIAL PARAMETER ESTIMATES =      1.00     1.00     1.00
1.00


FINAL PARAMETER ESTIMATES =  0.822474 0.445297 1.442967 1.001551


FINAL PHI = 0.7515061E+01


FINAL Q =   -0.4954202E-05   0.1068151E-05   0.1989809E-05
0.2217203E-04


ESTIMATED TRUE VALUES OF MEASUREMENTS
  -0.6633   1.4147   3.2692  11.8515   3.4361
  -1.3739   1.9729   5.6814  36.5576   4.7153
   1.7831   1.0674   3.1146  16.5644   2.8529
   2.9277   1.9753   7.6105  76.2357   4.7217
   0.8652   2.7678  12.1155 149.2269   7.2642
   3.1432   1.5559   5.5692  51.9294   3.7197


RESIDUALS
  -0.1167  -0.0247   0.7208  -0.1515   0.0039
   0.1039   0.0271  -0.5314   0.0224  -0.1253
   0.8369  -0.6874   2.6554  -0.0744  -2.5229
  -0.8777   0.9447  -0.8305  -0.5557  -1.9817
   0.1848  -0.3078   0.7345   0.1931   0.3258
  -0.1832  -0.0859  -1.6692   0.1806   3.1003


EXPECTED INFORMATION MATRIX
  72.8418  -3.7697   4.1724 -58.4874
  -3.7697   2.9379   0.7385   6.4070
   4.1724   0.7385  10.2625  18.3139
 -58.4874   6.4070  18.3139 334.4229


INVERSE OF EXPECTED INFORMATION MATRIX
   0.0187   0.0197  -0.0157   0.0037
   0.0197   0.3781  -0.0315  -0.0021
  -0.0157  -0.0315   0.1218  -0.0088
   0.0037  -0.0021  -0.0088   0.0042


FAULT STATUS
    0    0


C--------------------------------------------------------------------
C
C 6. EX2.FOR, EXAMPLE 2
C
C The data for this example were obtained as digitized observations on the
C outline of the nerve-head in a subject's eye.  There are 32 data (NDAT),
C one functional relation (NEQ), two variables (NVAR), and five parameters
C (NPAR).  The functional relation describes an ellipse with the following
C equation
C
```

```
C  F = T3*(XI1-T1)**2 + 2*T4*(XI1-T1)*(XI2-T2) + T5*(XI2-T2)**2 - 1 = 0
C
C where F is the functional value which numerically must be brought to zero,
C T1,T2,T3,T4 and T5 are the functional parameters the values of which are
C to be estimated.  XI1 and XI2 are the estimated true values of the measured
C variables.  The T's are represented by the Greek letter theta in the paper
C and the XI's by xi.  The parameters which describe the magnitude of the
C problem are NDAT = 32, NPAR = 4, NVAR = 2.  The program parameters were
C set at CRIT1 = 1.E-6 and CRIT2 = 1.E-5.  NITER1 was 50 and NITER2 15.  The
C full Fortran code as used is listed under EX2.FOR.  The data in the format
C used by the program is under EX2.DAT.  The output is listed under EX2.OUT.
C
C EX2.FOR


C FITTING OF ELLIPSE BY EVMS  91/3/25 PMR

      DIMENSION BS(2),BVS(2),DATA(32,2),EST(32,2),
     * G1(5,5),G2(5,5),
     * G3(5,5),IFAULT(2),Q1(5),Q2(5),RESID(32,2),THETA(5),V(2,2),
     * XI(2),ZS(5)
      DATA V/1,0,0,1/
C
      READ(21,*)((DATA(I,J),J=1,2),I=1,32)
      DO 2 I = 1,32
        DO 1 J = 1,2
1       DATA(I,J) = .001*DATA(I,J)
2     CONTINUE
      READ(21,*) THETA
      CALL EVMS(BS,BVS,DATA,EST,G1,G2,G3,IFAULT,32,5,2,PHI,
     *    Q1,Q2,RESID,THETA,V,XI,ZS)
      DO 4 I = 1,32
        DO 3 J = 1,2
          DATA(I,J) = 1000.*DATA(I,J)
          EST(I,J) = 1000.*EST(I,J)
          RESID(I,J) = 1000.*RESID(I,J)
3       CONTINUE
4     CONTINUE
      WRITE(31,5)
5     FORMAT(//'FAULT STATUS')
      WRITE(31,6) IFAULT
6     FORMAT(2I5)
      WRITE(31,7)
7     FORMAT(//'PARAMETER ESTIMATES')
      WRITE(31,8) THETA
8     FORMAT(5F9.4)
      WRITE(31,9)
9     FORMAT(//'INFORMATION MATRIX')
      WRITE(31,8) G1
      WRITE(31,10)
10    FORMAT(//'INVERSE OF INFORMATION MATRIX')
      WRITE(31,8) G3
      WRITE(31,11)
11    FORMAT(//6X,'DATA',11X,'ESTIMATES',14X,'RESIDUALS')
      WRITE(31,12)((DATA(I,J),J=1,2),(EST(I,J),J=1,2),
     *  (RESID(I,J),J=1,2),I=1,32)
12    FORMAT(2F7.0,2F11.4,1X,2F10.4)
      STOP
      END
```

```
      SUBROUTINE BF(B,F,THETA,XI)
      DIMENSION B(2),THETA(5),XI(2)
C
C CALCULATE F
      W1 = XI(1)-THETA(1)
      W2 = XI(2)-THETA(2)
      F = THETA(3)*W1*W1 + 2.D0*THETA(4)*W1*W2 + THETA(5)*W2*W2 - 1.D0
C
C CALCULATE B
      B(1) = 2.D0*(W1*THETA(3) + W2*THETA(4))
      B(2) = 2.D0*(W1*THETA(4) + W2*THETA(5))
      RETURN
      END


      SUBROUTINE ZED(B,F,THETA,XI,Z)
      DIMENSION B(2),THETA(5),XI(2),Z(5)
C
      W1 = XI(1)-THETA(1)
      W2 = XI(2)-THETA(2)
      Z(1) = -2.D0*(W1*THETA(3) + W2*THETA(4))
      Z(2) = -2.D0*(W1*THETA(4) + W2*THETA(5))
      Z(3) = W1*W1
      Z(4) = 2.D0*W1*W2
      Z(5) = W2*W2
      RETURN
      END


      SUBROUTINE EVMS(BS,BVS,DATA,EST,G1,G2,G3,IFAULT,
     *    NDAT,NPAR,NVAR,PHI,Q1,Q2,RESID,THETA,V,XI,ZS)
C
C  PARAMETER ESTIMATION FOR THE ERROR-IN-VARIABLES MODEL WITH M = 1
C  91/3/21 PMR
C
      REAL BS(NVAR), BVS(NVAR), BVB, CRIT1, DATA(NDAT,NVAR),
     * EST(NDAT,NVAR), FS, G1(NPAR,NPAR), G2(NPAR,NPAR),
     * G3(NPAR,NPAR), HS, ONE, PHI, Q1(NPAR), Q2(NPAR),
     * RESID(NDAT,NVAR), THETA(NPAR),
     * V(NVAR,NVAR), WDIFF, W1, W2, XI(NVAR), ZS(NPAR), ZERO
      DIMENSION IFAULT(2)
      PARAMETER(CRIT1=1.E-6,NITER1=50,ONE=1.E0,ZERO=0.E0)
C
      IFAULT(1) = 0
      IFAULT(2) = 0
      DO 4 ITER = 1,NITER1
C
C  EACH PASS THROUGH THIS LOOP PERFORMS A SINGLE OUTER ITERATION
C
        CALL INNERS(BS,BVS,DATA,EST,G1,IFAULT,NDAT,
     *      NPAR,NVAR,PHI,Q1,RESID,THETA,V,XI,ZS)
C
C  PUT Q1 INTO Q2 AND G1 INTO G2
C
        DO 2 IPAR1 = 1,NPAR
          Q2(IPAR1) = Q1(IPAR1)
          DO 1 IPAR2 = IPAR1,NPAR
1         G2(IPAR1,IPAR2) = G1(IPAR1,IPAR2)
2       CONTINUE
C
C  CALCULATE CORRECTION FOR THETA
```

```
C
        CALL CHOL(G2,Q2,NPAR)
        CALL BKSUB(G2,Q2,1,NPAR,1)
C
C CHECK FOR CONVERGENCE
C
        WDIFF = ZERO
        DO 3 IPAR = 1,NPAR
          W1 = Q2(IPAR)
          W2 = THETA(IPAR)
          WDIFF = MAX(WDIFF,ABS(W1/W2))
          THETA(IPAR) = W2-W1
3       CONTINUE
        IF(WDIFF.LE.CRIT1)GO TO 5
4     CONTINUE
      IFAULT(1) = 1
C
C  FILL LOWER TRIANGLE OF G1 AND PREPARE G2
C  FOR THE INVERSION OF G1
C
5     DO 7 IPAR1 = 1,NPAR
        DO 6 IPAR2 = IPAR1,NPAR
          W1 = G1(IPAR1,IPAR2)
          G1(IPAR2,IPAR1) = W1
          G2(IPAR1,IPAR2) = W1
          G3(IPAR1,IPAR2) = ZERO
6       CONTINUE
7     CONTINUE
C
C  INVERT G1 INTO G3
C
      CALL CHOL(G2,Q2,NPAR)
      DO 8 IPAR1 = 1,NPAR
8     G3(IPAR1,IPAR1) = ONE/G2(IPAR1,IPAR1)
      CALL BKSUB(G2,G3,NPAR,NPAR,3)
      RETURN
      END


      SUBROUTINE INNERS(BS,BVS,DATA,EST,G1,IFAULT,NDAT,
     *    NPAR,NVAR,PHI,Q1,RESID,THETA,V,XI,ZS)
C
C  INNER ITERATION FOR M = 1
C
      REAL BS(NVAR), BVS(NVAR), BVB, CRIT2, DATA(NDAT,NVAR),
     * EST(NDAT,NVAR), FS, HALF, G1(NPAR,NPAR), HS,
     * PHI, Q1(NPAR), RESID(NDAT,NVAR), THETA(NPAR),
     * V(NVAR,NVAR), WDIFF, W1, XI(NVAR), ZS(NPAR), ZERO
      DIMENSION IFAULT(2)
      PARAMETER (CRIT2=1.E-5, HALF=.5E0, NITER2=15, ZERO=0.E0)
C
C  INITIALIZE PHI, Q1, AND G1
C
      PHI = ZERO
      DO 2 IPAR1 = 1,NPAR
        Q1(IPAR1) = ZERO
        DO 1 IPAR2 = IPAR1,NPAR
1       G1(IPAR1,IPAR2) = ZERO
2     CONTINUE
C
C  PERFORM INNER ITERATION TO CONVERGENCE FOR EACH IDAT
```

```
C
      DO 14 IDAT = 1,NDAT
C
C  INITIALIZE XI
C
        DO 3 IVAR1 = 1,NVAR
3       XI(IVAR1) = DATA(IDAT,IVAR1)
        DO 9 ITER = 1,NITER2
C
C  THIS LOOP PERFORMS A SINGLE INNER ITERATION
C
          WDIFF = ZERO
          CALL BF(BS,FS,THETA,XI)
C
C  CALCULATE BVS
C
          DO 5 IVAR1 = 1,NVAR
            W1 = ZERO
            DO 4 IVAR2 = 1,NVAR
4           W1 = W1 + BS(IVAR2)*V(IVAR1,IVAR2)

            BVS(IVAR1) = W1
5         CONTINUE
C
C  CALCULATE BVB
C
          BVB = ZERO
          DO 6 IVAR1 = 1,NVAR
6         BVB = BVB + BVS(IVAR1)*BS(IVAR1)
C
C  PUT F+B(X-XI) INTO HS
C
          HS = FS
          DO 7 IVAR1 = 1,NVAR
7         HS = HS + BS(IVAR1)*(DATA(IDAT,IVAR1) - XI(IVAR1))
C
C  CALCULATE NEW XI AND PUT RELATIVE CHANGE IN XI INTO WDIFF
C
          DO 8 IVAR1 = 1,NVAR
            W1 = DATA(IDAT,IVAR1) - BVS(IVAR1)*HS/BVB
            WDIFF = MAX(WDIFF, ABS((W1-XI(IVAR1))/W1))
            XI(IVAR1) = W1
8         CONTINUE
C
C  CHECK FOR CONVERGENCE
C
          IF(WDIFF.LE.CRIT2)GO TO 10
9       CONTINUE
        IFAULT(2) = 1
C
C  ACCUMULATE PHI
C
10      PHI = PHI+HS*HS/BVB
C
C  CALCULATE FITTED VALUES AND RESIDUALS
C
        DO 11 IVAR1 = 1,NVAR
          W1 = XI(IVAR1)
          EST(IDAT,IVAR1) = W1
          RESID(IDAT,IVAR1) = DATA(IDAT,IVAR1) - W1
11      CONTINUE
```

```
C
C  CALCULATE CONTRIBUTIONS TO Q1 AND G1 AND COMPLETE THE LOOPS
C
        CALL ZED(BS,FS,THETA,XI,ZS)
        DO 13 IPAR1 = 1,NPAR
          W1 = ZS(IPAR1)
          Q1(IPAR1) = Q1(IPAR1) + W1*HS/BVB
          DO 12 IPAR2 = IPAR1,NPAR
12        G1(IPAR1,IPAR2) = G1(IPAR1,IPAR2) + W1*ZS(IPAR2)/BVB
13      CONTINUE
14    CONTINUE
      PHI = PHI*HALF
      RETURN
      END


      SUBROUTINE CHOL(A,B,N)
C
C  PERFORM CHOLESKY DECOMPOSITION OF A AND PRELIMINARY TREATMENT
C     OF B
C
      REAL A(N,N),B(N),W1,W2
      DO 5 IR1 = 1,N
        W1 = A(IR1,IR1)
        DO 1 IR2 = 1,IR1-1
1       W1 = W1-A(IR2,IR1)**2
        W1 = SQRT(W1)
        A(IR1,IR1) = W1
        DO 3 IC1 = IR1+1,N
          W2 = A(IR1,IC1)
          DO 2 IR2 = 1,IR1-1
2         W2 = W2 - A(IR2,IR1)*A(IR2,IC1)
          A(IR1,IC1) = W2/W1
3       CONTINUE
        W2 = B(IR1)
        DO 4 IR2 = 1,IR1-1
4       W2 = W2 - A(IR2,IR1)*B(IR2)
        B(IR1) = W2/W1
5     CONTINUE
      RETURN
      END


      SUBROUTINE BKSUB(A,B,M,N,JOB)
C
C  PERFORM BACK-SOLUTION ACCORDING AS JOB = 1, 2, OR 3
C
      REAL A(N,N),B(N,M),W1
      DO 3 IC1 = M,1,-1
        IF(JOB.EQ.2) THEN
          K1 = 1
          K2 = N
          K3 = 1
        ELSE
          K2 = 1
          K3 = -1
          IF(JOB.EQ.1) THEN
            K1 = N
          ELSE
            K1 = IC1
          END IF
```

```
         END IF
         DO 2 IR1 = K1,K2,K3
           W1 = B(IR1,IC1)
           IF (JOB.EQ.2) THEN
             K2 = IR1-1
           ELSE
             K1 = N
             K2 = IR1+1
           END IF
           DO 1 IR2 = K1,K2,K3
1          W1 = W1-A(IR1,IR2)*B(IR2,IC1)
           B(IR1,IC1) = W1/A(IR1,IR1)
           IF(JOB.EQ.3) B(IC1,IR1) = B(IR1,IC1)
2        CONTINUE
3      CONTINUE
       RETURN
       END

C-------------------------------------------------------------------------
```

7. EX2.DAT

```
1420 470
1188 476
1015 539
830 655
660 825
543 1041
468 1268
459 1528
510 1804
628 2063
795 2221
1040 2345
1303 2417
1452 2431
1738 2411
1901 2318
2091 2222
2230 2117
2311 1984
2385 1837
2451 1630
2467 1462
2472 1263
2446 1112
2387 963
2255 796
2109 696
1929 597
1846 556
1620 486
1418 437
1259 453
1 1 1  -.001 1
```

8. EX2.OUT

```
FAULT STATUS
    0    0
```

PARAMETER ESTIMATES

1.4594	1.4454	0.9582	0.0039	1.0186

INFORMATION MATRIX

14.6768	0.4050	-1.1978	0.2560	-0.2880
0.4050	17.3231	0.1325	-0.6215	1.2703
-1.1978	0.1325	3.1070	0.0154	0.9994
0.2560	-0.6215	0.0154	3.9975	0.1110
-0.2880	1.2703	0.9994	0.1110	3.1565

INVERSE OF INFORMATION MATRIX

0.0705	-0.0019	0.0277	-0.0049	-0.0014
-0.0019	0.0600	0.0051	0.0102	-0.0263
0.0277	0.0051	0.3698	0.0008	-0.1166
-0.0049	0.0102	0.0008	0.2524	-0.0137
-0.0014	-0.0263	-0.1166	-0.0137	0.3647

DATA		ESTIMATES		RESIDUALS	
1420.	470.	1419.3920	455.4515	0.6078	14.5485
1188.	476.	1191.7790	490.1515	-3.7791	-14.1515
1015.	539.	1021.0570	552.0391	-6.0567	-13.0391
830.	655.	835.8889	662.8408	-5.8889	-7.8408
660.	825.	663.1567	827.6081	-3.1567	-2.6081
543.	1041.	531.0313	1035.3420	11.9687	5.6577
468.	1268.	455.4884	1265.5690	12.5116	2.4308
459.	1528.	441.1519	1529.4960	17.8481	-1.4961
510.	1804.	506.2181	1805.5060	3.7819	-1.5061
628.	2063.	646.4303	2048.4960	-18.4303	14.5044
795.	2221.	804.9326	2208.6670	-9.9326	12.3329
1040.	2345.	1038.0070	2349.5790	1.9934	-4.5788
1303.	2417.	1301.8300	2424.9300	1.1703	-7.9303
1452.	2431.	1451.9830	2436.1840	0.0166	-5.1842
1738.	2411.	1734.5620	2398.5040	3.4379	12.4955
1901.	2318.	1908.4610	2333.5910	-7.4607	-15.5907
2091.	2222.	2090.8450	2221.7980	0.1550	0.2022
2230.	2117.	2218.0730	2105.9520	11.9267	11.0476
2311.	1984.	2313.4760	1985.6700	-2.4755	-1.6704
2385.	1837.	2394.2520	1841.1940	-9.2518	-4.1943
2451.	1630.	2461.8330	1632.1880	-10.8328	-2.1882
2467.	1462.	2480.6950	1462.2970	-13.6950	-0.2967
2472.	1263.	2464.4650	1264.4130	7.5355	-1.4126
2446.	1112.	2425.3450	1119.3370	20.6547	-7.3367
2387.	963.	2361.4770	977.0116	25.5227	-14.0116
2255.	796.	2242.8230	806.5399	12.1768	-10.5398
2109.	696.	2117.7400	685.2596	-8.7397	10.7405
1929.	597.	1942.6510	570.5976	-13.6507	26.4024
1846.	556.	1856.1900	530.8438	-10.1901	25.1563
1620.	486.	1622.9630	466.7079	-2.9625	19.2921
1418.	437.	1418.7830	455.4772	-0.7826	-18.4772
1259.	453.	1263.0220	473.7592	-4.0221	-20.7591

```
C        .. ULCC Toolpack/1 2.1
      SUBROUTINE initial(ns, m, emax, x, hx, hpx, lb, xlb, ub, xub,
     *                   ifault, iwv, rwv)
C
C        ALGORITHM AS 287.1 APPL.STATIST. (1993), VOL.42, NO.4
C
C        This subroutine takes as input the number of starting values M
C        and the starting points x(i), hx(i), hpx(i)  i=1, m. As output
C        we have pointer ipt along with ilow and ihigh and the lower
C        and upper hulls defined  by z, hz, scum, cu, hulb, huub stored
C        in working vectors iwv and rwv
C
      INTEGER ns, m, ifault, iwv( * )
      LOGICAL lb, ub
      REAL emax, x( * ), hx( * ), hpx( * ), xlb, xub, rwv( * )
C
      INTEGER i, ihigh, ihpx, ihuz, ihx, iipt, ilow, iscum, ix, iz, nn
      LOGICAL HORIZ
      REAL a, b, alcu, cu, eps, expon, hulb, huub, huzmax, zlog, zmax
C
      zlog(a) = alog(a)
      zmax(a, b) = amax1(a, b)
      eps = expon(-emax, emax)
C
      ifault = 0
      ilow = 1
      ihigh = 1
      nn = ns + 1
C
C        at least one starting point
C
      IF (m .LT. 1) ifault = 1
C
      huzmax = hx(1)
C
      IF ( .NOT. ub) xub = 0.0
      IF ( .NOT. lb) xlb = 0.0
C
      hulb = (xlb - x(1)) * hpx(1) + hx(1)
      huub = (xub - x(1)) * hpx(1) + hx(1)
C
C        if bounded on both sides
C
      IF ((ub) .AND. (lb)) THEN
         huzmax = zmax(huub, hulb)
         horiz = (abs(hpx(1)) .LT. eps)
         IF (horiz) THEN
            cu = expon((huub + hulb) * 0.5 - huzmax, emax) * (xub - xlb)
         ELSE
            cu = expon(huub - huzmax, emax) *
     *           (1 - expon(hulb - huub, emax)) / hpx(1)
         END IF
      ELSE IF ((ub) .AND. ( .NOT. lb)) THEN
C
C        if bounded on the right and unbounded on the left
C
         huzmax = huub
         cu = 1.0 / hpx(1)
      ELSE IF (( .NOT. ub) .AND. (lb)) THEN
C
C        if bounded on the left and unbounded on the right
```

```
C
         huzmax = hulb
         cu = -1.0 / hpx(1)
C
C        if unbounded at least 2 starting points
C
      ELSE
         cu = 0.0
         IF (m .LT. 2) ifault = 1
      END IF
C
      IF (cu .GT. 0.0) alcu = zlog(cu)
C
C        set pointers
C
      iipt = 6
      iz = 9
      ihuz = nn + iz
      iscum = nn + ihuz
      ix = nn + iscum
      ihx = nn + ix
      ihpx = nn + ihx
C
C        store values in working vectors
C
      iwv(1) = ilow
      iwv(2) = ihigh
      iwv(3) = ns
      iwv(4) = 1
C
      IF (lb) THEN
         iwv(5) = 1
      ELSE
         iwv(5) = 0
      END IF
C
      IF (ub) THEN
         iwv(6) = 1
      ELSE
         iwv(6) = 0
      END IF
C
      IF (ns .LT. m) ifault = 2
C
      iwv(iipt + 1) = 0
      rwv(1) = hulb
      rwv(2) = huub
      rwv(3) = emax
      rwv(4) = eps
      rwv(5) = cu
      rwv(6) = alcu
      rwv(7) = huzmax
      rwv(8) = xlb
      rwv(9) = xub
      rwv(iscum + 1) = 1.0
      DO 10 i = 1, m
         rwv(ix + i) = x(i)
         rwv(ihx + i) = hx(i)
         rwv(ihpx + i) = hpx(i)
   10 CONTINUE
C
```

```
C        create lower and upper hulls
C
      i = 1
   20 IF (i .LT. m) THEN
         CALL update(iwv(4), iwv(1), iwv(2), iwv(iipt + 1),
     *               rwv(iscum + 1), rwv(5), rwv(ix + 1), rwv(ihx + 1),
     *               rwv(ihpx + 1), rwv(iz + 1), rwv(ihuz + 1), rwv(7),
     *               rwv(3), lb, rwv(8), rwv(1), ub, rwv(9), rwv(2),
     *               ifault, rwv(4), rwv(6))
         i = iwv(4)
         IF (ifault .NE. 0) RETURN
         GO TO 20
      END IF
C
C        test for wrong starting points
C
      IF (( .NOT. lb) .AND. (hpx(iwv(1)) .LT. eps)) ifault = 3
      IF (( .NOT. ub) .AND. (hpx(iwv(2)) .GT. -eps)) ifault = 4
C
      RETURN
      END
      SUBROUTINE sample(iwv, rwv, eval, beta, ifault)
C
C        ALGORITHM AS 287.2 APPL.STATIST. (1993), VOL.42, NO.4
C
      INTEGER iwv( * ), ifault
      REAL rwv( * ), beta
C
      INTEGER ihpx, ihuz, ihx, iipt, iscum, ix, iz, nn, ns
      LOGICAL lb, ub
      EXTERNAL EVAL
C
C        set pointers
C
      iipt = 6
      iz = 9
      ns = iwv(3)
      nn = ns + 1
      ihuz = nn + iz
      iscum = nn + ihuz
      ix = nn + iscum
      ihx = nn + ix
      ihpx = nn + ihx
      lb = .FALSE.
      ub = .FALSE.
C
      IF (iwv(5) .EQ. 1) lb = .TRUE.
      IF (iwv(6) .EQ. 1) ub = .TRUE.
C
C        call sampling subroutine
C
      CALL spl1(ns, iwv(4), iwv(1), iwv(2), iwv(iipt + 1),
     *          rwv(iscum + 1), rwv(5), rwv(ix + 1), rwv(ihx + 1),
     *          rwv(ihpx + 1), rwv(iz + 1), rwv(ihuz + 1), rwv(7), lb,
     *          rwv(8), rwv(1), ub, rwv(9), rwv(2), eval, beta, ifault,
     *          rwv(3), rwv(4), rwv(6))
C
      RETURN
      END
      SUBROUTINE spl1(ns, n, ilow, ihigh, ipt, scum, cu, x, hx, hpx, z,
     *                huz, huzmax, lb, xlb, hulb, ub, xub, huub, eval,
```

```
     *                beta, ifault, emax, eps, alcu)
C
C        ALGORITHM AS 287.3 APPL.STATIST. (1993), VOL.42, NO.4
C
C        this subroutine performs the adaptive rejection sampling, it
C        calls subroutine splhull to sample from the upper hull, if the
C        sampling involves a function evaluation it calls the updating
C        subroutine
C
      INTEGER ns, n, ilow, ihigh, ipt( * ), ifault
      LOGICAL lb, ub
      REAL scum( * ), cu, x( * ), hx( * ), hpx( * ), z( * ), huz( * ),
     *     huzmax, xlb, hulb, xub, huub, beta, emax, eps, alcu
C
      INTEGER i, j, l, n1
      LOGICAL sampld
      REAL a, alhl, alhu, alu1, fx, random, u1, u2, zlog
      EXTERNAL eval
C
      zlog(a) = alog(a)
C
      sampld = .FALSE.
   10 IF ( .NOT. sampld) THEN
         u2 = random(L)
C
C        test for zero random number
C
         IF (u2 .EQ. 0.0) THEN
            ifault = 6
            RETURN
         END IF
         CALL splhull(u2, ipt, ilow, lb, xlb, hulb, huzmax, alcu, x, hx,
     *                hpx, z, huz, scum, eps, emax, beta, i, j)
C
C        sample u1 to compute rejection
C
         u1 = random(l)
         IF (u1 .EQ. 0.0) ifault = 6
         alu1 = zlog(u1)
C
C        compute alhu: upper hull at point u1
C
         alhu = hpx(i) * (beta - x(i)) + hx(i) - huzmax
         IF ((beta .GT. x(ilow)) .AND. (beta .LT. x(ihigh))) THEN
C
C        compute alhl: value of the lower hull at point u1
C
            IF (beta .GT. x(i)) THEN
               j = i
               i = ipt(i)
            END IF
            alhl = hx(i) + (beta - x(i)) * (hx(i) - hx(j)) /
     *             (x(i) - x(j)) - huzmax
C
C        squeezing test
C
            IF ((alhl - alhu) .GT. alu1) THEN
               sampld = .TRUE.
            END IF
         END IF
C
```

```
C        if not sampled evaluate the function, do the rejection test
C        and update
C
         IF ( .NOT. sampld) THEN
            n1 = n + 1
            x(n1) = beta
            CALL eval(x(n1), hx(n1), hpx(n1))
            fx = hx(n1) - huzmax
            IF (alu1 .LT. (fx - alhu)) sampld = .TRUE.
C
C        update while the number of points defining the hulls is lower
C        than ns
C
            IF (n .LT. ns) THEN
               CALL update(n, ilow, ihigh, ipt, scum, cu, x, hx, hpx, z,
     *                     huz, huzmax, emax, lb, xlb, hulb, ub, xub,
     *                     huub, ifault, eps, alcu)
            END IF
            IF (ifault .NE. 0) RETURN
         END IF
         GO TO 10
       END IF
C
       RETURN
       END
       SUBROUTINE splhull(u2, ipt, ilow, lb, xlb, hulb, huzmax, alcu, x,
     *                    hx, hpx, z, huz, scum, eps, emax, beta, i, j)
C
C        ALGORITHM AS 287.4 APPL.STATIST. (1993), VOL.42, NO.4
C
C        this subroutine samples beta from the normalised upper hull
C
       REAL u2, xlb, hulb, huzmax, alcu, x( * ), hx( * ), hpx( * ),
     *      z( * ), huz( * ), scum( * ), eps, emax, beta
       INTEGER ipt( * ), ilow, i, j
       LOGICAL lb
C
       REAL a, eh, expon, logdu, logtg, sign, zlog
       LOGICAL horiz
C
       zlog(a) = alog(a)
       i = ilow
C
C        find from which exponential piece you sample
C
   10  IF (u2 .GT. scum(i)) THEN
          j = i
          i = ipt(i)
          GO TO 10
       END IF
C
       IF (i .EQ. ilow) THEN
C
C        sample below z(ilow), depending on the existence of a lower
C        bound
C
          IF (lb) THEN
             eh = hulb - huzmax - alcu
             horiz = (abs(hpx(ilow)) .LT. eps)
             IF (horiz) THEN
                beta = xlb + u2 * expon(-eh, emax)
```

```
               ELSE
                  sign = abs(hpx(i)) / hpx(i)
                  logtg = zlog(abs(hpx(i)))
                  logdu = zlog(u2)
                  eh = logdu + logtg - eh
                  IF (eh .LT. emax) THEN
                     beta = xlb + zlog(1.0 + sign * expon(eh, emax)) /
     *                    hpx(i)
                  ELSE
                     beta = xlb + eh / hpx(i)
                  END IF
               END IF
            ELSE
C
C              hpx(i) must be positive, x(ilow) is left of the mode
C
               beta = (zlog(hpx(i) * u2) + alcu - hx(i) + x(i) * hpx(i) +
     *                huzmax) / hpx(i)
            END IF
         ELSE
C
C           sample above (j)
C
            eh = huz(j) - huzmax - alcu
            horiz = (abs(hpx(i)) .LT. eps)
            IF (horiz) THEN
               beta = z(j) + (u2 - scum(j)) * expon(-eh, emax)
            ELSE
               sign = abs(hpx(i)) / hpx(i)
               logtg = zlog(abs(hpx(i)))
               logdu = zlog(u2 - scum(j))
               eh = logdu + logtg - eh
               IF (eh .LT. emax) THEN
                  beta = z(j) + (zlog(1.0 + sign * expon(eh, emax))) /
     *                  hpx(i)
               ELSE
                  beta = z(j) + eh / hpx(i)
               END IF
            END IF
         END IF
C
         RETURN
         END
         SUBROUTINE intersection(x1, y1, yp1, x2, y2, yp2, z1, hz1, eps,
     *                           ifault)
C
C           ALGORITHM AS 287.5 APPL.STATIST. (1993), VOL.42, NO.4
C
C           computes the intersection (z1,hz1) between 2 tangents defined
C           by x1, y1, yp1 and x2, y2, yp2
C
         REAL x1, y1, yp1, x2, y2, yp2, z1, hz1, eps
         INTEGER ifault
C
         REAL y12, dh
C
C           first test for non-concavity
C
         y12 = y1 + yp1 * (x2 - x1)
         y21 = y2 + yp2 * (x1 - x2)
         IF ((y21 .LT. y1) .OR. (y12 .LT. y2)) THEN
```

```
            ifault = 5
            RETURN
         END IF
C
         dh = yp2 - yp1
C
C        IF the lines are nearly parallel,
C        the intersection is taken at the mid-point
C
         IF (abs(dh) .LE. eps) THEN
            z1 = 0.5 * (x1 + x2)
            hz1 = 0.5 * (y1 + y2)
C
C        Else compute from the left or the right for greater
C        numerical precision
C
         ELSE IF (abs(yp1) .LT. abs(yp2)) THEN
            z1 = x2 + (y1 - y2 + yp1 * (x2 - x1)) / dh
            hz1 = yp1 * (z1 - x1) + y1
         ELSE
            z1 = x1 + (y1 - y2 + yp2 * (x2 - x1)) / dh
            hz1 = yp2 * (z1 - x2) + y2
         END IF
C
C        test for misbehaviour due to numerical imprecision
C
         IF ((z1 .LT. x1) .OR. (z1 .GT. x2)) ifault = 7
C
         RETURN
         END
         SUBROUTINE update(n, ilow, ihigh, ipt, scum, cu, x, hx, hpx, z,
        *                  huz, HUZMAX, EMAX, lb, xlb, hulb, ub, xub, huub,
        *                  ifault, eps, alcu)
C
C        ALGORITHM AS 287.6 APPL.STATIST. (1993), VOL.42, NO.4
C
C        this subroutine increments n and updates all the parameters
C        which define the lower and the upper hull
C
         INTEGER n, ilow, ihigh, ipt( * ), ifault
         LOGICAL lb, ub
         REAL scum( * ), cu, x( * ), hx( * ), hpx( * ), z( * ), huz( * ),
        *     huzmax, emax, xlb, hulb, xub, huub, eps, alcu
C
         INTEGER i, j
         LOGICAL horiz
         REAL a, b, dh, expon, u, zlog, zmax
         EXTERNAL eval
C
         zlog(a) = alog(a)
         zmax(a, b) = amax1(a, b)
C
         n = n + 1
C
C        update z, huz and ipt
C
         IF (x(n) .LT. x(ilow)) THEN
C
C        insert x(n) below x(ilow), test for non-concavity
C
            IF (hpx(ilow) .GT. hpx(n)) ifault = 5
```

```
C
         ipt(n) = ilow
         CALL intersection(x(n), hx(n), hpx(n), x(ilow), hx(ilow),
     *                     hpx(ilow), z(n), huz(n), eps, ifault)
C
         IF (ifault .NE. 0) RETURN
         IF (lb) hulb = hpx(n) * (xlb - x(n)) + hx(n)
C
         ilow = n
       ELSE
         i = ilow
         j = i
C
C        find where to insert x(n)
C
   10    IF ((x(n) .GE. x(i)) .AND. (ipt(i) .NE. 0)) THEN
            j = i
            i = ipt(i)
            GO TO 10
         END IF
C
         IF (x(n) .GE. x(i)) THEN
C
C        insert above x(ihigh), test for non-concavity
C
            IF (hpx(i) .LT. hpx(n)) ifault = 5
C
            ihigh = n
            ipt(i) = n
            ipt(n) = 0
            CALL intersection(x(i), hx(i), hpx(i), x(n), hx(n), hpx(n),
     *                        z(i), huz(i), eps, ifault)
C
            IF (ifault .NE. 0) RETURN
C
            huub = hpx(n) * (xub - x(n)) + hx(n)
            z(n) = 0.0
            huz(n) = 0.0
         ELSE
C
C        insert x(n) between x(j) and x(i), test for non-concavity
C
            IF ((hpx(j) .LT. hpx(n)) .OR.
     *          (hpx(i) .GT. hpx(n))) ifault = 5
C
            ipt(j) = n
            ipt(n) = i
C
C        insert z(j) between x(j) and x(n)
C
            CALL intersection(x(j), hx(j), hpx(j), x(n), hx(n), hpx(n),
     *                        z(j), huz(j), eps, ifault)
C
            IF (ifault .NE. 0) RETURN
C
C        insert z(n) between x(n) and x(i)
C
            CALL intersection(x(n), hx(n), hpx(n), x(i), hx(i), hpx(i),
     *                        z(n), huz(n), eps, ifault)
C
            IF (ifault .NE. 0) RETURN
```

```
C
         END IF
       END IF
C
C        update huzmax
C
       j = ilow
       i = ipt(j)
       huzmax = huz(j)
C
   20  IF ((huz(j) .LT. huz(i)) .AND. (ipt(i) .NE. 0)) THEN
          j = i
          i = ipt(i)
          huzmax = zmax(huzmax, huz(j))
          GO TO 20
       END IF
C
       IF (lb) huzmax = zmax(huzmax, hulb)
       IF (ub) huzmax = zmax(huzmax, huub)
C
C        update cu, scum receives area below exponentiated upper hull
C        left of z(i)
C
       i = ilow
       horiz = (abs(hpx(ilow)) .LT. eps)
       IF (( .NOT. lb) .AND. ( .NOT. horiz)) THEN
          cu = expon(huz(i) - huzmax, emax) / hpx(i)
       ELSE IF (lb .AND. horiz) THEN
          cu = (z(ilow) - xlb) * expon(hulb - huzmax, emax)
       ELSE IF (lb .AND. ( .NOT. horiz)) THEN
          dh = hulb - huz(i)
          IF (dh .GT. emax) THEN
             cu = -expon(hulb - huzmax, emax) / hpx(i)
          ELSE
             cu = expon(huz(i) - huzmax, emax) * (1 - expon(dh, emax)) /
     *            hpx(i)
          END IF
       ELSE
          cu = 0
       END IF
       scum(i) = cu
       j = i
       i = ipt(i)
C
   30  IF (ipt(i) .NE. 0) THEN
          dh = huz(j) - huz(i)
          horiz = (abs(hpx(i)) .LT. eps)
          IF (horiz) THEN
             cu = cu + (z(i) - z(j)) * expon((huz(i) + huz(j)) * 0.5 -
     *            huzmax, emax)
          ELSE
             IF (dh .LT. emax) THEN
                cu = cu + expon(huz(i) - huzmax, emax) *
     *               (1 - expon(dh, emax)) / hpx(i)
             ELSE
                cu = cu - expon(huz(j) - huzmax, emax) / hpx(i)
             END IF
          END IF
          j = i
          i = ipt(i)
          scum(j) = cu
```

```
            GO TO 30
         END IF
C
         horiz = (abs(hpx(i)) .LT. eps)
C
C           if the derivative is very small the tangent is nearly
C           horizontal
C
         IF ( .NOT. (ub .OR. horiz)) THEN
            cu = cu - expon(huz(j) - huzmax, emax) / hpx(i)
         ELSE IF (ub .AND. horiz) THEN
            cu = cu + (xub - x(i)) * expon((huub + hx(i)) * 0.5 - huzmax,
     *           emax)
         ELSE IF (ub .AND. ( .NOT. horiz)) THEN
            dh = huz(j) - huub
            IF (dh .GT. emax) THEN
               cu = cu - expon(huz(j) - huzmax, emax) / hpx(i)
            ELSE
               cu = cu + expon(huub - huzmax, emax) *
     *              (1 - expon(dh, emax)) / hpx(i)
            END IF
         END IF
C
         scum(i) = cu
         IF (cu .GT. 0) alcu = zlog(cu)
C
C           normalize scum to obtain a cumulative probability while
C           excluding unnecessary points
C
         i = ilow
         u = (cu - scum(i)) / cu
C
         IF ((u .EQ. 1.0) .AND. (hpx(ipt(i)) .GT. 0)) THEN
            ilow = ipt(i)
            scum(i) = 0.0
         ELSE
            scum(i) = 1.0 - u
         END IF
C
         j = i
         i = ipt(i)
C
   40    IF (ipt(i) .NE. 0) THEN
            j = i
            i = ipt(i)
            u = (cu - scum(j)) / cu
            IF ((u .EQ. 1.0) .AND. (hpx(i) .GT. 0)) THEN
               ilow = i
            ELSE
               scum(j) = 1.0 - u
            END IF
            GO TO 40
         END IF
C
         scum(i) = 1.0
         IF (ub) huub = hpx(ihigh) * (xub - x(ihigh)) + hx(ihigh)
         IF (lb) hulb = hpx(ilow) * (xlb - x(ilow)) + hx(ilow)
C
         RETURN
         END
         REAL FUNCTION expon(x, emax)
```

```
C
C        ALGORITHM AS 287.7 APPL.STATIST. (1993), VOL.42, NO.4
C
C        performs an exponential without underflow
C
      REAL x, emax
C
      REAL ZEXP
C
      zexp(x) = exp(x)
C
      IF (x .LT. -emax) THEN
         expon = 0.0
      ELSE
         expon = zexp(x)
      END IF
C
      RETURN
      END
```

```
      SUBROUTINE GSMIRN (NX, NY, KIND, M, DSTAT, P, Q, IFAULT)
C
C  ALGORITHM AS 288 APPL.STATIST. (1994), VOL.43, NO.1
C
C  P-value calculation for the generalized two-sample
C  Smirnov tests.
C  The tests are conditional on ties in the pooled sample.
C
      DIMENSION P(*), M(*)
      REAL P, DSTAT, Q, SLOPE, DELTA, DEVIAT, DL, ONE, ZERO, EPS, FLO
     *, SMALL, SCL, SMALLN, T, FL, ALN2, CHKNUM
      LOGICAL NEWRCT
      DATA ONE /1.0/ ZERO /0.0/ EPS /1E-6/
      DATA SMALL /1E-35/ SMALLN /-80.5904782547916/
      DATA ALN2 /0.69314718056/ CHKNUM /1E32/ ITERUP /116/
      FLO(L) = FLOAT(L)
C
      N = NX + NY
      IFAULT = 1
      IF (NX .LE. 0 .OR. NY .LE. 0) RETURN
      IFAULT = 2
      IF (KIND .LT. 1 .OR. KIND .GT. 3) RETURN
      IFAULT = 4
      ICAT = 0
      L = 0
    1 L = L + 1
      IF (M(L) .LE. 0) RETURN
      ICAT = ICAT + M(L)
      IF (ICAT - N) 1, 3, 2
    2 RETURN
C
    3 IFAULT = 0
      Q = ONE
      DELTA = DSTAT - EPS
      IF (DELTA .LE. ZERO) RETURN
      P(1) = ONE
C
C  Parameters to define a set for trajectories to lie within it
C
      SLOPE = NX / FLO(N)
      DEVIAT = SLOPE * DELTA * NY
C*    DELTA = DEVIAT
      NEWRCT = .TRUE.
      ICAT = 1
      NTIES = M(1)
      ILE = 0
      IRI = 0
C
C  Variables to prevent from overflows in P
C
      IC = ITERUP
      NOFDIV = 0
      SCL = ONE
C
      DO 100 L = 1, N - 1
        IF (NTIES .EQ. 1) THEN
C
C  Calculate boundaries for current L (if Lth value in the
C  pooled sample is unique or `last' in a series of tied values)
C
          DL = L * SLOPE
```

```
C*        T = FLOAT (L) / N
C*        DEVIAT = DELTA * SQRT (T * (ONE - T))
          IRI = MIN0 (INT (DL + DEVIAT), L, NX)
          ILE = MAX0 (INT (DL - DEVIAT + ONE), L - NY, 0)
          ICAT = ICAT + 1
          NTIES = M(ICAT)
          NEWRCT = .TRUE.
        ELSE
C
C      Calculations for tied observations
C
          NTIES = NTIES - 1
C
C  If we have the first observation with the new value (that
C  is not unique), then determine the ICATth `rectangle'
C
          IF (NEWRCT) THEN
            NEWRCT = .FALSE.
            L2 = L + NTIES
C*          IF (L2 .EQ. N) THEN
C*            IRI2 = NX
C*            ILE2 = NX
C*          ELSE
              DL = L2 * SLOPE
C*            T = FLOAT (L2) / N
C*            DEVIAT = DELTA * SQRT (T * (ONE - T))
C
C  X axis boundaries of the subset of `line' L(l+1) within
C  ICATth rectangle
C
              IRI2 = MIN0 (INT (DL + DEVIAT), L2, NX)
              ILE2 = MAX0 (INT (DL - DEVIAT + ONE),  L2 - NY, 0)
C*          ENDIF
C
C  Four sides of the rectangle on the X and Y axes
C
            ILEFT = ILE
            IRIGHT = IRI2
            JUPP = L2 - ILE2
            JLOW = L - IRI - 1
          ENDIF
C
C  Calculate boundaries for current L (Lth value is tied)
C
          ILE = MAX0 (ILEFT, L - JUPP)
          IRI = MIN0 (IRIGHT, L - JLOW)
        ENDIF
C
C  Set the left (right) boundary for the one-sided test
C
        GOTO (30, 20, 10) KIND
   10   IRI = MIN0 (NX, L)
        GOTO 30
   20   ILE = MAX0 (0, L - NY)
C
C  Calculate the number of trajectories p(i,j) for current L
C
   30   ILES = MAX0 (1, ILE)
        IRIS = MIN0 (L - 1, IRI)

        DO 50 I = IRIS, ILES, - 1
```

```
   50   P(I + 1) = P(I + 1) + P(I)
C
C  Check whether elements of P are large enough to multiply
C  them by SMALL
C
        IC = IC - 1
        IF (IC .LE. 0) THEN
          DL = ZERO
          DO 60 I = ILES + 1, IRIS + 1
   60     DL = MAX (P(I), DL)
          IF (DL .EQ. ZERO) RETURN
          IF (DL .GT. CHKNUM) THEN
            DO 65 I = ILES + 1, IRIS + 1
   65       P(I) = P(I) * SMALL
            IC = ITERUP
            NOFDIV = NOFDIV + 1
            SCL = SCL * SMALL
          ELSE
C
C  Estimate the number of iterations for DL=Pmax to became
C  of order 1/SMALL
C
            IC = (- SMALLN - LOG (DL)) / ALN2
          END IF
        END IF
C
C  Define, whether boundaries lie on the left and lower sides
C  of the rectangle R and define boundary values for the next
C  iterations
C
        IF (ILE .EQ. 0) THEN
          P(ILES) = SCL
        ELSE
          P(ILES) = ZERO
        ENDIF

        IF (IRI .EQ. L) THEN
          P(IRIS + 2) = SCL
        ELSE
          P(IRIS + 2) = ZERO
        ENDIF
C
  100 CONTINUE
C
      DL = P(NX + 1) + P(NX)
      IF (DL .EQ. ZERO) RETURN
C
C  The P-value
C
      Q= ONE - EXP (FL(NX) + FL(NY) + LOG(DL) - NOFDIV * SMALLN - FL(N))
C
C  Q=1 is allowable, Q<=0 is not (accuracy loss due to
C  rounding errors)
C
      IF (Q .LE. ZERO) IFAULT = 3
      END
```

```
      SUBROUTINE CNV2X2(IK, MXS, MXZ, MXD, LGE, ITAB, HYP, DS, II, K1,
     *                  K2, IERR)
C
C        ALGORITHM AS 289.1 APPL.STATIST. (1994), VOL.43, NO.1
C
C        Convolves hypergeometric distributions generated by
C        several 2x2 tables
C
      INTEGER IK, MXS, MXZ, MXD, ITAB(IK,4), II, K1, K2, IERR
      REAL LGE, HYP(0:MXZ), DS(0:1,0:MXD)
C
      INTEGER I, IAM, IH1, IH2, ILS, IL1, IL2, IMM, INN, IR, ITT, IXX,
     *        J, JJ, JR, KK
      REAL AM, DD1, DD2, DSMX, EL, HYMAX, ONE, SUMLG, X, ZERO, ZEXP,
     *     ZLOG
C
      DATA ONE, ZERO / 1.0E + 00, 0.0E + 00 /
C
      EXTERNAL SUMLG
C
      ZEXP(X) = EXP(X)
      ZLOG(X) = LOG(X)
C
      IERR = 0
C
C        Check input parameters
C
      IF (IK .GT. MXS) IERR = 1
      K1 = 0
      DO 10 I = 1, IK
         IMM = ITAB(I, 1) + ITAB(I, 2)
         INN = ITAB(I, 3) + ITAB(I, 4)
         ITT = ITAB(I, 1) + ITAB(I, 3)
C
C        Lower and upper limits for stratum distribution
C
         IF (ITT .GT. INN) THEN
            IL1 = ITT - INN
         ELSE
            IL1 = 0
         ENDIF
         IF (ITT .LT. IMM) THEN
            IL2 = ITT
         ELSE
            IL2 = IMM
         ENDIF
         ITT = IL2 - IL1
         IF (ITT .GT. MXZ) IERR = 2
         K1 = K1 + IL1
   10 CONTINUE
      IF (IERR .GT. 0) RETURN
C
      DD1 = 10 * ONE
      EL  = LGE * ZLOG(DD1)
C
C        Initialize and set log-scale indicator
C
      II = 0
      JJ = 1
      IR = 0
      DS(0, 0) = ONE / ZEXP(EL)
```

```
      DSMX = ZERO - EL
      ILS = 0
C
C        For strata = 1, ..., IK, compute hypergeometric distribution
C        and perform convolution in a recursive fashion
C
      DO 130 I = 1, IK
         IMM = ITAB(I, 1) + ITAB(I, 2)
         INN = ITAB(I, 3) + ITAB(I, 4)
         ITT = ITAB(I, 1) + ITAB(I, 3)
C
C        Lower and upper limits for convolution
C
         IF (ITT .GT. INN) THEN
            IL1 = ITT - INN
         ELSE
            IL1 = 0
         ENDIF
         IF (ITT .LT. IMM) THEN
            IL2 = ITT
         ELSE
            IL2 = IMM
         ENDIF
         IL2 = IL2 - IL1
         K2 = IR + IL2
         IF (K2 .GT. MXD) THEN
            IERR = 3
            RETURN
         ENDIF
C
C        Compute stratum distribution on log-scale
C
         HYP(0) = ZERO
         DO 20 J = 1, IL2
            DD1 = (FLOAT(IMM-J-IL1+1)) * (FLOAT(ITT-J-IL1+1))
            DD2 = (FLOAT(J+IL1)) * (FLOAT(INN-ITT+J+IL1))
            HYP(J) = HYP(J - 1) + ZLOG(DD1 / DD2)
   20    CONTINUE
         IF (ILS .EQ. 1) GOTO 90
C
C        Get maximum hypergeometric coefficient on log-scale and
C        check for potential overflow in stratum distribution
C
         AM = (1.0 + FLOAT(ITT)) / (1.0 + FLOAT(INN + 1)/FLOAT(IMM + 1))
         IAM = IFIX(AM) - IL1
         IF (HYP(0) .GT. HYP(IL2)) THEN
            DO 30 J = 0, IL2
               HYP(J) = HYP(J) - HYP(IL2)
   30       CONTINUE
         ENDIF
         HYMAX = HYP(IAM)
         IF (HYMAX .GT. EL) THEN
            ILS = 1
            GOTO 70
         ENDIF
C
C        Check for potential overflow in the ith convolution
C
         IF (IL2 .LT. IR) THEN
            IXX = IL2 + 1
         ELSE
```

```
               IXX = IR + 1
            ENDIF
            DD1 = IXX
            DD1 = ZLOG(DD1)
            DSMX = DSMX + HYMAX + DD1
            IF (DSMX .GE. (EL - ONE)) THEN
               ILS = 1
               GOTO 70
            ENDIF
C
C           Convert stratum distribution to natural scale
C
            DO 40 J = 0, IL2
               HYP(J) = ZEXP(HYP(J))
   40       CONTINUE
C
C           Perform convolution on natural scale
C
            DSMX = ZERO - EL
            DO 60 J = 0, K2
               IF (J .GT. IL2) THEN
                  IH1 = J - IL2
               ELSE
                  IH1 = 0
               ENDIF
               IF (J .LT. IR) THEN
                  IH2 = J
               ELSE
                  IH2 = IR
               ENDIF
               DS(JJ, J) = DS(II, IH1) * HYP(J - IH1)
               DO 50 JR = IH1 + 1, IH2
                  DD1 = DS(II, JR) * HYP(J - JR)
                  DS(JJ, J) = DS(JJ, J) + DD1
   50          CONTINUE
               DD1 = ZLOG(DS(JJ, J))
               IF (DD1 .GT. DSMX) DSMX = DD1
   60       CONTINUE
            GOTO 120
   70       CONTINUE
C
C           Convert (i-1)th convolved distribution to log-scale
C
            DO 80 KK = 0, IR
               DS(II, KK) = EL + ZLOG(DS(II, KK))
   80       CONTINUE
C
C           Perform convolution on logarithmic scale
C
   90       CONTINUE
            DO 110 J = 0, K2
               IF (J .GT. IL2) THEN
                  IH1 = J - IL2
               ELSE
                  IH1 = 0
               ENDIF
               IF (J .LT. IR) THEN
                  IH2 = J
               ELSE
                  IH2 = IR
               ENDIF
```

```
               DS(JJ, J) = DS(II, IH1) + HYP(J - IH1)
               DO 100 JR = IH1 + 1, IH2
                  DD1 = DS(II, JR) + HYP(J - JR)
                  DD2 = DS(JJ, J)
                  DS(JJ, J) = SUMLG(DD1, DD2)
  100          CONTINUE
  110       CONTINUE
C
C           Reset for next step
C
  120       II = JJ
            JJ = 1 - II
            IR = K2
C
  130 CONTINUE
C
C           Normalize final distribution
C
          IF (ILS .NE. 1) THEN
             DO 140 I = 0, K2
                DS(II, I) = ZLOG(DS(II, I))
  140        CONTINUE
          ENDIF
          DSMX = DS(II, 0)
          DO 150 I = 1, K2
             DD1 = DS(II, I)
             DD2 = SUMLG(DSMX, DD1)
             DSMX = DD2
  150     CONTINUE
          DO 160 I=0, K2
             DS(II, I) = DS(II, I) - DSMX
  160     CONTINUE
          K2 = K1 + K2
          RETURN
          END
C
          REAL FUNCTION SUMLG(DD1, DD2)
C
C           ALGORITHM AS 289.2 APPL.STATIST. (1994), VOL.43, NO.1
C
C           Adds two logarithmic scale numbers
C
          REAL DD1, DD2
C
          REAL DDD, X, ZEXP, ZLOG
C
          ZEXP(X) = EXP(X)
          ZLOG(X) = LOG(X)
C
          DDD = DD1
          IF (DD2 .GT. DD1) DDD = DD2
          DD1 = ZEXP(DD1 - DDD)
          DD2 = ZEXP(DD2 - DDD)
          DDD = ZLOG(DD1 + DD2) + DDD
          SUMLG = DDD
          RETURN
          END
```

```
      SUBROUTINE HALPRN(PARAM, NP, NCOLS, NOBS, NP1, NP2, P1LO, P1HI,
     +         P2LO, P2HI, CONF, NCONF, F, GRID, DIM1, NGRID1, NGRID2,
     +         DERIV, WK, DIMWK, LINDEP, IFAULT)
C
C     ALGORITHM AS290  APPL. STATIST. (1994) VOL. 43, NO. 1
C
C     Generate a rectangular 2-D grid of variance ratios from which
C     to plot confidence regions for two parameters using Halperin's
C     method.   If the model is linear in all parameters other than
C     the two selected, the confidence regions are exact; otherwise
C     they are approximate and the user should test the sensitivity of
C     the confidence regions to variation in the other parameters.
C
C     Auxiliary routines required: CLEAR, INCLUD, SS, TOLSET and SING,
C     from AS 274 and a routine DERIV supplied by the user to calculate
C     first derivatives.
C
C Arguments:
C PARAM(NP)  Parameter values
C NP         Number of parameters
C NCOLS      Number of first derivatives, usually the same as NP
C NOBS       Number of observations (needed to calculate deg. of freedom
C NP1, NP2   The numbers (positions) of the two parameters for which
C            the confidence regions are needed
C P1LO, P1HI Upper and lower limits for parameter number NP1
C P2LO, P2HI Same for the other parameter
C CONF(NCONF) The confidence levels in percent
C NCONF      The number of confidence levels
C F          (Output) F ratios corresponding to the confidence levels
C GRID(DIM1, NGRID2) (Output) 2-D array of variance ratios on a regular
C            grid of values of the two parameters
C DIM1       1st dimension of array GRID in the calling program
C NGRID1, NGRID2 The numbers of grid points required
C DERIV      The name of the user's routine to calculate 1st derivatives
C WK(DIMWK)  Workspace
C DIMWK      Must be at least NCOLS(NCOLS + 9)/2
C LINDEP(NCOLS) (Output) Logical array.   LINDEP(I) = .TRUE. if the Ith
C            column of derivatives is linearly related to previous columns
C IFAULT     (Output) Error indicator
C            = 0 if no errors detected
C            = 1 if NCOLS < 2
C            = 2 if NOBS < NCOLS
C            = 4 if NP1 outside the range 1 to NP
C            = 8 ditto for NP2
C            = 16 if NP1 = NP2
C            = 32 if NGRID1 < 2 or NGRID2 < 2 or NGRID1 > DIM1
C            = 64 if DIMWK is too small
C            = -I if the Ith confidence level is not between 0 and 100
C N.B. IFAULT may equal a sum of some of the above values if there is more
C      than one error.
C
      INTEGER NP, NCOLS, NOBS, NP1, NP2, NCONF, DIM1, NGRID1, NGRID2,
     +          DIMWK, IFAULT
      REAL PARAM(NP), P1LO, P1HI, P2LO, P2HI, CONF(*), F(NCONF),
     +          GRID(DIM1, NGRID2), WK(DIMWK)
      LOGICAL LINDEP(NCOLS)
      EXTERNAL DERIV
C
C     Local variables
C
      INTEGER I, NDF, NRBAR, XPTR, DPTR, THPTR, RSSPTR, TOLPTR, I1, I2,
```

```
     +        IOBS
      REAL ZERO, ONE, HUNDRD, HALF, TWO, Q, RESID, SSERR, STEP1, STEP2,
     +          WT, P1SAVE, P2SAVE, SSQ1
      DATA ZERO/0.0/, ONE/1.0/, HUNDRD/100.0/, HALF/0.5/, TWO/2.0/,
     +          WT/1.0/
C
C     Some checks
C
      IFAULT = 0
      IF (NCOLS .LT. 2) IFAULT = 1
      IF (NOBS .LE. NCOLS) IFAULT = IFAULT + 2
      IF (NP1 .LT. 1 .OR. NP1 .GT. NP) IFAULT = IFAULT + 4
      IF (NP2 .LT. 1 .OR. NP2 .GT. NP) IFAULT = IFAULT + 8
      IF (NP1 .EQ. NP2) IFAULT = IFAULT + 16
      IF (NGRID1 .LT. 2 .OR. NGRID2 .LT. 2 .OR. NGRID1 .GT. DIM1)
     +          IFAULT = IFAULT + 32
      IF (IFAULT .GT. 0) RETURN
C
C     Set up pointers to the workspace.
C
      NRBAR = NCOLS * (NCOLS - 1) / 2
      XPTR = NRBAR + 1
      DPTR = XPTR + NCOLS
      THPTR = DPTR + NCOLS
      RSSPTR = THPTR + NCOLS
      TOLPTR = RSSPTR + NCOLS
      IF (TOLPTR + NCOLS - 1 .GT. DIMWK) THEN
        IFAULT = 64
        RETURN
      END IF
C
C     Calculate step sizes.
C
      STEP1 = (P1HI - P1LO) / (NGRID1 - 1)
      STEP2 = (P2HI - P2LO) / (NGRID2 - 1)
      P1SAVE = PARAM(NP1)
      P2SAVE = PARAM(NP2)
C-----------------------------------------------------------------------
C
C     Start of cycle through the grid.
C
      PARAM(NP1) = P1LO
      DO 30 I1 = 1, NGRID1
        PARAM(NP2) = P2LO
        DO 20 I2 = 1, NGRID2
C
C     Initialize orthogonal reduction.
C
          CALL CLEAR(NCOLS, NRBAR, WK(DPTR), WK, WK(THPTR), SSERR,
     *               IFAULT)
          DO 10 IOBS = 1, NOBS
            CALL DERIV(PARAM, NP, NCOLS, IOBS, WK(XPTR), RESID, WT)
C
C     Re-order the derivatives so that those for parameters NP1 & NP2
C     are at the end.
C
            IF (NP1 .LT. NCOLS-1) THEN
              IF (NP2 .NE. NCOLS-1) THEN
                Q = WK(XPTR + NCOLS - 2)
                WK(XPTR + NCOLS - 2) = WK(XPTR + NP1 - 1)
                WK(XPTR + NP1 - 1) = Q
```

```
                ELSE
                  Q = WK(XPTR + NCOLS - 1)
                  WK(XPTR + NCOLS - 1) = WK(XPTR + NP1 - 1)
                  WK(XPTR + NP1 - 1) = Q
                END IF
              END IF
              IF (NP2 .LT. NCOLS-1) THEN
                Q = WK(XPTR + NCOLS - 1)
                WK(XPTR + NCOLS - 1) = WK(XPTR + NP2 - 1)
                WK(XPTR + NP2 - 1) = Q
              END IF
C
C     Update QR factorization.
C
              CALL INCLUD(NCOLS, NRBAR, WT, WK(XPTR), RESID, WK(DPTR), WK,
     +              WK(THPTR), SSERR, IFAULT)
   10     CONTINUE
C
C-----------------------------------------------------------------------
C
C     Test for singularities
C
      CALL TOLSET(NCOLS, NRBAR, WK(DPTR), WK, WK(TOLPTR), WK(XPTR),
     +            IFAULT)
      CALL SING(NCOLS, NRBAR, WK(DPTR), WK, WK(THPTR), SSERR,
     +          WK(TOLPTR), LINDEP, WK(XPTR), IFAULT)
C
C     If IFAULT < 0, it contains the rank deficiency.
C
      NDF = NOBS - NCOLS - IFAULT
C
C     Calculate variance ratio.
C
          CALL SS(NCOLS, WK(DPTR), WK(THPTR), SSERR, WK(RSSPTR), IFAULT)
          IF (NCOLS .GT. 2) THEN
            SSQ1 = WK(RSSPTR + NCOLS - 3)
          ELSE
            SSQ1 = WK(RSSPTR) + WK(DPTR)*WK(THPTR)**2
          END IF
          GRID(I1, I2) = HALF * (SSQ1 - SSERR) / (SSERR / NDF)
          PARAM(NP2) = PARAM(NP2) + STEP2
   20   CONTINUE
        PARAM(NP1) = PARAM(NP1) + STEP1
   30 CONTINUE
C
      PARAM(NP1) = P1SAVE
      PARAM(NP2) = P2SAVE
C
C     Convert confidence levels in % to F-values, if NCONF > 0.
C
      DO 40 I = 1, NCONF
        Q = ONE - CONF(I) / HUNDRD
        IF (Q .LE. ZERO .OR. Q .GT. ONE) THEN
          IFAULT = -I
          RETURN
        END IF
        F(I) = HALF * NDF * (Q**(-TWO/NDF) - ONE)
   40 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE OCCALC(SSEQ, L, MAXL, OC)
C
C        ALGORITHM AS 291.1 APPL.STATIST. (1994), VOL.43, NO.2
C
C        Calculates overlap capability of subsequence SSEQ of length L
C        OC(I) = 1 if SSEQ can overlap its own last I letters
C                  (I = 1, ... , L)
C              = 0 otherwise
C
      INTEGER L, MAXL, OC(MAXL), SSEQ(MAXL)
C
      INTEGER I, INDEX, LM1, LMI
C
      OC(L) = 1
      LM1 = L - 1
      DO 20 I = 1, LM1
         LMI = L - I
         OC(I) = 0
         DO 10 INDEX = 1, I
            IF (SSEQ(INDEX) .EQ. SSEQ(INDEX + LMI)) GO TO 10
            GO TO 20
   10    CONTINUE
         OC(I) = 1
   20 CONTINUE
      RETURN
      END
C
      SUBROUTINE EXPVAR(M, L, MAXL, OC, P, EXPECT, VARIAN, IFAULT)
C
C        ALGORITHM AS 291.2 APPL.STATIST. (1994), VOL.43, NO.2
C
C        Compute expected value and variance of frequency of
C        occurrence
C
      INTEGER M, L, MAXL, OC(MAXL), IFAULT
      REAL P(MAXL), EXPECT, VARIAN
C
      INTEGER I, INDEX, LM1, NN
      REAL TERM
C
      IFAULT = 1
      IF (M .LT. L .OR. L .LT. 2 .OR. L .GT. MAXL) RETURN
      IFAULT = 0
      NN = M - L + 1
      EXPECT = NN * P(L)
      VARIAN = EXPECT * (1.0 - EXPECT)
      IF ((NN - L) .GT. 0) VARIAN = VARIAN +
     *   ((P(L) ** 2) * (NN - L) * (NN - L + 1))
      TERM = 0.0
      LM1 = L - 1
      DO 10 I = 1, LM1
         IF ((NN - I) .LE. 0) GO TO 10
         INDEX = L - I
         IF (OC(INDEX) .EQ. 0) GO TO 10
         TERM = TERM + OC(INDEX) * (NN - I) * P(I)
   10 CONTINUE
      TERM = 2.0 * P(L) * TERM
      VARIAN = VARIAN + TERM
      RETURN
      END
C
```

```
      SUBROUTINE PROW(M, N, MAXNP1, L, MAXL, OC, INDPP, C, POWER, PP,
     *                ROWNEW, IFAULT)
C
C        ALGORITHM AS 291.3 APPL.STATIST. (1994), VOL.43, NO.2
C
C        Calculate prob(J occurrences of a subsequence of length L
C                       within a sequence of length M) for J = 0, ... , N.
C        Prob(J occurrences) is stored in ROWNEW(J + 1)
C
      INTEGER M, N, MAXNP1, L, MAXL, OC(MAXL), INDPP(MAXL), IFAULT
      REAL C(MAXL, 2), POWER(MAXL), PP(MAXL, MAXNP1),
     *     ROWNEW(MAXNP1)
C
      INTEGER I, INDOLD, IP1, JP1, LM1, NP1, SEQLEN
C
C        Calculate IFAULT
C
      IFAULT = 0
      NP1 = N + 1
      IF (M .GE. L .AND. N .GE. 0 .AND. N .LE. M - L + 1
     *  .AND. NP1 .LE. MAXNP1 .AND. L .GE. 2 .AND. L .LE. MAXL) GO TO 10
      IFAULT = 1
      GO TO 80
C
C        Initialize PP using boundary conditions
C        (PP = previous L rows of probabilities needed to calculate
C              ROWNEW recursively)
C
   10 DO 30 IP1 = 1, L
         PP(IP1, 1) = 1.0
         IF (N .LT. 1) GO TO 30
         DO 20 JP1 = 2, NP1
            PP(IP1, JP1) = 0.0
   20    CONTINUE
   30 CONTINUE
C
C        Initialize INDPP
C        (INDPP = address in PP of rows 1 to L of P)
C
      DO 40 I = 1, L
         INDPP(I) = I
   40 CONTINUE
C
C        Loop to generate new rows of PP recursively
C
      DO 70 SEQLEN = L, M
C
C        Generate next row of probabilities
C
         CALL ROWGEN(N, L, MAXL, OC, MAXNP1, PP, INDPP, C,
     *               POWER, ROWNEW)
         IF (SEQLEN .EQ. M) GO TO 80
C
C        Shift rows of PP up one, overwriting first row
C        (effectively, using indirect addresses)
C
         LM1 = L - 1
         INDOLD = INDPP(1)
         DO 50 I = 1, LM1
            INDPP(I) = INDPP(I + 1)
   50    CONTINUE
```

```
            INDPP(L) = INDOLD
C
C           Move ROWNEW into bottom row of PP
C           (effectively, using indirect addresses)
C
            DO 60 JP1 = 1, NP1
               PP(INDOLD, JP1) = ROWNEW(JP1)
   60       CONTINUE
   70    CONTINUE
   80    RETURN
         END
C
         SUBROUTINE ROWGEN (N, L, MAXL, OC, MAXNP1, PP,
        *                   INDPP, C, POWER, ROWNEW)
C
C           ALGORITHM AS 291.4 APPL.STATIST. (1994), VOL.43, NO.2
C
C           Used by SUBROUTINE PROW to calculate one row of values:
C              prob(J occurrences of subsequence of length L)
C                   for J = 0, ... , N.
C           Prob(J occurrences) is stored in ROWNEW(J + 1)
C
         INTEGER N, L, MAXL, OC(MAXL), MAXNP1, INDPP(MAXL)
         REAL PP(MAXL, MAXNP1), C(MAXL, 2), POWER(MAXL),
        *     ROWNEW(MAXNP1)
C
         INTEGER I, INDEX, J, JNDEX, JP1, K, LIMJ, LM1, LM2, NP1
         REAL Q, Q1
C
         NP1 = N + 1
         LM1 = L - 1
         LM2 = L - 2
         DO 10 I = 1, NP1
            ROWNEW(I) = 0.0
   10    CONTINUE
C
C           Define C and POWER:
C              - C is coefficients in recursive formula;
C              - POWER is powers in recursive formula
C
         DO 20 I = 1, L
            POWER(I) = 4 ** I
   20    CONTINUE
         Q = OC(L - 1)
         C(1, 1) = (4.0 - Q) / POWER(1)
         C(1, 2) = Q / POWER(1)
         Q = OC(1)
         C(L, 1) = (4.0 * Q - 1.0) / POWER(L)
         C(L, 2) = -C(L, 1)
         IF (L .EQ. 2) GO TO 40
         DO 30 K = 1, LM2
            Q = OC(K)
            Q1 = OC(K + 1)
            C(L - K, 1) = ( - Q + 4.0 * Q1) / POWER(L - K)
            C(L - K, 2) = -C(L - K, 1)
   30    CONTINUE
C
C           Compute new probabilities from old ones
C
   40    DO 70 JP1 = 1, NP1
            ROWNEW(JP1) = 0.0
```

```
      LIMJ = 2
      IF (JP1 .EQ. 1) LIMJ = 1
      DO 60 J = 1, LIMJ
         DO 50 I = 1, L
            INDEX = INDPP(L - I + 1)
            JNDEX = JP1 - J + 1
            ROWNEW(JP1) = ROWNEW(JP1) + C(I, J) * PP(INDEX, JNDEX)
50       CONTINUE
60    CONTINUE
70 CONTINUE
   RETURN
   END
```

```
      SUBROUTINE LSINF(IDIST, ITYPE, ZL, ZR, F11, F12, F22, IFAULT)
C
C        ALGORITHM AS 292.1 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Computes Fisher information matrix elements for time (type I)
C        or failure (type II) censored units from the smallest extreme
C        value (sev), largest extreme value (lev), normal or logistic
C        distribution
C
      INTEGER IDIST, ITYPE, IFAULT
      REAL ZL, ZR, F11, F12, F22
C
      INTEGER NDIST
C
      PARAMETER (NDIST = 4)
C
      REAL BIG, BOUNDS(2, NDIST), CDF, DEN, ETA, PDF, SUR, THET0L,
     *     THET1L, THET2L, THET0R, THET1R, THET2R, ZERO, ZMAX, ZMIN,
     *     ZU
C
      DATA BOUNDS / 3.6E + 00, -34.0E + 00,
     *             34.0E + 00,  -3.6E + 00,
     *              8.0E + 00,  -8.0E + 00,
     *             34.0E + 00, -34.0E + 00 /
C
      PARAMETER (BIG = 1.0E + 10, ZERO = 0.0E + 00)
C
      EXTERNAL LSINT
C
C        Check for illegal idist
C
      IFAULT = 1
      IF (IDIST .LT. 1 .OR. IDIST .GT. NDIST) RETURN
C
C        Check for illegal itype
C
      IFAULT = 2
      IF (ITYPE .LT. 1 .OR. ITYPE .GT. 4) RETURN
C
C        Check for illegal arguments
C
      IFAULT = 3
      IF (ITYPE .EQ. 4 .AND. ZL .GT. ZR) RETURN
      IFAULT = 0
      ZMAX = BOUNDS(1, IDIST)
      ZMIN = BOUNDS(2, IDIST)
C
C        Identify the censoring type
C
      GOTO (10, 20, 30, 40), ITYPE
C
C        Uncensored case
C
   10 CALL LSINT(IDIST, ZMIN, ZMAX, BIG, F11, F12, F22, IFAULT)
      RETURN
C
C        Right censored at zr
C
   20 CALL LSINT(IDIST, ZMIN, ZMAX, ZR, F11, F12, F22, IFAULT)
      RETURN
C
```

```
C        Left censored at zl
C
   30 ZU = BIG
C
C        Left and right censored data
C
   40 IF (ITYPE .EQ. 4) ZU = ZR
      CALL LSINT(IDIST, ZMIN, ZMAX, ZU, THET0R, THET1R, THET2R, IFAULT)
      IF (IFAULT .EQ. 4 .OR. IFAULT .EQ. 5) RETURN
      CALL LSINT(IDIST, ZMIN, ZMAX, ZL, THET0L, THET1L, THET2L, IFAULT)
      IF (IFAULT .EQ. 4 .OR. IFAULT .EQ. 5) RETURN
C
      ETA = ZERO
      IF (ZL .GT. ZMIN .AND. ZL .LT. ZMAX) THEN
         CALL PCSFUN(IDIST, ZL, PDF, CDF, SUR)
         DEN = CDF * SUR
         IF (DEN .GT. ZERO) ETA = PDF * PDF / DEN
      ENDIF
C
C        Left or left and right censored data - Fisher matrix elements
C
      F11 = THET0R - THET0L + ETA
      F12 = THET1R - THET1L + ZL * ETA
      F22 = THET2R - THET2L + ZL * ZL * ETA
      RETURN
      END
C
      SUBROUTINE LSINT(IDIST, ZMIN, ZMAX, Z, THETA0, THETA1, THETA2,
     *                 IFAULT)
C
C        ALGORITHM AS 292.2 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Computes theta0, theta1, theta2 for the sev, lev, normal and
C        logistic distributions
C
      INTEGER IDIST, IFAULT
      REAL ZMIN, ZMAX, Z, THETA0, THETA1, THETA2
C
      INTEGER NDIST
      PARAMETER (NDIST = 4)
C
      INTEGER IDISEV, J, JMAXLO, JTERM, N
      REAL ASYM(3, NDIST), ANUZ, CDF, CONST0, DEN, DUM, EMABSZ, ETASEV,
     *     HALF, ONE, PDF, SUR, S3, S3OLD, THREE, TMP0, TMP1, TOL, TWO,
     *     T0SEV, T1SEV, T2SEV, WKSP(40), X1, X2, X3, X4, ZERO, ZSEV
C
      DATA ASYM /
     * 1.0E + 00, 0.4227843350984671E + 00, 0.1823680660852879E + 01,
     * 1.0E + 00,-0.4227843350984671E + 00, 0.1823680660852879E + 01,
     * 1.0E + 00, 0.0E + 00, 2.0E + 00,
     * 0.3333333333333333E + 00, 0.0E + 00, 0.1429956044565484 2E + 01 /
C
      PARAMETER (CONST0 = 0.1644934066848226E + 01, HALF = 0.5E + 00,
     *           IDISEV = 1, JMAXLO = 40, ONE = 1.0E + 00,
     *           THREE = 3.0E + 00, TOL = 1.0E - 11, TWO = 2.0E + 00,
     *           ZERO = 0.0E + 00)
C
      EXTERNAL INTEGR, PCSFUN
C
      IFAULT = 0
C
```

```
C        Routing computations
C
      IF (Z .LE. ZMIN) THEN
C
C        Integrals are negligible
C
         THETA0 = ZERO
         THETA1 = ZERO
         THETA2 = ZERO
      ELSE IF (Z .GE. ZMAX) THEN
C
C        Integrals take limiting values
C
         THETA0 = ASYM(1, IDIST)
         THETA1 = ASYM(2, IDIST)
         THETA2 = ASYM(3, IDIST)
      ELSE
C
C        Computed integrals - z is in interval (zmin, zmax)
C        First select the distribution
C
         GOTO (10, 20, 30, 40), IDIST
C
C        Smallest extreme value (sev) distribution
C
   10    CALL INTEGR(Z, THETA0, THETA1, THETA2, IFAULT)
         IF (IFAULT .NE. 0) IFAULT = 4
         RETURN
C
C        Largest extreme value (lev) distribution
C        First compute thetas and eta for the sev at -z
C
   20    ZSEV = -Z
         CALL INTEGR(ZSEV, T0SEV, T1SEV, T2SEV, IFAULT)
         IF (IFAULT .NE. 0) THEN
            IFAULT = 4
            RETURN
         ENDIF
         ETASEV = ZERO
         CALL PCSFUN(IDISEV, ZSEV, PDF, CDF, SUR)
         DEN = CDF * SUR
         IF (DEN .GT. ZERO) ETASEV = PDF * PDF / DEN
C
C        Compute thetas for the lev
C
         THETA0 = ONE - T0SEV + ETASEV
         THETA1 = ASYM(2, IDIST) + T1SEV + Z * ETASEV
         THETA2 = ASYM(3, IDIST) - T2SEV + Z * Z * ETASEV
         RETURN
C
C        Normal distribution
C
   30    CALL PCSFUN(IDIST, Z, PDF, CDF, SUR)
         THETA0 = CDF - Z * PDF
         IF (SUR .GT. ZERO) THETA0 = THETA0 + PDF * PDF / SUR
         THETA1 = Z * THETA0 - Z * CDF - PDF
         THETA2 = Z * THETA1 + TWO * CDF
         RETURN
C
C        Logistic distribution
C
```

```
   40    CALL PCSFUN(IDIST, Z, PDF, CDF, SUR)
         THETA0 = (ONE - SUR ** 3) / THREE
         TMP0 = LOG(ONE + EXP(Z))
         THETA1 = Z * THETA0 + (PDF - TMP0) / THREE
C
C        Compute theta2 for the logistic
C        S3 is computed using a power series expansion with
C        accelerated convergence using Euler transformation
C
         EMABSZ = EXP(-ABS(Z))
         X1 = EMABSZ
         S3 = ZERO
C
C        Euler partial sum based on the first term
C
         N = 1
         WKSP(1) = X1
         S3 = HALF * X1
         S3OLD = S3
         DO 60 JTERM = 2, JMAXLO
            X2 = REAL(JTERM) ** 2
            X3 = REAL(JTERM - 1) ** 2
            X1 = -X1 * EMABSZ * X3 / X2
C
C        Euler partial sum based on two or more terms
C
            TMP1 = WKSP(1)
            WKSP(1) = X1
            DO 50 J = 1, N
               IF (J .LT. N) DUM = WKSP(J + 1)
               WKSP(J + 1) = HALF * (WKSP(J) + TMP1)
               IF (J .LT. N) TMP1 = DUM
   50       CONTINUE
C
C        Euler improved partial sums
C
            IF (ABS(WKSP(N + 1)) .GT. ABS(WKSP(N))) THEN
               S3 = S3 + WKSP(N + 1)
            ELSE
               S3 = S3 + HALF * WKSP(N + 1)
               N = N + 1
            ENDIF
C
C        Tests for convergence of the series - a fault is declared if
C        convergence is not reached in a maximum of jmaxlo terms
C
            X4 = ABS(S3OLD - S3)
            S3OLD = S3
            IF (X4 .LT. TOL) THEN
C
C        Add terms to obtain the integral
C
               ANUZ = S3
               IF (Z .GT. ZERO) ANUZ = CONST0 + Z * Z / TWO - S3
               THETA2 = Z * THETA1
     *              + (CDF - Z * TMP0 + Z * PDF + TWO * ANUZ) / THREE
               RETURN
            ENDIF
   60    CONTINUE
         IFAULT = 5
      ENDIF
```

```
      RETURN
      END
C
      SUBROUTINE PCSFUN(IDIST, Z, PDF, CDF, SUR)
C
C        ALGORITHM AS 292.3 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Computes the p.d.f., c.d.f. and survival functions for the
C        sev, lev, normal and logistic distributions
C
      INTEGER IDIST
      REAL Z, PDF, CDF, SUR
C
      REAL ALNORM, CVAL, ENZ, EPZ, HALF, ONE
C
      PARAMETER (CVAL = 0.3989422804014326E + 00, HALF = 0.5E + 00,
     *           ONE = 1.0E + 00)
C
      EXTERNAL ALNORM
C
C        Select the distribution
C
      GOTO (10, 20, 30, 40), IDIST
C
C        sev
C
   10 EPZ = EXP(Z)
      PDF = EXP(Z - EPZ)
      CDF = ONE - EXP(-EPZ)
      SUR = EXP(-EPZ)
      RETURN
C
C        lev
C
   20 ENZ = EXP(-Z)
      PDF = EXP(-Z - ENZ)
      CDF = EXP(-ENZ)
      SUR = ONE - EXP(-ENZ)
      RETURN
C
C        Normal
C
   30 PDF = CVAL * EXP(-HALF * Z * Z)
      CDF = ALNORM(Z, .FALSE.)
      SUR = ALNORM(Z, .TRUE.)
      RETURN
C
C        Logistic
C
   40 ENZ = EXP(-Z)
      CDF = ONE / (ONE + ENZ)
      SUR = ENZ / (ONE + ENZ)
      PDF = CDF * SUR
      RETURN
C
      END
```

```
      SUBROUTINE CNV2XK(IK, K, MC, MXC, MXR, MS, MZ, MD, LGE, ITAB, ISC,
     *                  LBUF, CBUF, X, DS, LFACT, II, K1, K2, IERR)
C
C        ALGORITHM AS 293.1 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Convolves conditional distributions generated by
C        several 2xK tables
C
      INTEGER IK, K, MC, MXC, MXR, MS, MZ, MD, ITAB(MS, 2, MC), ISC,
     *        LBUF(0:1, 0:MXR, 2), X(MC, 0:2), II, K1, K2, IERR
      REAL LGE, CBUF(0:1, 0:MZ), DS(0:1, 0:MD), LFACT(MXC+1)
C
      INTEGER I, IAA, IH1, IH2, IKK, ILS, IMC, IMM, IMR, IXX, IZ1, IZ2,
     *        J, JJ, JR, KK
      REAL DD1, DD2, DSMX, EL, HYMAX, ONE, SUMLG, Y, ZERO, ZEXP, ZLOG
C
      DATA ONE, ZERO / 1.0E + 00, 0.0E + 00 /
C
      EXTERNAL SUMLG
C
      ZEXP(Y) = EXP(Y)
      ZLOG(Y) = ALOG(Y)
C
      IERR = 0
C
C        Check input parameters
C
      IF (IK .GT. MS) IERR = 1
      IF (K .GT. MC) IERR = 2
      IAA = 0
      DO 20 I = 1, IK
         IMM = 0
         IMR = 0
         DO 10 J = 1, K
            IMM = IMM + ITAB(I, 1, J) + ITAB(I, 2, J)
            IMR = IMR + ITAB(I, 2, J)
            IMC = ITAB(I, 1, J) + ITAB(I, 2, J)
            IF (IMC .GT. MXC) IERR = 3
            IAA = MAX0(IAA, IMC)
   10    CONTINUE
         IF (IMR .GT. MXR) IERR = 4
   20 CONTINUE
      IF (IERR .GT. 0) RETURN
C
      DD1 = 10 * ONE
      EL = LGE * ZLOG(DD1)
C
C        Compute log-factorials:
C           LFACT(I) = LOG( (I - 1)! )
C
      LFACT(1) = ZERO
      DO 30 I = 2, IAA + 1
         DD1 = I - 1
         LFACT(I) = LFACT(I - 1) + ZLOG(DD1)
   30 CONTINUE
C
C        Initialize and set scale indicator
C
      II = 0
      JJ = 1
      IR = 0
```

```
      K1 = 0
      DS(0, 0) = ONE / ZEXP(EL)
      DSMX = ZERO - EL
      ILS = 0
C
C        For strata = 1, ..., IK, compute sub-distributions and
C        perform convolution in a recursive fashion
C
      DO 160 I = 1, IK
C
C        Extract the ith stratum
C
         IAA = 0
         IKK = 0
         DO 40 J = 1, K
            X(J, 0) = ITAB(I, 1, J)
            X(J, 1) = ITAB(I, 2, J)
            IKK = IKK + X(J, 1)
            IAA = IKK + X(J, 0)
   40    CONTINUE
C
C        Log-scale check
C
         IF (ILS .EQ. 1) GOTO 110
C
C        Check for overflow in ith conditional distribution
C
         HYMAX = LFACT(IAA + 1) - LFACT(IKK + 1)
         HYMAX = HYMAX - LFACT(IAA - IKK + 1)
         IF (HYMAX .GT. EL) ILS = 1
         CALL EX2XK(K, MC, MXC, MXR, MZ, ILS, EL, X, CBUF, LBUF, LFACT,
     *              IP, ISC, IZ1, IZ2, IERR)
         IF (IERR .GT. 0) RETURN
C
C        Lower and upper limits for convolution
C
         K1 = K1 + IZ1
         IZ2 = IZ2 - IZ1
         K2  = IR + IZ2
         IF (K2 .GT. MD) THEN
            IERR = 5
            RETURN
         ENDIF
         IF (ILS .EQ. 1) GOTO 90
C
C        Check for potential overflow in ith convolution
C
         IF (IZ2 .LT. IR) THEN
            IXX = IZ2 + 1
         ELSE
            IXX = IR + 1
         ENDIF
         DD1 = IXX
         DD1 = ZLOG(DD1)
         DSMX = DSMX + HYMAX + DD1
         IF (DSMX .GE. (EL - ONE)) THEN
            ILS = 1
            GOTO 70
         ENDIF
C
C        Perform convolution on natural scale
```

```
C
         DSMX = ZERO - EL
         DO 60 J = 0, K2
            IF (J .GT. IZ2) THEN
               IH1 = J - IZ2
            ELSE
               IH1 = 0
            ENDIF
            IF (J .LT. IR) THEN
               IH2 = J
            ELSE
               IH2 = IR
            ENDIF
            DS(JJ, J) = DS(II, IH1) * CBUF(IP, J - IH1)
            DO 50 JR = IH1 + 1, IH2
               DD1 = DS(II, JR) * CBUF(IP, J - JR)
               DS(JJ, J) = DS(JJ, J) + DD1
   50       CONTINUE
            IF (DS(JJ, J) .GT. ZEXP(-EL)) THEN
               DD1 = ZLOG(DS(JJ, J))
            ELSE
               DD1 = -EL
            ENDIF
            IF (DD1 .GT. DSMX) DSMX = DD1
   60    CONTINUE
         GOTO 150
C
C        Convert ith stratum distribution to log-scale
C
   70    CONTINUE
         DO 80 KK = 0, IR
            CBUF(IP, KK) = ZLOG(CBUF(IP, KK))
   80    CONTINUE
C
C        Convert (i-1)th cumulated distribution to log-scale
C
   90    CONTINUE
         DO 100 KK = 0, IR
            DS(II, KK) = EL + ZLOG(DS(II, KK))
  100    CONTINUE
         DSMX = EL + DSMX
         GOTO 120
C
C        Perform convolution on logarithmic scale
C
  110    CONTINUE
         CALL EX2XK(K, MC, MXC, MXR, MZ, ILS, EL, X, CBUF, LBUF, LFACT,
     *              IP, ISC, IZ1, IZ2, IERR)
         IF (IERR .GT. 0) RETURN
         K1 = K1 + IZ1
         IZ2 = IZ2 - IZ1
         K2 = IR + IZ2
         IF (K2 .GT. MD) THEN
            IERR = 5
            RETURN
         ENDIF
  120    CONTINUE
         DO 140 J = 0, K2
            IF (J .GT. IZ2) THEN
               IH1 = J - IZ2
            ELSE
```

```
                IH1 = 0
              ENDIF
              IF (J .LT. IR) THEN
                IH2 = J
              ELSE
                IH2 = IR
              ENDIF
              DS(JJ, J) = DS(II, IH1) + CBUF(IP, J - IH1)
              DO 130 JR = IH1 + 1, IH2
                DD1 = DS(II, JR) + CBUF(IP, J - JR)
                DD2 = DS(JJ, J)
                DS(JJ, J) = SUMLG(DD1, DD2)
  130         CONTINUE
  140       CONTINUE
C
C           Reset for next step
C
  150       II = JJ
            JJ = 1 - II
            IR = K2
  160     CONTINUE
C
C           Normalize final distribution
C
          IF (ILS .NE. 1) THEN
            DO 170 I = 0, K2
              IF (DS(II, I) .EQ. ZERO) THEN
                DS(II, I) = -EL
              ELSE
                DS(II, I) = EL + ZLOG(DS(II, I))
              ENDIF
  170       CONTINUE
          ENDIF
          DSMX = DS(II, 0)
          DO 180 I = 1, K2
            DD1 = DS(II, I)
            IF (DD1 .EQ. -EL) GOTO 180
            DD2 = SUMLG(DSMX, DD1)
            DSMX = DD2
  180     CONTINUE
          DO 190 I = 0, K2
            IF (DS(II, I) .EQ. -EL) GOTO 190
            DS(II, I) = DS(II, I) - DSMX
  190     CONTINUE
          K2 = K1 + K2
          RETURN
          END
C
          SUBROUTINE EX2XK(K, MC, MXC, MXR, MZ, ILS, EL, X, CBUF, LBUF,
         *                 LFACT, IP, ISC, IZ1, IZ2, IERR)
C
C           ALGORITHM AS 293.2 APPL.STATIST. (1994), VOL.43, NO.3
C
C           Recursively generates the exact distribution for a single
C           ordered 2xK table using a bivariate shift algorithm
C
          INTEGER K, MC, MXC, MXR, MZ, ILS, X(MC, 0:2),
         *        LBUF(0:1, 0:MXR, 2), IP, ISC, IZ1, IZ2, IERR
          REAL EL, CBUF(0:1, 0:MZ), LFACT(MXC + 1)
C
          INTEGER I, IJ, ILL, ILT(0:1), IMI, IMN, IS, ISS, IS1, IS2,
```

```
     *        IST(0:1), IT, ITT, IT1, IT2, IXT, J, JP, LADR, LLL, NI,
     *        TO(0:1)
      REAL ADD, BNM, CCC, ONE, SL, SUM, SUMLG, Y, ZERO, ZEXP
C
      DATA ONE, ZERO / 1.0E + 00, 0.0E + 00 /
C
      EXTERNAL BNM, LADR, SUMLG
C
      ZEXP(Y) = EXP(Y)
C
      SL = ZERO - EL
C
C        Construct the data matrix ordered scores set in x(i,2)
C
      DO 10 I = 1, K
         X(I, 0) = X(I, 0) + X(I, 1)
         IF (ISC .EQ. 1) X(I, 2) = I - 1
         IF (ISC .EQ. 2) X(I, 2) = (I - 1) * (I - 1)
         IF (ISC .EQ. 3) X(I, 2) = 2 ** (I - 1)
   10 CONTINUE
C
C        to(j) : observed value of sufficient statistics
C
      DO 20 J = 0, 1
         TO(J) = 0
   20 CONTINUE
      DO 30 I = 1, K
         TO(0) = TO(0) + X(I, 1)
         TO(1) = TO(1) + X(I, 1) * X(I, 2)
   30 CONTINUE
C
C        Scale factor: ils
C        Set record for initial stage
C           jp - current record indicator
C           ip - past record indicator
C
      DO 40 J=0, MZ
         IF (ILS .EQ. 0) THEN
            CBUF(0, J) = ZERO
         ELSE
            CBUF(0, J) = SL
         ENDIF
   40 CONTINUE
      DO 50 J=0, MXR
         LBUF(0, J, 1) = 0
         LBUF(0, J, 2) = 0
         LBUF(1, J, 1) = 0
         LBUF(1, J, 2) = 0
   50 CONTINUE
C
      IMN = 0
      DO 60 I=1, K
         IMN = IMN + X(I, 0)
   60 CONTINUE
      IMI = 0
C
C        ij - current stage (ij <= k)
C        ip - previous stage indicator
C        jp - current stage indicator
C
      IJ = 0
```

```
      IP = 0
      JP = 1 - IP
      IS = 0
      IST(IP) = 0
      ILT(IP) = 0
C
C        Smallest t for s=is
C        Address of largest t with s=is
C
      LBUF(IP, IS, 1) = 0
      LBUF(IP, IS, 2) = 0
C
C        Store the initial distribution
C
      LLL = 0
      IF (ILS .EQ. 0) THEN
         CBUF(IP, LLL) = ONE
      ELSE
         CBUF(IP, LLL) = ZERO
      ENDIF
C
C        Perform the recursions for obtaining the conditional
C        distribution
C
   70 IJ = IJ + 1
C
C        Compute lower and upper bounds for s
C
      IMI = IMI + X(IJ, 0)
      IS1 = MAX0(0, IMI + TO(0) - IMN)
      IS2 = MIN0(TO(0), IMI)
C
C        ist(jp) - smallest value of s for current stage
C        ilt(jp) - largest value of s for current stage
C
      IST(JP) = IS1
      ILT(JP) = IS2
      DO 100 IS = IS1, IS2
C
C        Compute minimum (it1) and maximum (it2) of t given s and
C        set buffer limits for this stage
C
         CALL OPT(MC, IJ, IS, X, IT1, IT2)
         LBUF(JP, IS, 1) = IT1
         IXT = IT2 - IT1
         IF (IS .EQ. IS1) THEN
            LBUF(JP, IS, 2) = IXT
         ELSE
            LBUF(JP, IS, 2) = LBUF(JP, IS - 1, 2) + IXT + 1
         ENDIF
C
C        Perform recursion
C
         NI = X(IJ, 0)
         IZ1 = MAX0(0, IS - ILT(IP))
         IZ2 = MIN0(NI, IS - IST(IP))
         DO 90 IT = IT1, IT2
            IF (ILS .EQ. 0) THEN
               SUM = ZERO
            ELSE
               SUM = SL
```

```
               ENDIF
               DO 80 KK = IZ1, IZ2
                  ISS = IS - KK
                  ITT = IT - KK * X(IJ, 2)
                  ILL = LBUF(IP, ISS, 1)
                  IF (ITT .LT. ILL) GOTO 80
                  ILL = ILL + LBUF(IP, ISS, 2)
                  IF (ISS .GT. IST(IP)) THEN
                     ILL = ILL - LBUF(IP, ISS - 1, 2) - 1
                  ENDIF
                  IF (ITT .GT. ILL) GOTO 80
                  LLL = LADR(ISS, ITT, IP, LBUF, IST, MXR)
                  CCC = CBUF(IP, LLL)
                  IF (ILS. EQ. 0) THEN
                     IF (CCC .EQ. ZERO) GOTO 80
                  ELSE
                     IF (CCC .EQ. SL) GOTO 80
                  ENDIF
                  ADD = BNM(KK, NI, LFACT, MXC)
                  IF (ILS .EQ. O) THEN
                     ADD = ZEXP(ADD)
                     ADD = ADD * CCC
                     SUM = SUM + ADD
                  ELSE
                     ADD = ADD + CCC
                     SUM = SUMLG(SUM, ADD)
                  ENDIF
   80          CONTINUE
               LLL = LADR(IS, IT, JP, LBUF, IST, MXR)
               IF (LLL .GE. MZ) THEN
                  IERR = 6
                  RETURN
               ENDIF
               CBUF(JP, LLL) = SUM
   90       CONTINUE
  100    CONTINUE
C
C        Check if final stage is reached
C
         IF (IJ .GE. K) GOTO 110
C
C        Change parity and continue
C
         IP = JP
         JP = 1 - JP
         GOTO 70
  110    CONTINUE
C
C        Lower and upper support of stratum distribution
C
         IZ1 = LBUF(JP, TO(0), 1)
         IZ2 = IZ1 + LBUF(JP, TO(0), 2)
         IP = JP
         RETURN
         END
C
         SUBROUTINE OPT(MC, IJ, IS, X, IT1, IT2)
C
C        ALGORITHM AS 293.3 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Computes optimum values for t given s
```

```
C
      INTEGER MC, IJ, IS, X(MC, 0:2), IT1, IT2
C
      INTEGER I, IX, MM
C
C        Compute minimum
C
      IT1 = 0
      IX = IS
      DO 10 I = 1, IJ
         MM = MIN0(IX, X(I, 0))
         IT1 = IT1 + MM * X(I, 2)
         IF (IX .LE. X(I, 0)) THEN
            GOTO 20
         ELSE
            IX = IX - X(I, 0)
         ENDIF
   10 CONTINUE
   20 CONTINUE
      IT2 = 0
      IX = IS
      DO 30 I = IJ, 1, -1
         MM = MIN0(IX, X(I, 0))
         IT2 = IT2 + MM * X(I, 2)
         IF (IX .LE. X(I, 0)) THEN
            RETURN
         ELSE
            IX = IX - X(I, 0)
         ENDIF
   30 CONTINUE
      RETURN
      END
C
      INTEGER FUNCTION LADR(IS, IT, KP, LBUF, IST, MXR)
C
C        ALGORITHM AS 293.4 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Determines an address for (s,t) in the linear array cbuf
C
      INTEGER IS, IT, KP, LBUF(0:1, 0:MXR, 2), IST(0:1), MXR
C
      INTEGER IXT
C
      IXT = IT - LBUF(KP, IS, 1)
      IF (IS .EQ. IST(KP)) THEN
         LADR = IXT
      ELSE
         LADR = LBUF(KP, IS - 1, 2) + IXT + 1
      ENDIF
      RETURN
      END
C
      REAL FUNCTION BNM(K, N, LFACT, MXC)
C
C        ALGORITHM AS 293.5 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Computes binomial coefficient on log-scale
C
      INTEGER K, N, MXC
      REAL LFACT(MXC + 1)
C
```

```
      INTEGER L
      REAL DD
C
      L = N - K
      DD = LFACT(N + 1)
      DD = DD - LFACT(L + 1)
      DD = DD - LFACT(K + 1)
      BNM = DD
      RETURN
      END
C
      REAL FUNCTION SUMLG(D1, D2)
C
C        ALGORITHM AS 293.6 APPL.STATIST. (1994), VOL.43, NO.3
C
C        Adds two logarithmic scale numbers
C
      REAL D1, D2
C
      REAL DD, X, ZEXP, ZLOG
C
      ZEXP(X) = EXP(X)
      ZLOG(X) = ALOG(X)
C
      DD = D1
      IF (D2 .GT. D1) DD = D2
      D1 = ZEXP(D1 - DD)
      D2 = ZEXP(D2 - DD)
      DD = ZLOG(D1 + D2) + DD
      SUMLG = DD
      RETURN
      END
```

```
PROCEDURE Error(VAR Size: IntGC; MaxSize: IntGC; ErrorNumber: NonnegInt;
                AnotherError: Boolean; VAR Fault: NonnegInt);

{ALGORITHM AS 294.1 APPL.STATIST. (1994) VOL.43, N0.3}

BEGIN  IF (Size>MaxSize) OR AnotherError THEN
        BEGIN  Fault:=ErrorNumber;  Size:=MaxSize
        END
END; {of Error}

PROCEDURE Reduce(VAR NumNumerSet,NumDenomSet:              IntGC;
    VAR NumeratorSet,NumerSetOC,DenominatorSet,DenomSetOC: SetArr;
        AnObsConfig:                                       IntSet);

{ALGORITHM AS 294.2 APPL.STATIST. (1994) VOL.43, N0.3}

{Reduce the fraction: NumeratorSet/DenominatorSet}

VAR i,j,k,m: IntGC;  Reducible: Boolean;
BEGIN  k:=0;
 FOR i:=1 TO NumNumerSet DO
  BEGIN
   IF AnObsConfig>=NumeratorSet[i] THEN
    BEGIN  j:=1;
     WHILE (j<=NumDenomSet) AND (NOT(AnObsConfig>=DenominatorSet[j]) OR
           (NumerSetOC[i]<>DenomSetOC[j]) OR
           (NumeratorSet[i]<>DenominatorSet[j])) DO j:=j+1;
     Reducible:= j <= NumDenomSet;
     IF Reducible THEN
      BEGIN
       FOR m:=j+1 TO NumDenomSet DO
       BEGIN DenominatorSet[m-1]:=DenominatorSet[m];
        DenomSetOC[m-1]:=DenomSetOC[m]
       END;
       NumDenomSet:=NumDenomSet-1
      END
    END;
   IF NOT Reducible OR NOT(AnObsConfig>=NumeratorSet[i]) THEN
    BEGIN k:=k+1; NumeratorSet[k]:=NumeratorSet[i];
     NumerSetOC[k]:=NumerSetOC[i]
    END
  END; {of FOR i}
 NumNumerSet:=k
END; {of Reduce}

PROCEDURE MissMLE (SizeGeClass, NumObsConfig:               IntGC;
      GeClass, ObsConfig:                                   SetArr;
  VAR NumNumer, NumDenom,NumSubGC:                          IntGC;
  VAR Numerator,Denominator,NumerOC,DenomOC,SGClassOC:      SetArr;
  VAR SubGeClass:                                           SetArr2;
  VAR SizeSubGC:                                            IntArr;
  VAR VarSetModel:                                          IntSet;
  VAR Fault:                                                NonnegInt);

{ALGORITHM AS 294.3 APPL.STATIST. (1994) VOL.43, N0.3}

{Find the factor formula of MLEs}

VAR i,j,k,m,OrigNumNumer: IntGC;  Incomplete: Boolean;
    Variables,NumerConfig,DenomConfig,OCInModel,Block,BlockOC: IntSet;
```

```
PROCEDURE Nest(Block:IntSet; VAR BlockOC: IntSet);

{Factorizing by nested pattern of observations}

VAR j,m: NonnegInt; OCj,OCUnion,Union,Intersect: IntSet;
    Unchange,Separa: Boolean;
BEGIN
 IF BlockOC<>[] THEN
{Find the union of inside observing configurations, OCUnion}
 BEGIN  Intersect:=Block; Union:=[]; OCUnion:=[];
  REPEAT  Unchange:=True;
   FOR j:=1 TO NumObsConfig DO
    IF j IN BlockOC THEN
     BEGIN OCj:=ObsConfig[j]*Block;
      IF Intersect>=OCj THEN
       BEGIN  Intersect:=OCj; OCUnion:=OCj; Union:=[j]
       END
      ELSE
       BEGIN Intersect:=Intersect*OCj;
        IF NOT(j IN Union) AND (NOT(OCj>=OCUnion) OR (OCj=OCUnion)) THEN
         BEGIN  OCUnion:=OCUnion+OCj; Union:=Union+[j]; Unchange:=false
         END
       END
     END
  UNTIL  Unchange;
  IF NOT (OCUnion>=Block) THEN
   BEGIN
{Check whether OCUnion is contained in a generator}
    IF i<=OrigNumNumer THEN Separa:=True
    ELSE
     BEGIN  m:=i-OrigNumNumer; j:=0;
      REPEAT j:=j+1; Separa:=OCUnion<=SubGeClass[m,j]
      UNTIL Separa OR (j>=SizeSubGC[m])
     END;
    IF Separa THEN
{Factorize the MLEs by nested pattern}
     BEGIN  NumNumer:=NumNumer+1;
      Error(NumNumer,MaxGenerator,3,false,Fault);
      Numerator[NumNumer]:=OCUnion; NumerOC[NumNUmer]:=BlockOC;
      NumDenom:=NumDenom+1; Error(NumDenom,MaxGenerator,4,false,Fault);
      Denominator[NumDenom]:=OCUnion; DenomOC[NumDenom]:=BlockOC-Union;
      BlockOC:=BlockOC-Union; Nest(Block,BlockOC)
     END
   END {of IF NOT (OCUnion>=Block) THEN}
 END {of IF BlockOC<>[] THEN}
END; {of Nest}

{Check Whether the block A is separable}
FUNCTION Separable(OtherConditionHolds:Boolean;
   VarsInA,Generator:IntSet; VAR Bound,OCInModel:IntSet): Boolean;
VAR k: NonnegInt; Unchange,Separa: Boolean;
BEGIN
 IF NOT OtherConditionHolds THEN Separable:=false
 ELSE
  IF Bound=[] THEN
{For the cases that the blocks A and B do not intersect:}
   BEGIN  Separable:=True; NumerConfig:=OCInModel;
    FOR k:=1 TO NumObsConfig DO
     IF k IN OCInModel THEN
      BEGIN  IF ObsConfig[k]*(Variables-VarsInA)=[]
             THEN OCInModel:=OCInModel-[k];
```

```
        IF ObsConfig[k]*VarsInA=[] THEN NumerConfig:=NumerConfig-[k]
       END
    END
   ELSE
{For the cases that the blocks A and B intersect:}
    BEGIN
{Adjust the Bound according to the observing configurations}
     REPEAT   Unchange:=true;
      FOR k:=1 TO NumObsConfig DO
       IF k IN OCInModel THEN
        IF (ObsConfig[k]*(VarsInA-Bound)<>[]) AND
           (ObsConfig[k]*(Variables-VarsInA)<>[]) AND
           NOT (ObsConfig[k]>=Bound) THEN
         BEGIN  Bound:=Bound+ObsConfig[k]*VarsInA; Unchange:=false
         END
     UNTIL  Unchange;
     IF NOT (Generator>=Bound) OR (VarsInA-Bound=[]) THEN
      Separable:=false
     ELSE
      BEGIN
{Check whether the block A is separable as a submodel}
       k:=0;  NumerConfig:=[];  Separa:=true;
       REPEAT  k:=k+1;
        IF k IN OCInModel THEN
         BEGIN  Separa:=ObsConfig[k]*(VarsInA-Bound)=[];
          IF NOT Separa THEN
           BEGIN  Separa:= Bound<=ObsConfig[k];
            IF Separa THEN NumerConfig:=NumerConfig+[k]
           END
         END
       UNTIL (k>=NumObsConfig) OR NOT Separa;
       IF Separa THEN DenomConfig:=NumerConfig
       ELSE
{Check whether the block A is separable as a main model}
        BEGIN  k:=0;  Separa:=true;
         REPEAT  k:=k+1;
          IF k IN OCInModel THEN
           Separa:= (VarsInA>=ObsConfig[k]*Variables)
                 OR (Bound<=ObsConfig[k])
         UNTIL (k>=NumObsConfig) OR NOT Separa;
         IF Separa THEN
{Find sets of observing configurations of blocks A, B and denominator}
          BEGIN NumerConfig:=OCInModel; DenomConfig:=OCInModel;
           FOR k:=1 TO NumObsConfig DO
            IF (k IN OCInModel) AND(VarsInA>=ObsConfig[k]*Variables) THEN
             DenomConfig:=DenomConfig-[k];
           OCInModel:=DenomConfig
          END
        END; {of IF Separa ... ELSE}
       Separable:=Separa
      END {of IF NOT (Generator>=Bound)...ELSE}
    END {of IF Bound=[]...ELSE}
END; {of Separable}

{Check whether the generating class is decomposable or not}
PROCEDURE Deco(VAR SizeGeClass: IntGC; VAR GeClass: SetArr;
    VAR OCInModel:IntSet);
VAR i,j: IntGC;  Bound,OneClass: IntSet;
    GeClassUnchange,GeneratorUnchange,NoSubGenerator: Boolean;
BEGIN {of Deco}
 REPEAT  GeClassUnchange:=true;
```

```
  {Delete variables which exist in only a single generator}
    REPEAT  GeneratorUnchange:=true;
     FOR i:=1 TO SizeGeClass DO
      BEGIN  OneClass:=GeClass[i];
       FOR j:=1 TO SizeGeClass DO
        IF i<>j THEN OneClass:=OneClass-GeClass[j];
       Bound:=GeClass[i]-OneClass;
       IF Separable(OneClass<>[],GeClass[i],GeClass[i],Bound,OCInModel)
        THEN
        BEGIN Variables:=Variables-GeClass[i]+Bound;
         NumNumer:=NumNumer+1; Error(NumNumer,MaxGenerator,3,false,Fault);
         Numerator[NumNumer]:=GeClass[i]; NumerOC[NumNumer]:=NumerConfig;
         IF Bound<>[] THEN
          BEGIN NumDenom:=NumDenom+1;
           Error(NumDenom,MaxGenerator,4,false,Fault);
           Denominator[NumDenom]:=Bound; DenomOC[NumDenom]:=DenomConfig
          END;
         GeClass[i]:=Bound;  GeneratorUnchange:=false
        END
      END
    UNTIL GeneratorUnchange;
  {Delete generators which are contained in others}
    REPEAT  NoSubGenerator:=true;
     FOR i:=1 TO SizeGeClass DO
      FOR j:=1 TO SizeGeClass DO
       IF (i<>j) AND (GeClass[i]<=GeClass[j]) AND (GeClass[i]<>[]) THEN
        BEGIN GeClass[i]:=[];NoSubGenerator:=false;GeClassUnchange:=false
        END;
  {Delete the null generators}
     j:=0;
     FOR i:=1 TO SizeGeClass DO
      IF GeClass[i]<>[] THEN
       BEGIN j:=j+1;  GeClass[j]:=GeClass[i]
       END;
     SizeGeClass:=j
    UNTIL NoSubGenerator OR (SizeGeClass=0)
   UNTIL GeClassUnchange OR (SizeGeClass=0)
  END; {of Deco}

  {Decompose generating class}
  PROCEDURE DecompGC(VAR SizeGeClass: IntGC; VAR GeClass: SetArr;
      OCInModel: IntSet);
  VAR i,j,k,m,InAGeneratorsNum,InBGeneratorsNum: IntGC; GeClassA: SetArr;
      BoundaryOfAB,Bound,VarInA,VarInB,T: IntSet;
      Researched: ARRAY [1..MaxGenerator] OF Boolean;
      DecomposedOrCannotDecompose,NoVarAddedInA: Boolean;
  BEGIN  Variables:=[];
   FOR i:=1 TO SizeGeClass DO Variables:=Variables+GeClass[i];
   Deco(SizeGeClass,GeClass,OCInModel);
   IF SizeGeClass<>0 THEN
    BEGIN  i:=0;  DecomposedOrCannotDecompose:=false;
     REPEAT
  {Set [] or GeClass[i] as the separator, BoundaryOfAB}
      IF i=0 THEN BoundaryOfAB:=[] ELSE BoundaryOfAB:=GeClass[i];  m:=0;
      REPEAT  m:=m+1;
       IF (m<>i) AND NOT ((i=0) AND (m<>1))  THEN
        BEGIN  InAGeneratorsNum:=1; GeClassA[1]:=GeClass[m];
         Researched[m]:=true; VarInA:=GeClassA[1];
         FOR j:=1 TO SizeGeClass DO IF j<>m THEN Researched[j]:=false;
  {Find the set of variables in A, VarInA}
        REPEAT  NoVarAddedInA:=true;
```

```
         FOR j:=1 TO SizeGeClass DO
          IF (i<>j) AND NOT Researched[j] AND
             (VarInA*GeClass[j]-BoundaryOfAB<>[]) THEN
           BEGIN NoVarAddedInA:=false;  VarInA:=VarInA+GeClass[j];
            Researched[j]:=true; InAGeneratorsNum:=InAGeneratorsNum+1;
            GeClassA[InAGeneratorsNum]:=GeClass[j]
           END
        UNTIL NoVarAddedInA;
        IF i<>0 THEN InBGeneratorsNum:=SizeGeClass-InAGeneratorsNum-1
        ELSE  InBGeneratorsNum:=SizeGeClass-InAGeneratorsNum;
        VarInA:=VarInA+BoundaryOfAB;  VarInB:=[];
{Find the set of variables in B, VarInB}
        FOR k:=1 TO SizeGeClass DO
         IF (k<>i) AND NOT Researched[k] THEN VarInB:=VarInB+GeClass[k];
        Bound:=VarInA*VarInB;
        IF Separable(InBGeneratorsNum<>0,VarInA,BoundaryOfAB,
                     Bound,OCInModel) THEN
{The generating class can be decomposed by GeClass[i]}
         BEGIN  j:=0;  DecomposedOrCannotDecompose:=true;
          FOR k:=1 TO SizeGeClass DO
           IF (k<>i) AND NOT Researched[k] THEN
            BEGIN  j:=j+1; GeClass[j]:=GeClass[k]
            END;
          IF i<>0 THEN
           BEGIN  InAGeneratorsNum:=InAGeneratorsNum+1;
            GeClassA[InAGeneratorsNum]:=BoundaryOfAB;
            InBGeneratorsNum:=InBGeneratorsNum+1;
            GeClass[InBGeneratorsNum]:=Bound;
            NumDenom:=NumDenom+1;
            Error(NumDenom,MaxGenerator,4,false,Fault);
            Denominator[NumDenom]:=GeClass[InBGeneratorsNum];
            DenomOC[NumDenom]:=DenomConfig
           END;
{Decompose recursively A and B}
          DecompGC(InAGeneratorsNum,GeClassA,NumerConfig);
          DecompGC(InBGeneratorsNum,GeClass,OCInModel)
         END {of IF Separable(...) THEN}
       END {of IF (m<>i) AND NOT ((i=0) AND (m<>1)) THEN}
    UNTIL (m>=SizeGeClass) OR DecomposedOrCannotDecompose;
    IF NOT DecomposedOrCannotDecompose THEN
     BEGIN
{The generating class cannot be decomposed by GeClass[i]}
      IF i=SizeGeClass THEN
{The generating class cannot be decomposed by any generator}
       BEGIN  DecomposedOrCannotDecompose:=true; NumSubGC:=NumSubGC+1;
        Error(NumSubGC,MaxGenerator,5,false,Fault);
        SGClassOC[NumSubGC]:=OCInModel;SizeSubGC[NumSubGC]:=SizeGeClass;
        FOR k:=1 TO SizeGeClass DO SubGeClass[NumSubGC,k]:=GeClass[k]
       END
      ELSE i:=i+1
     END {of IF NOT Decomp... THEN}
   UNTIL DecomposedOrCannotDecompose
  END {of IF SizeGeClass<>0 THEN}
END; {of DecompGC}
BEGIN {of MissMLE}    Fault:=0;
 Error(SizeGeClass,MaxGenerator,1,SizeGeClass<1,Fault); VarSetModel:=[];
 IF (NumObsConfig<=0) OR (NumObsConfig>MaxGenerator) THEN Fault:=6;
 IF Fault=0 THEN
  FOR i:=1 TO SizeGeClass DO
   BEGIN
    IF NOT([1..MaxVar]>=GeClass[i]) OR (GeClass[i]=[]) THEN Fault:=2;
```

```
      VarSetModel:=VarSetModel+GeClass[i]
     END;
  IF Fault=0 THEN
   FOR i:=1 TO NumObsConfig DO
    IF NOT([1..MaxVar]>=ObsConfig[i])OR(ObsConfig[i]=[]) THEN Fault:=7;
  NumNumer:=0;  NumDenom:=0;  NumSubGC:=0;
  IF Fault=0 THEN
   BEGIN  OCInModel:=[1..NumObsConfig];
    DecompGC(SizeGeClass,GeClass,OCInModel);  OrigNumNumer:=NumNumer;
 {Factorize MLEs according to nested pattern}
    FOR i:=1 TO OrigNumNumer+NumSubGC DO
     BEGIN  IF i<=OrigNumNumer THEN
              BEGIN Block:=Numerator[i]; Nest(Block,NumerOC[i])
              END
      ELSE
       BEGIN  m:=i-OrigNumNumer;  Block:=[];
        FOR j:=1 TO SizeSubGC[m] DO Block:=Block+SubGeClass[m,j];
        Nest(Block,SGClassOC[m])
       END
     END;
    Reduce(NumNumer,NumDenom,Numerator,NumerOC,Denominator,DenomOC,
           VarSetModel);
 {Change the incomplete Numerator into SubGeClass}
    k:=0;
    FOR i:=1 TO NumNumer DO
     BEGIN  j:=1;
      Incomplete:=(j IN NumerOC[i]) AND NOT(ObsConfig[j]>=Numerator[i]);
      WHILE (j<=NumObsConfig) AND NOT Incomplete DO
       BEGIN  Incomplete:=(j IN NumerOC[i]) AND
                          NOT(ObsConfig[j]>=Numerator[i]);  j:=j+1
       END;
      IF Incomplete THEN
       BEGIN NumSubGC:=NumSubGC+1;
        Error(NumSubGC,MaxGenerator,5,false,Fault);
        SGClassOC[NumSubGC]:=NumerOC[i];  SizeSubGC[NumSubGC]:=1;
        SubGeClass[NumSubGC,1]:=Numerator[i]
       END
      ELSE
       BEGIN  k:=k+1;
        Numerator[k]:=Numerator[i]; NumerOC[k]:=NumerOC[i]
       END
     END; {of FOR i}
    NumNumer:=k
   END {of IF Fault=0 THEN}
 END; {of MissMLE}

 PROCEDURE MissHiMod(NumModels:                                    NonnegInt;
       M1SizeGeClass,M2SizeGeClass,NumObsConfig:                   IntGC;
       M1GeClass,M2GeClass,ObsConfig:                              SetArr;
   VAR LRNNumNumer,LRNNumDenom,LRNNumGeClass,
       LRDNumNumer,LRDNumDenom,LRDNumGeClass:                      IntArr;
   VAR LRNSizeGeClass,LRDSizeGeClass:                              IntArr2;
   VAR LRNNumer,LRNDenom,LRDNumer,LRDDenom,LRNOCNumer,LRNOCDenom,
       LRNOCGeClass,LRDOCNumer,LRDOCDenom,LRDOCGeClass:            SetArr2;
   VAR LRNGeClass,LRDGeClass:                                      SetArr3;
   VAR VariableSet:                                                IntSet;
   VAR Fault:                                                      NonnegInt);

 {ALGORITHM AS 294.4 APPL.STATIST. (1994) VOL.43, N0.3}

 {Find the reduced formula of LR statistic}
```

```
VAR i,j,k,m,n,p: NonnegInt;       VarSetModel,UnionN,Union:   IntSet;
    Reducible:   Boolean;         M1SubGeClass,M2SubGeClass:  SetArr2;
    M1NumNumer,M2NumNumer,M1NumDenom,M2NumDenom,
      M1NumSubGC,M2NumSubGC,NNumNumer,DNumNumer:              IntGC;
    M1SizeSubGC,M2SizeSubGC,NNumerInd,DNumerInd:              IntArr;
    M1Numerator,M2Numerator,M1Denominator,M2Denominator,
      M1NumerOC,M2NumerOC,M1DenomOC,M2DenomOC,M1SGClassOC,
      M2SGClassOC,NNumer,DNumer,NOCNumer,DOCNumer:            SetArr;
{Check whether the sub-generating classes in models 1 and 2 are same}
FUNCTION ClassesNotSame(h, i, j: NonnegInt): Boolean;
VAR k: IntGC;  m: NonnegInt;  NotSame: Boolean;
BEGIN  k:=0;  NotSame:=LRNSizeGeClass[h,i]<>LRDSizeGeClass[h,j];
 WHILE NOT NotSame AND (k<LRNSizeGeClass[h,i]) DO
  BEGIN  k:=k+1;  m:=1;
   WHILE (m<=LRDSizeGeClass[h,j]) AND
         (LRNGeClass[h,i,k]<>LRDGeClass[h,j,m]) DO m:=m+1;
   NotSame:=NotSame OR (m>LRDSizeGeClass[h,j])
  END;
 ClassesNotSame:=NotSame
END; {of ClassesNotSame}

{Find the union of generators}
PROCEDURE UnionSet(NumSet: IntGC;  Sets: SetArr;  VAR Union: IntSet);
VAR i: NonnegInt;
BEGIN  Union:=[];  FOR i:=1 TO NumSet DO  Union:=Union+Sets[i]
END;

{Set summed numerator and summed denominator of LR}
PROCEDURE Sum(ObsConfig: IntSet; LRNumNumer,LRNumGeClass: IntGC;
  VAR NumNumer: IntGC;  VAR LRNumer,LROCNumer,LROCGeClass,Numer,OCNumer:
   SetArr;  VAR LRSizeGeClass,NumerInd: IntArr; VAR LRGeClass: SetArr2);
VAR p:IntGC; j,k,m: NonnegInt;  Union,Unionj,UnionOther: IntSet;
BEGIN  UnionOther:=[];
 FOR k:=1 TO LRNumGeClass DO
  FOR m:=1 TO LRSizeGeClass[k] DO UnionOther:=UnionOther+LRGeClass[k,m];
 p:=0;
 FOR j:=1 TO LRNumNumer DO
  BEGIN Union:=UnionOther;
   FOR k:=1 TO LRNumNumer DO  IF k<>j THEN Union:=Union+LRNumer[k];
   IF LRNumer[j]*Union <= ObsConfig THEN
    BEGIN p:=p+1; Numer[p]:=LRNumer[j]*ObsConfig;
     OCNumer[p]:=LROCNumer[j]; NumerInd[p]:=j
    END
  END; {of FOR j}
 UnionOther:=[];
 FOR k:=1 TO LRNumNumer DO UnionOther:=UnionOther+LRNumer[k];
 FOR j:=1 TO LRNumGeClass DO
  BEGIN Union:=UnionOther; Unionj:=[];
   FOR k:=1 TO LRNumGeClass DO
    FOR m:=1 TO LRSizeGeClass[k] DO
     IF k=j THEN Unionj:=Unionj+LRGeClass[j,m]
     ELSE        Union :=Union +LRGeClass[k,m];
   IF Unionj*Union <= ObsConfig THEN
    BEGIN  m:=1;  Unionj:=Unionj*ObsConfig;
     WHILE (m<=LRSizeGeClass[j]) AND NOT(LRGeClass[j,m]>=Unionj) DO
      m:=m+1;
     IF m<=LRSizeGeClass[j] THEN
      BEGIN p:=p+1; Error(p,MaxGenerator,8,false,Fault);
       OCNumer[p]:=LROCGeClass[j]; Numer[p]:=Unionj;
       NumerInd[p]:=LRNumNumer+j
```

```
        END
      END
    END; {of FOR j}
   NumNumer:=p
  END; {of Sum}

  {Delete null numerator}
  PROCEDURE Delete(VAR LRNumNumer,LRNumGeClass: IntGC;
    VAR LRNumer,LROCNumer,LROCGeClass: SetArr; VAR LRSizeGeClass: IntArr;
    VAR LRGeClass: SetArr2);
  VAR j,k,m: NonnegInt;
  BEGIN  k:=0;
   FOR j:=1 TO LRNumNumer DO  IF LRNumer[j] <> [] THEN
    BEGIN k:=k+1; LRNumer[k]:=LRNumer[j]; LROCNumer[k]:=LROCNumer[j]
    END;
   LRNumNumer:=k; k:=0;
   FOR j:=1 TO LRNumGeClass DO  IF LRSizeGeClass[j]<>0 THEN
    BEGIN k:=k+1; LRSizeGeClass[k]:=LRSizeGeClass[j];
     LROCGeClass[k]:=LROCGeClass[j];
     FOR m:=1 TO LRSizeGeClass[j] DO LRGeClass[k,m]:=LRGeClass[j,m]
    END;
   LRNumGeClass:=k
  END; {of Delete}
  BEGIN {of MissHiMod}
   M1NumNumer:=0; M1NumDenom:=0; M1NumSubGC:=0; VariableSet:=[];
   M2NumNumer:=0; M2NumDenom:=0; M2NumSubGC:=0; Fault:=0;
   IF NumModels <> 2 THEN Fault:=9
   ELSE
    BEGIN
     MissMLE(M1SizeGeClass,NumObsConfig,M1GeClass,ObsConfig,M1NumNumer,
       M1NumDenom,M1NumSubGC,M1Numerator,M1Denominator,M1NumerOC,
       M1DenomOC,M1SGClassOC,M1SubGeClass,M1SizeSubGC,VarSetModel,Fault);
     VariableSet:=VarSetModel;
     IF Fault=0 THEN
      BEGIN
       MissMLE(M2SizeGeClass,NumObsConfig,M2GeClass,ObsConfig,M2NumNumer,
        M2NumDenom,M2NumSubGC,M2Numerator,M2Denominator,M2NumerOC,
        M2DenomOC,M2SGClassOC,M2SubGeClass,M2SizeSubGC,VarSetModel,Fault);
       IF VariableSet<>VarSetModel THEN Fault:=10;
       IF Fault=0 THEN
  {Find the likelihood ratio: M1/M2}
        FOR i:=1 TO NumObsConfig DO
         BEGIN  k:=0;
  {Set the numerator of LR}
          FOR j:=1 TO M1NumNumer DO
           IF i IN M1NumerOC[j] THEN
            BEGIN  k:=k+1; LRNNumer[i,k]:=M1Numerator[j];
             LRNOCNumer[i,k]:=M1NumerOC[j]
            END;
          LRNNumNumer[i]:=k; k:=0;
          FOR j:=1 TO M1NumDenom DO
           IF i IN M1DenomOC[j] THEN
            BEGIN k:=k+1; LRNDenom[i,k]:=M1Denominator[j];
             LRNOCDenom[i,k]:=M1DenomOC[j]
            END;
          LRNNumDenom[i]:=k; k:=0;
          FOR j:=1 TO M1NumSubGC DO
           IF i IN M1SGClassOC[j] THEN
            BEGIN  k:=k+1; LRNSizeGeClass[i,k]:=M1SizeSubGC[j];
             LRNOCGeClass[i,k]:=M1SGClassOC[j];
             FOR m:=1 TO M1SizeSubGC[j] DO
```

```
            LRNGeClass[i,k,m]:=M1SubGeClass[j,m]
          END;
        LRNNumGeClass[i]:=k; k:=0;
{Set the denominator of LR}
        FOR j:=1 TO M2NumNumer DO
        IF i IN M2NumerOC[j] THEN
          BEGIN  k:=k+1; LRDNumer[i,k]:=M2Numerator[j];
           LRDOCNumer[i,k]:=M2NumerOC[j]
          END;
        LRDNumNumer[i]:=k; k:=0;
        FOR j:=1 TO M2NumDenom DO
         IF i IN M2DenomOC[j] THEN
          BEGIN k:=k+1; LRDDenom[i,k]:=M2Denominator[j];
           LRDOCDenom[i,k]:=M2DenomOC[j]
          END;
        LRDNumDenom[i]:=k; k:=0;
        FOR j:=1 TO M2NumSubGC DO
         IF i IN M2SGClassOC[j] THEN
          BEGIN  k:=k+1; LRDSizeGeClass[i,k]:=M2SizeSubGC[j];
           LRDOCGeClass[i,k]:=M2SGClassOC[j];
           FOR m:=1 TO M2SizeSubGC[j] DO
            LRDGeClass[i,k,m]:=M2SubGeClass[j,m]
          END;
        LRDNumGeClass[i]:=k;
{Reduce the numerator and denominator contained in ObsConfig[i]}
        Reduce(LRNNumNumer[i],LRDNumNumer[i],LRNNumer[i],LRNOCNumer[i],
               LRDNumer[i],LRDOCNumer[i],ObsConfig[i]);
        Reduce(LRNNumDenom[i],LRDNumDenom[i],LRNDenom[i],LRNOCDenom[i],
               LRDDenom[i],LRDOCDenom[i],ObsConfig[i]);
{Reduce sub-generating classes contained in ObsConfig[i]}
        k:=0;
        FOR m:=1 TO LRNNumGeClass[i] DO
         BEGIN  j:=1;
          UnionSet(LRNSizeGeClass[i,m],LRNGeClass[i,m],UnionN);
          IF ObsConfig[i] >= UnionN THEN
           BEGIN UnionSet(LRDSizeGeClass[i,j],LRDGeClass[i,j],Union);
            WHILE (j<=LRDNumGeClass[i]) AND (NOT(ObsConfig[i]>=Union) OR
                  (LRDOCGeClass[i,j]<>LRNOCGeClass[i,m]) OR
                  ClassesNotSame(i,m,j)) DO
             BEGIN j:=j+1;
              IF j<=LRDNumGeClass[i] THEN
               UnionSet(LRDSizeGeClass[i,j],LRDGeClass[i,j],Union)
             END;
            Reducible:= j <= LRDNumGeClass[i];
            IF Reducible THEN
             BEGIN
              FOR p:=j+1 TO LRDNumGeClass[i] DO
               BEGIN  LRDSizeGeClass[i,p-1]:=LRDSizeGeClass[i,p];
                LRDOCGeClass[i,p-1]:=LRDOCGeClass[i,p];
                FOR n:=1 TO LRDSizeGeClass[i,p-1] DO
                 LRDGeClass[i,p-1,n]:=LRDGeClass[i,p,n]
               END;
              LRDNumGeClass[i]:=LRDNumGeClass[i]-1
             END
           END;
          IF NOT Reducible OR NOT(ObsConfig[i] >= UnionN) THEN
           BEGIN k:=k+1; LRNSizeGeClass[i,k]:=LRNSizeGeClass[i,m];
            LRNOCGeClass[i,k]:=LRNOCGeClass[i,m];
            FOR n:=1 TO LRNSizeGeClass[i,k] DO
             LRNGeClass[i,k,n]:=LRNGeClass[i,m,n]
           END
```

```
          END; {of FOR m}
         LRNNumGeClass[i]:=k;
   {Reduce summed numerator and denominator of LR}
         Sum(ObsConfig[i],LRNNumNumer[i],LRNNumGeClass[i],NNumNumer,
             LRNNumer[i],LRNOCNumer[i],LRNOCGeClass[i],NNumer,NOCNumer,
             LRNSizeGeClass[i],NNumerInd,LRNGeClass[i]);
         Sum(ObsConfig[i],LRDNumNumer[i],LRDNumGeClass[i],DNumNumer,
             LRDNumer[i],LRDOCNumer[i],LRDOCGeClass[i],DNumer,DOCNumer,
             LRDSizeGeClass[i],DNumerInd,LRDGeClass[i]);
         FOR p:=1 TO NNumNumer DO
          BEGIN  j:=1;
           WHILE (j<=DNumNumer) AND ((NOCNumer[p]<>DOCNumer[j]) OR
                 (NNumer[p]<>DNumer[j])) DO j:=j+1;
           IF j<=DNumNumer THEN
            BEGIN n:=NNumerInd[p];
             IF n<=LRNNumNumer[i] THEN LRNNumer[i,n]:=[]
             ELSE LRNSizeGeClass[i,n-LRNNumNumer[i]]:=0;
             n:=DNumerInd[j];
             IF n<=LRDNumNumer[i] THEN LRDNumer[i,n]:=[]
             ELSE LRDSizeGeClass[i,n-LRDNumNumer[i]]:=0
            END
          END; {of FOR p}
         Delete(LRNNumNumer[i],LRNNumGeClass[i],LRNNumer[i],
          LRNOCNumer[i],LRNOCGeClass[i],LRNSizeGeClass[i],LRNGeClass[i]);
         Delete(LRDNumNumer[i],LRDNumGeClass[i],LRDNumer[i],
          LRDOCNumer[i],LRDOCGeClass[i],LRDSizeGeClass[i],LRDGeClass[i]);
   {Reduce the whole LR if numerator is the same as denominator}
         Reducible:= (LRNNumNumer[i]=LRDNumNumer[i]) AND
                     (LRNNumDenom[i]=LRDNumDenom[i]) AND
                     (LRNNumGeClass[i]=LRDNumGeClass[i]);
         IF Reducible THEN
          BEGIN  j:=1;
           WHILE Reducible AND (j<=LRNNumNumer[i]) DO
            BEGIN  Reducible:=Reducible AND (LRNNumer[i,j]=LRDNumer[i,j])
                     AND (LRNOCNumer[i,j]=LRDOCNumer[i,j]);  j:=j+1
            END;
           j:=1;
           WHILE Reducible AND (j<=LRNNumDenom[i]) DO
            BEGIN  Reducible:=Reducible AND (LRNDenom[i,j]=LRDDenom[i,j])
                     AND (LRNOCDenom[i,j]=LRDOCDenom[i,j]);  j:=j+1
            END;
           j:=1;
           WHILE Reducible AND (j<=LRNNumGeClass[i]) DO
            BEGIN  Reducible:= Reducible
                     AND (LRNOCGeClass[i,j]=LRDOCGeClass[i,j]);  k:=1;
             WHILE ClassesNotSame(i,j,k) AND (k<=LRDNumGeClass[i]) DO
              k:=k+1;
             Reducible:=Reducible AND (k<=LRDNumGeClass[i]);  j:=j+1
            END;
           IF Reducible THEN
            BEGIN LRNNumNumer[i]:=0;LRDNumNumer[i]:=0;LRNNumDenom[i]:=0;
             LRDNumDenom[i]:=0; LRNNumGeClass[i]:=0; LRDNumGeClass[i]:=0
            END
          END {IF Reducible THEN}
        END {of FOR i}
    END
  END
END; {of MissHiMod}
```

```
      SUBROUTINE DOPT(X, DIM1, NCAND, KIN, N, NBLOCK, IN, BLKSIZ, K,
     *    RSTART, NRBAR, D, RBAR, PICKED, LNDET, XX, TOL, ZPZ, WK,
     *    IFAULT)
C
C     ALGORITHM AS295.1 APPL. STATIST. (1994) VOL.43, NO.4
C
C     Heuristic algorithm to pick N rows of X out of NCAND to
C     maximize the determinant of X'X, using the Fedorov exchange
C     algorithm.
C
      INTEGER DIM1, NCAND, KIN, N, NBLOCK, IN(*), BLKSIZ(*), K, NRBAR,
     *        PICKED(N), IFAULT
      REAL X(DIM1, KIN), D(K), RBAR(NRBAR), LNDET, XX(K),
     *     TOL(K), ZPZ(NCAND, *), WK(K)
      LOGICAL RSTART
C
      INTEGER I, J, NIN, POINT, CASE, NB, BLOCK, L, POS, BEST, FIRST,
     *    LAST, CAND, LASTIN, LSTOUT, DROP, REMPOS, BL, RPOS, LAST1,
     *    LAST2, FIRST1, FIRST2, POS1, POS2, BLOCK1, BLOCK2, CASE1,
     *    CASE2, POSI, POSJ, BESTB1, BESTB2, BESTP1, BESTP2, RANK,
     *    MXRANK, INC
      REAL ONE, ZERO, MINUS1, TEMP, EPS, DETMAX, ABOVE1,
     *    SUM, SMALL, HUNDRD
      LOGICAL CHANGE
C
      REAL DELTA, RAND
      EXTERNAL BKSUB1, BKSUB2, CLEAR, DELTA, GETX, MODTRI, MODTR2,
     *         RAND, REGCF, SINGM
C
      DATA ONE /1.0E + 00/,  ZERO /0.0E + 00/,  ABOVE1 /1.0001E + 00/,
     *     EPS /1.0E - 06/, MINUS1 /-1.0E + 00/,  SMALL /1.0E - 04/,
     *     HUNDRD /100.0E + 00/
C
      IFAULT = 0
      IF (DIM1 .LT. NCAND) IFAULT = 1
      IF (K .GT. N) IFAULT = IFAULT + 2
      IF (NRBAR .LT. K*(K-1)/2) IFAULT = IFAULT + 4
      IF (K .NE. KIN+NBLOCK) IFAULT = IFAULT + 8
      IF (NBLOCK .GT. 1) THEN
         L = 0
         DO 10 BLOCK = 1, NBLOCK
            L = L + BLKSIZ(BLOCK)
   10    CONTINUE
         IF (N .NE. L) IFAULT = IFAULT + 16
      ELSE
         IF (N .NE. BLKSIZ(1)) IFAULT = IFAULT + 16
      END IF
C
C     NB = max(1, NBLOCK) so that we can force it to go through
C     DO-loops once. NIN = no. of design points forced into the
C     design.
C
      NB = MAX(1, NBLOCK)
      NIN = 0
      DO 20 I = 1, NB
         IF (IN(I) .LT. 0) GO TO 30
         IF (IN(I) .GT. 0) THEN
            IF (IN(I) .GT. BLKSIZ(I)) GO TO 30
            NIN = NIN + IN(I)
         END IF
   20 CONTINUE
```

```
      IF (NIN .LE. N) GO TO 40
   30 IFAULT = IFAULT + 32
   40 CONTINUE
      IF (IFAULT .NE. 0) RETURN
      CALL CLEAR(K, NRBAR, D, RBAR, IFAULT)
C
C        Set up an array of tolerances
C
      DO 50 I = 1, K
         TOL(I) = ZERO
   50 CONTINUE
      BLOCK = 1
      DO 70 CASE = 1, NCAND
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, CASE)
         DO 60 I = 1, K
            TOL(I) = TOL(I) + ABS(XX(I))
   60    CONTINUE
   70 CONTINUE
      TEMP = FLOAT(N) * EPS / NCAND
      DO 80 I = 1, K
         IF (I .LE. NBLOCK) THEN
            TOL(I) = EPS
         ELSE
            TOL(I) = TOL(I) * TEMP
         END IF
   80 CONTINUE
C
C        Form initial Cholesky factorization
C
      POS = 1
      DO 120 BLOCK = 1, NB
         IF (RSTART) THEN
            LAST1 = (IN(BLOCK) + BLKSIZ(BLOCK))/2
            INC = SQRT(FLOAT(NCAND) + SMALL)
         END IF
         DO 110 I = 1, BLKSIZ(BLOCK)
            IF (RSTART .AND. I .GT. IN(BLOCK)) THEN
               POINT = 1 + NCAND * RAND()
C
C     If I <= LAST1, use a random point, otherwise find the
C     candidate which maximizes the rank, and then maximizes the
C     subspace determinant for that rank.
C
               IF (I .GT. LAST1) THEN
                  MXRANK = 0
                  LNDET = -HUNDRD
                  DO 100 CAND = 1, NCAND, INC
                  CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, POINT)
                  CALL MODTR2(K, NRBAR, XX, D, RBAR, TOL, RANK, SUM)
                  IF (RANK .LT. MXRANK) GO TO 90
                  IF (RANK .EQ. MXRANK .AND. SUM .LT. LNDET) GO TO 90
                  BEST = POINT
                  MXRANK = RANK
                  LNDET = SUM * ABOVE1
   90             POINT = POINT + INC
                  IF (POINT .GT. NCAND) POINT = POINT - NCAND
  100             CONTINUE
                  POINT = BEST
               END IF
               PICKED(POS) = POINT
            ELSE
```

```
C
C        Case in which a full design has been input, or points are
C        to be forced into the design.
C
               POINT = PICKED(POS)
            END IF
C
C        Augment the Cholesky factorization
C
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, POINT)
            CALL MODTRI(K, NRBAR, ONE, XX, D, RBAR, TOL)
            POS = POS + 1
  110    CONTINUE
  120 CONTINUE
C
C        Adjust factorization in case of singular matrix
C
      CALL SINGM(K, NRBAR, D, RBAR, TOL, WK, IFAULT)
C
C        If rank of input design < K, try replacing points.
C
      IF (IFAULT .EQ. 0) GO TO 280
C
C     Find first row of Cholesky factorization with a zero
C     multiplier
C
  180 DO 190 POS = 1, K
         IF (D(POS) .LT. TOL(POS)) GO TO 200
  190 CONTINUE
      GO TO 280
C
C     Find linear relationship between variable in position POS
C     and the previous variables
C
  200 L = POS - 1
      DO 210 I = 1, POS - 1
         WK(I) = RBAR(L)
         L = L + K - I - 1
  210 CONTINUE
      CALL REGCF(K, NRBAR, D, RBAR, WK, TOL, WK, POS-1, IFAULT)
C
C     Find a candidate point which does not satisfy this linear
C     relationship.   Use a random start.
C
      BL = 1
      CASE = 1 + NCAND * RAND()
      DO 230 CAND = 1, NCAND
         CALL GETX(X, DIM1, KIN, NBLOCK, K, BL, XX, CASE)
         SUM = XX(POS)
         DO 220 I = 1, POS - 1
            SUM = SUM - WK(I) * XX(I)
  220    CONTINUE
         IF (ABS(SUM) .GT. HUNDRD * TOL(POS)) GO TO 240
         CASE = CASE + 1
         IF (CASE .GT. NCAND) CASE = 1
  230 CONTINUE
C
C     Failed to find any candidate point which would make the design
C     of higher rank
C
      IFAULT = -1
```

```
      RETURN
C
C     Before adding the point, find one which it can replace without
C     lowering the rank.
C
  240 BL = 0
      TEMP = ONE - SMALL
      POS = IN(1) + 1
      DO 270 BLOCK = 1, NB
         DO 260 J = IN(BLOCK) + 1, BLKSIZ(BLOCK)
            L = PICKED(POS)
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, L)
            CALL BKSUB2(RBAR, NRBAR, K, XX, WK)
            SUM = ZERO
            DO 250 I = 1, K
               IF (D(I) .GT. TOL(I)) SUM = SUM + WK(I)**2 / D(I)
  250       CONTINUE
            IF (SUM .LT. TEMP) THEN
               TEMP = SUM
               REMPOS = POS
               BL = BLOCK
            END IF
            POS = POS + 1
  260    CONTINUE
         IF (BLOCK .LT. NBLOCK) POS = POS + IN(BLOCK+1)
  270 CONTINUE
C
C     If BL = 0 it means that any point removed from the existing
C     design would reduce the rank
C
      IF (BL .EQ. 0) THEN
         IFAULT = -1
         RETURN
      END IF
C
C     Add candidate CASE in block BL, then delete the design point
C     already in that position.
C
      CALL GETX(X, DIM1, KIN, NBLOCK, K, BL, XX, CASE)
      CALL MODTRI(K, NRBAR, ONE, XX, D, RBAR, TOL)
      L = PICKED(REMPOS)
      CALL GETX(X, DIM1, KIN, NBLOCK, K, BL, XX, L)
      CALL MODTRI(K, NRBAR, MINUS1, XX, D, RBAR, TOL)
      PICKED(REMPOS) = CASE
      GO TO 180
C
C     Design is now of full rank. Calculate z'z for all candidate
C     points. z is the solution of R'z = x, so that
C     z'z = x'.inv(X'X).x. WK holds sqrt(D) times vector z on
C     return from BKSUB2.
C
  280 DO 310 BLOCK = 1, NB
         DO 300 CASE = 1, NCAND
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, CASE)
            CALL BKSUB2(RBAR, NRBAR, K, XX, WK)
            TEMP = ZERO
            DO 290 I = 1, K
               TEMP = TEMP + WK(I)**2 / D(I)
  290       CONTINUE
            ZPZ(CASE, BLOCK) = TEMP
  300    CONTINUE
```

```
  310 CONTINUE
C
C     Start of Fedorov exchange algorithm
C
      LASTIN = 0
      LSTOUT = 0
  320 CHANGE = .FALSE.
      LAST = 0
      DO 420 BLOCK = 1, NB
         FIRST = LAST + 1 + IN(BLOCK)
         LAST = LAST + BLKSIZ(BLOCK)
         DETMAX = SMALL
         BEST = 0
C
C     Start at a random position within the block.
C     I = no. of point being considered for deletion.
C
         POS = FIRST + (BLKSIZ(BLOCK) - IN(BLOCK)) * RAND()
         DO 350 CASE = IN(BLOCK)+1, BLKSIZ(BLOCK)
            POS = POS + 1
            IF (POS .GT. LAST) POS = FIRST
            I = PICKED(POS)
            IF (I .EQ. LASTIN) GO TO 350
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, I)
            CALL BKSUB2(RBAR, NRBAR, K, XX, WK)
            CALL BKSUB1(RBAR, NRBAR, K, WK, WK, TOL, D)
C
C     Cycle through the candidates for exchange, using a random
C     start. J = no. of point being considered for addition.
C
            J = 1 + NCAND * RAND()
            DO 340 CAND = 1, NCAND
               J = J + 1
               IF (J .GT. NCAND) J = 1
               IF (J .EQ. I .OR. J .EQ. LSTOUT) GO TO 340
C
C     The Cauchy-Schwarz test
C
               TEMP = ZPZ(J,BLOCK) - ZPZ(I,BLOCK)
               IF (TEMP .LT. DETMAX) GO TO 340
C
               CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, J)
               SUM = ZERO
               DO 330 L = 1, K
                  SUM = SUM + XX(L) * WK(L)
  330          CONTINUE
               TEMP = TEMP + SUM**2 - ZPZ(I,BLOCK) * ZPZ(J,BLOCK)
               IF (TEMP .GT. DETMAX) THEN
                  DETMAX = TEMP * ABOVE1
                  BEST = J
                  REMPOS = POS
                  DROP = I
               END IF
  340       CONTINUE
  350    CONTINUE
C
C     Exchange points BEST and DROP in position REMPOS, if the
C     determinant is increased.
C
         IF (BEST .NE. 0) THEN
            CHANGE = .TRUE.
```

```
            IF (NB .EQ. 1) THEN
               LASTIN = BEST
               LSTOUT = DROP
            END IF
C
C     Add the new point, BEST, first to avoid ill-conditioning.
C     Update z'z.
C
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, BEST)
            CALL BKSUB2(RBAR, NRBAR, K, XX, WK)
            CALL BKSUB1(RBAR, NRBAR, K, WK, WK, TOL, D)
            CALL MODTRI(K, NRBAR, ONE, XX, D, RBAR, TOL)
            TEMP = ONE + ZPZ(BEST,BLOCK)
            DO 360 BL = 1, NB
               DO 380 CASE = 1, NCAND
                  CALL GETX(X, DIM1, KIN, NBLOCK, K, BL, XX, CASE)
                  SUM = ZERO
                  DO 370 L = 1, K
                     SUM = SUM + XX(L) * WK(L)
  370             CONTINUE
                  ZPZ(CASE,BL) = ZPZ(CASE,BL) - SUM**2 / TEMP
  380          CONTINUE
  360       CONTINUE
C
C     Remove the point DROP, and update z'z.
C
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, DROP)
            CALL BKSUB2(RBAR, NRBAR, K, XX, WK)
            CALL BKSUB1(RBAR, NRBAR, K, WK, WK, TOL, D)
            CALL MODTRI(K, NRBAR, MINUS1, XX, D, RBAR, TOL)
            TEMP = ONE - ZPZ(DROP,BLOCK)
            DO 390 BL = 1, NB
               DO 410 CASE = 1, NCAND
                  CALL GETX(X, DIM1, KIN, NBLOCK, K, BL, XX, CASE)
                  SUM = ZERO
                  DO 400 L = 1, K
                     SUM = SUM + XX(L)*WK(L)
  400             CONTINUE
                  ZPZ(CASE,BL) = ZPZ(CASE,BL) + SUM**2 / TEMP
  410          CONTINUE
  390       CONTINUE
C
            PICKED(REMPOS) = BEST
         END IF
  420 CONTINUE
C
C     Repeat until there is no further improvement
C
      IF (CHANGE) GO TO 320
C
C     If there is more than one block, try swapping treatments
C     between blocks. This is the Cook & Nachtsheim(1989) algorithm.
C
      IF (NBLOCK .LE. 1) GO TO 500
C
C     RPOS is the position of the first element in RBAR after the
C     rows for the block constants
C
      RPOS = NBLOCK * K - NBLOCK * (NBLOCK + 1)/2 + 1
C
  430 LAST1 = 0
```

```
C
C     POS1 and POS2 will hold the positions of the start of the means
C     of the X-variables in the two blocks being considered in RBAR
C
      POS1 = NBLOCK
      DETMAX = SMALL
      CHANGE = .FALSE.
      DO 490 BLOCK1 = 1, NBLOCK - 1
         FIRST1 = LAST1 + 1 + IN(BLOCK1)
         LAST1 = LAST1 + BLKSIZ(BLOCK1)
         LAST2 = LAST1
         POS2 = POS1 + K - 1 - BLOCK1
         DO 480 BLOCK2 = BLOCK1+1, NBLOCK
            FIRST2 = LAST2 + 1 + IN(BLOCK2)
            LAST2 = LAST2 + BLKSIZ(BLOCK2)
            DO 470 CASE1 = IN(BLOCK1)+1, BLKSIZ(BLOCK1)
               POSI = FIRST1 - 1 + CASE1
               I = PICKED(POSI)
               CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, I)
               DO 440 L = 1, KIN
                  ZPZ(L,1) = XX(L+NBLOCK)
 440           CONTINUE
               DO 460 CASE2 = IN(BLOCK2) + 1, BLKSIZ(BLOCK2)
                  POSJ = FIRST2 - 1 + CASE2
                  J = PICKED(POSJ)
                  IF (I .EQ. J) GO TO 460
                  CALL GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, J)
                  DO 450 L = 1, KIN
                     ZPZ(L,2) = XX(L+NBLOCK)
  450             CONTINUE
C
C     Pass the orthogonal factorization to DELTA with the top NBLOCK
C     rows removed, i.e. without that part relating to the blocks.
C
               TEMP = DELTA(KIN, ZPZ(1,1), ZPZ(1,2), RBAR(POS1),
     *        RBAR(POS2), BLKSIZ(BLOCK1), BLKSIZ(BLOCK2), NRBAR,
     *        D(NBLOCK+1), RBAR(RPOS), ZPZ(1,3), WK, XX, ZPZ(1,2))
                  IF (TEMP .GT. DETMAX) THEN
                     DETMAX = TEMP * ABOVE1
                     BESTB1 = BLOCK1
                     BESTB2 = BLOCK2
                     BESTP1 = POSI
                     BESTP2 = POSJ
                     CHANGE = .TRUE.
                  END IF
  460          CONTINUE
  470       CONTINUE
            POS2 = POS2 + K - 1 - BLOCK2
  480    CONTINUE
         POS1 = POS1 + K - 1 - BLOCK1
  490 CONTINUE
C
C  If CHANGE=.TRUE. then make the swap, otherwise the search ends.
C
      IF (CHANGE) THEN
         I = PICKED(BESTP1)
         J = PICKED(BESTP2)
         CALL GETX(X, DIM1, KIN, NBLOCK, K, BESTB2, XX, I)
         CALL MODTRI(K, NRBAR, ONE, XX, D, RBAR, TOL)
         CALL GETX(X, DIM1, KIN, NBLOCK, K, BESTB1, XX, J)
         CALL MODTRI(K, NRBAR, ONE, XX, D, RBAR, TOL)
```

```
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BESTB1, XX, I)
            CALL MODTRI(K, NRBAR, MINUS1, XX, D, RBAR, TOL)
            CALL GETX(X, DIM1, KIN, NBLOCK, K, BESTB2, XX, J)
            CALL MODTRI(K, NRBAR, MINUS1, XX, D, RBAR, TOL)
            PICKED(BESTP1) = J
            PICKED(BESTP2) = I
            GO TO 430
         END IF
C
C        Calculate log of determinant
C
  500 LNDET = ZERO
         DO 510 I = 1, K
            LNDET = LNDET + LOG(D(I))
  510 CONTINUE
         RETURN
         END
C
         SUBROUTINE MODTRI(NP, NRBAR, WEIGHT, XROW, D, RBAR, TOL)
C
C        ALGORITHM AS295.2 APPL. STATIST. (1994) VOL.43, NO.4
C
C        Modify a triangular (Cholesky) decomposition. Calling this
C        routine updates D and RBAR by adding another design point with
C        weight = WEIGHT, which may be negative. Algorithm based on
C        AS75.1 with modifications.
C        *** WARNING: Array XROW is overwritten ***
C
         INTEGER NP, NRBAR
         REAL WEIGHT, XROW(NP), D(NP), RBAR(*), TOL(NP)
C
         INTEGER I, K, NEXTR
         REAL CBAR, DI, DPI, SBAR, W, WXI, XI, XK, ZERO
C
         DATA ZERO /0.0E + 00/
C
         W = WEIGHT
         NEXTR = 1
         DO 30 I = 1, NP
C
C        Skip unnecessary transformations.   Test on exact zeroes must
C        be used or stability can be destroyed.
C
            IF (W .EQ. ZERO) RETURN
C
            XI = XROW(I)
            IF (ABS(XI) .LT. TOL(I)) THEN
               NEXTR = NEXTR + NP - I
               GO TO 30
            END IF
C
            DI = D(I)
            WXI = W*XI
            DPI = DI + WXI * XI
C
C           Test for new singularity
C
            IF (DPI .LT. TOL(I)) THEN
               DPI = ZERO
               CBAR = ZERO
               SBAR = ZERO
```

```
            W = ZERO
         ELSE
            CBAR = DI/DPI
            SBAR = WXI/DPI
            W = CBAR*W
         END IF
C
         D(I) = DPI
         DO 20 K = I + 1, NP
            XK = XROW(K)
            XROW(K) = XK - XI*RBAR(NEXTR)
            RBAR(NEXTR) = CBAR*RBAR(NEXTR) + SBAR*XK
            NEXTR = NEXTR + 1
   20    CONTINUE
   30 CONTINUE
      RETURN
      END
C
      SUBROUTINE MODTR2(NP, NRBAR, XROW, D, RBAR, TOL, RANK, LNDET)
C
C        ALGORITHM AS295.3 APPL. STATIST. (1994) VOL.43, NO.4
C
C        Calculate the effect of an update of a QR factorization
C        upon the rank and determinant, without changing D or
C        RBAR. Algorithm based on AS75.1 with modifications.
C        *** WARNING: Array XROW is overwritten ***
C
      INTEGER NP, NRBAR, RANK
      REAL XROW(NP), D(NP), RBAR(*), TOL(NP), LNDET
C
      INTEGER I, J, K, NEXTR
      REAL CBAR, DI, DPI, ONE, W, WXI, XI, XK, ZERO
C
      DATA ZERO /0.0E + 00/, ONE /1.0E + 00/
C
      W = ONE
      RANK = 0
      LNDET = ZERO
      NEXTR = 1
      DO 30 I = 1, NP
C
C     Skip unnecessary transformations. Test on exact zeroes must be
C     used or stability can be destroyed.
C
         IF (W .EQ. ZERO) THEN
            DO 10 J = I, NP
               IF (D(J) .GT. TOL(J)) THEN
                  RANK = RANK + 1
                  LNDET = LNDET + LOG(D(J))
               END IF
   10       CONTINUE
            RETURN
         END IF
C
         XI = XROW(I)
         IF (ABS(XI) .LT. TOL(I)) THEN
            IF (D(I) .GT. TOL(I)) THEN
               RANK = RANK + 1
               LNDET = LNDET + LOG(D(I))
            END IF
            NEXTR = NEXTR + NP - I
```

```
               GO TO 30
            END IF
C
            DI = D(I)
            WXI = W * XI
            DPI = DI + WXI * XI
C
C           Test for new singularity
C
            IF (DPI .LT. TOL(I)) THEN
               DPI = ZERO
               CBAR = ZERO
               W = ZERO
            ELSE
               CBAR = DI / DPI
               W = CBAR * W
               LNDET = LNDET + LOG(DPI)
               RANK = RANK + 1
            END IF
C
            DO 20 K = I + 1, NP
               XK = XROW(K)
               XROW(K) = XK - XI * RBAR(NEXTR)
               NEXTR = NEXTR + 1
   20       CONTINUE
   30    CONTINUE
         RETURN
         END
C
         SUBROUTINE GETX(X, DIM1, KIN, NBLOCK, K, BLOCK, XX, CASE)
C
C           ALGORITHM AS295.4 APPL. STATIST. (1994) VOL.43, NO.4
C
C           Copy one case from X to XX
C
         INTEGER DIM1, KIN, NBLOCK, K, BLOCK, CASE
         REAL X(DIM1, KIN), XX(K)
C
         INTEGER I, J
         REAL ONE, ZERO
C
         DATA ZERO /0.0E + 00/, ONE /1.0E + 00/
C
         DO 10 I = 1, NBLOCK
            IF (I .NE. BLOCK) THEN
               XX(I) = ZERO
            ELSE
               XX(I) = ONE
            END IF
   10    CONTINUE
         J = NBLOCK + 1
         DO 20 I = 1, KIN
            XX(J) = X(CASE, I)
            J = J + 1
   20    CONTINUE
         RETURN
         END
C
         SUBROUTINE BKSUB1(RBAR, NRBAR, K, RHS, SOLN, TOL, D)
C
C     ALGORITHM AS295.5 APPL. STATIST. (1994) VOL.43, NO.4
```

```
C
C     Solves  D R y = z for y (SOLN), where z = RHS.
C     RBAR is an upper-triangular matrix with implicit 1's on it's
C     diagonal, stored by rows.
C
      INTEGER NRBAR, K
      REAL RBAR(NRBAR), RHS(K), SOLN(K), TOL(K), D(K)
C
      INTEGER COL, POS, ROW
      REAL TEMP, ZERO
C
      DATA ZERO /0.0E + 00/
C
      POS = K * (K - 1) / 2
      DO 20 ROW = K, 1, -1
         IF (D(ROW) .GT. TOL(ROW)) THEN
            TEMP = RHS(ROW) / D(ROW)
            DO 10 COL = K, ROW + 1, -1
               TEMP = TEMP - RBAR(POS) * SOLN(COL)
               POS = POS - 1
   10       CONTINUE
            SOLN(ROW) = TEMP
         ELSE
            POS = POS - K + ROW
            SOLN(ROW) = ZERO
         END IF
   20 CONTINUE
      RETURN
      END
C
      SUBROUTINE BKSUB2(RBAR, NRBAR, K, RHS, SOLN)
C
C     ALGORITHM AS295.6 APPL. STATIST. (1994) VOL.43, NO.4
C
C     Solves  R'(sqrt(D).z) = x where (sqrt(D).z) = SOLN and x = RHS.
C     RBAR is an upper-triangular matrix with implicit 1's on it's
C     diagonal, stored by rows.
C
      INTEGER NRBAR, K
      REAL RBAR(NRBAR), RHS(K), SOLN(K)
C
      INTEGER COL, POS, ROW
      REAL TEMP
C
      SOLN(1) = RHS(1)
      DO 20 ROW = 2, K
         TEMP = RHS(ROW)
         POS = ROW - 1
         DO 10 COL = 1, ROW - 1
            TEMP = TEMP - RBAR(POS) * SOLN(COL)
            POS = POS + K - COL - 1
   10    CONTINUE
         SOLN(ROW) = TEMP
   20 CONTINUE
      RETURN
      END
C
      REAL FUNCTION DELTA(K, XJ, XL, XBARI, XBARK, NI, NK,
     *                    NRBAR, D, RBAR, Z, A, B, DIFF)
C
C     ALGORITHM AS295.7 APPL. STATIST. (1994) VOL.43, NO.4
```

```
C
C     Calculate the delta function for the swap of case J in block I
C     with case L in block K. Uses the method of Cook & Nachtsheim.
C
      INTEGER K, NI, NK, NRBAR
      REAL XJ(K), XL(K), XBARI(K), XBARK(K), D(K),
     *     RBAR(NRBAR), Z(K,3), A(K), B(K), DIFF(K)
C
      INTEGER I
      REAL CONST, E11, E12, E21, E22, ONE, TEMP, TWO
C
      REAL DOTPRD
      EXTERNAL BKSUB2, DOTPRD
C
      DATA ONE /1.0E + 00/, TWO /2.0E + 00/
C
C     Calculate vectors DIFF, A and B
C
      CONST = TWO - ONE/NI - ONE/NK
      DO 10 I = 1, K
         TEMP = XJ(I) - XL(I)
         DIFF(I) = -TEMP
         A(I) = TEMP - XBARI(I) + XBARK(I)
         B(I) = A(I) - CONST * TEMP
   10 CONTINUE
C
C     Calculate the z-vectors by back-substitution. Z1 for A, Z2
C     for B and Z3 for DIFF. The solutions returned from
C     BKSUB2 have the I-th element multiplied by sqrt(D(I)).
C
      CALL BKSUB2(RBAR, NRBAR, K, A, Z(1,1))
      CALL BKSUB2(RBAR, NRBAR, K, B, Z(1,2))
      CALL BKSUB2(RBAR, NRBAR, K, DIFF, Z(1,3))
C
C        Calculate the elements E11, E12, E21 and E22 as dot-products
C        of the appropriate z-vectors
C
      E11 = DOTPRD(K, Z(1,3), Z(1,1), D)
      E12 = DOTPRD(K, Z(1,3), Z(1,3), D)
      E21 = DOTPRD(K, Z(1,2), Z(1,1), D)
      E22 = DOTPRD(K, Z(1,2), Z(1,3), D)
C
C        Return the determinant of the matrix:   E11+1    E12
C                                                 E21    E22+1
C        minus 1
C
      DELTA = (E11+ONE) * (E22+ONE) - E12 * E21 - ONE
      RETURN
      END
C
      REAL FUNCTION DOTPRD(K, X, Y, D)
C
C     ALGORITHM AS295.8 APPL. STATIST. (1994) VOL.43, NO.4
C
C     Dot-product scaled by vector D
C
      INTEGER K
      REAL X(K), Y(K), D(K)
C
      INTEGER I
      REAL ZERO
```

```
C
      DATA ZERO /0.0E + 00/
C
      DOTPRD = ZERO
      DO 10 I = 1, K
         DOTPRD = DOTPRD + X(I) * Y(I) / D(I)
   10 CONTINUE
      RETURN
      END
C
      SUBROUTINE SINGM(NP, NRBAR, D, RBAR, TOL, WORK, IFAULT)
C
C        ALGORITHM AS295.9 APPL. STATIST. (1994) VOL.43, NO.4
C
C        Checks for singularities, and adjusts orthogonal
C        reductions produced by AS75.1. Modified from AS274.5
C
      INTEGER NP, NRBAR, IFAULT
      REAL D(NP), RBAR(NRBAR), TOL(NP), WORK(NP)
C
      INTEGER COL, J, NP2, POS, POS1, ROW
      REAL TEMP, ZERO
C
      EXTERNAL MODTRI
C
      DATA ZERO /0.0E + 00/
C
C     Check input parameters
C
      IFAULT = 0
      IF (NP .LE. 0) IFAULT = 1
      IF (NRBAR .LT. NP * (NP - 1)/2) IFAULT = IFAULT + 2
      IF (IFAULT .NE. 0) RETURN
C
      DO 10 COL = 1, NP
         WORK(COL) = SQRT(D(COL))
   10 CONTINUE
C
      DO 40 COL = 1, NP
C
C     Set elements within RBAR to zero if they are less than TOL(COL)
C     in absolute value after being scaled by the square root of
C     their row multiplier
C
         TEMP = TOL(COL)
         POS = COL - 1
         DO 20 ROW = 1, COL - 1
            IF (ABS(RBAR(POS)) * WORK(ROW) .LT. TEMP) RBAR(POS) = ZERO
            POS = POS + NP - ROW - 1
   20    CONTINUE
C
C     If diagonal element is near zero, set it to zero, and use
C     MODTRI to augment the projections in the lower rows of the
C     factorization.
C
         IF (WORK(COL) .LE. TEMP) THEN
            IFAULT = IFAULT - 1
         IF (COL .LT. NP) THEN
            NP2 = NP - COL
            POS2 = POS + NP - COL + 1
            IF (NP2 .GT. 1) THEN
```

```
                  CALL MODTRI(NP2, NP2*(NP2-1)/2, D(COL), RBAR(POS+1),
     *                        D(COL+1), RBAR(POS2), TOL)
               ELSE
                  CALL MODTRI(1, 0, D(COL), RBAR(POS+1), D(COL+1),
     *                        RBAR(1), TOL)
               END IF
               DO 30 J = POS + 1, POS2 - 1
                  RBAR(J) = ZERO
   30          CONTINUE
            END IF
            D(COL) = ZERO
            END IF
   40    CONTINUE
         RETURN
         END
C
         SUBROUTINE XXTR(NP, NRBAR, D, RBAR, NREQ, TRACE, RINV)
C
C           ALGORITHM AS295.10 APPL. STATIST. (1994) VOL.43, NO.4
C
C           Calculate the trace of the inverse of X'X (= R'R)
C
         INTEGER NP, NRBAR, NREQ
         REAL D(NP), RBAR(NRBAR), TRACE, RINV(*)
C
         INTEGER COL, POS, ROW
         REAL ONE, ZERO
C
         EXTERNAL INV
C           AS274.8, Appl.Statist.(1992), vol.41, no.2
C
         DATA ONE /1.0E + 00/, ZERO /0.0E + 00/
C
C        Get the inverse of R
C
         CALL INV(NP, RBAR, NREQ, RINV)
C
C        Trace = the sum of the diagonal elements
C        of RINV * (1/D) * (RINV)'
C
         TRACE = ZERO
         POS = 1
         DO 20 ROW = 1, NREQ
            TRACE = TRACE + ONE / D(ROW)
            DO 10 COL = ROW + 1, NREQ
               TRACE = TRACE + RINV(POS)**2 / D(COL)
               POS = POS + 1
   10       CONTINUE
   20    CONTINUE
         RETURN
         END
```

```
      SUBROUTINE SWILK (INIT, X, N, N1, N2, A, W, PW, IFAULT)
C
C        ALGORITHM AS R94 APPL. STATIST. (1995) VOL.44, NO.4
C
C        Calculates the Shapiro-Wilk W test and its significance level
C
      INTEGER N, N1, N2, IFAULT
      REAL X(*), A(*), PW, W
      REAL C1(6), C2(6), C3(4), C4(4), C5(4), C6(3), C7(2)
      REAL C8(2), C9(2), G(2)
      REAL Z90, Z95, Z99, ZM, ZSS, BF1, XX90, XX95, ZERO, ONE, TWO
      REAL THREE, SQRTH, QTR, TH, SMALL, PI6, STQR
      REAL SUMM2, SSUMM2, FAC, RSN, AN, AN25, A1, A2, DELTA, RANGE
      REAL SA, SX, SSX, SSA, SAX, ASA, XSX, SSASSX, W1, Y, XX, XI
      REAL GAMMA, M, S, LD, BF, Z90F, Z95F, Z99F, ZFM, ZSD, ZBAR
C
C        Auxiliary routines
C
      REAL PPND, ALNORM, POLY
C
      INTEGER NCENS, NN2, I, I1, J
      LOGICAL INIT, UPPER
C
      DATA C1 /0.0E0, 0.221157E0, -0.147981E0, -0.207119E1,
     *     0.4434685E1, -0.2706056E1/
      DATA C2 /0.0E0, 0.42981E-1, -0.293762E0, -0.1752461E1,
     *     0.5682633E1, -0.3582633E1/
      DATA C3 /0.5440E0, -0.39978E0, 0.25054E-1, -0.6714E-3/
      DATA C4 /0.13822E1, -0.77857E0, 0.62767E-1, -0.20322E-2/
      DATA C5 /-0.15861E1, -0.31082E0, -0.83751E-1, 0.38915E-2/
      DATA C6 /-0.4803E0, -0.82676E-1, 0.30302E-2/
      DATA C7 /0.164E0, 0.533E0/
      DATA C8 /0.1736E0, 0.315E0/
      DATA C9 /0.256E0, -0.635E-2/
      DATA G  /-0.2273E1, 0.459E0/
      DATA Z90, Z95, Z99 /0.12816E1, 0.16449E1, 0.23263E1/
      DATA ZM, ZSS /0.17509E1, 0.56268E0/
      DATA BF1 /0.8378E0/, XX90, XX95 /0.556E0, 0.622E0/
      DATA ZERO /0.0E0/, ONE/1.0E0/, TWO/2.0E0/, THREE/3.0E0/
      DATA SQRTH /0.70711E0/, QTR/0.25E0/, TH/0.375E0/, SMALL/1E-19/
      DATA PI6 /0.1909859E1/, STQR/0.1047198E1/, UPPER/.TRUE./
C
      PW  =  ONE
      IF (W .GE. ZERO) W = ONE
      AN = N
      IFAULT = 3
      NN2 = N/2
      IF (N2 .LT. NN2) RETURN
      IFAULT = 1
      IF (N .LT. 3) RETURN
C
C        If INIT is false, calculates coefficients for the test
C
      IF (.NOT. INIT) THEN
         IF (N .EQ. 3) THEN
            A(1) = SQRTH
         ELSE
            AN25 = AN + QTR
            SUMM2 = ZERO
            DO 30 I = 1, N2
               A(I) = PPND((I - TH)/AN25)
```

```
                  SUMM2 = SUMM2 + A(I) ** 2
   30          CONTINUE
               SUMM2 = SUMM2 * TWO
               SSUMM2 = SQRT(SUMM2)
               RSN = ONE / SQRT(AN)
               A1 = POLY(C1, 6, RSN) - A(1) / SSUMM2
   C
   C        Normalize coefficients
   C
               IF (N .GT. 5) THEN
                  I1 = 3
                  A2 = -A(2)/SSUMM2 + POLY(C2,6,RSN)
                  FAC = SQRT((SUMM2 - TWO * A(1) ** 2 - TWO *
     *                 A(2) ** 2)/(ONE - TWO * A1 ** 2 - TWO * A2 ** 2))
                  A(1) = A1
                  A(2) = A2
               ELSE
                  I1 = 2
                  FAC = SQRT((SUMM2 - TWO * A(1) ** 2)/
     *                    (ONE - TWO * A1 ** 2))
                  A(1) = A1
               END IF
               DO 40 I = I1, NN2
                  A(I) = -A(I)/FAC
   40          CONTINUE
            END IF
            INIT = .TRUE.
         END IF
         IF (N1 .LT. 3) RETURN
         NCENS = N - N1
         IFAULT = 4
         IF (NCENS .LT. 0 .OR. (NCENS .GT. 0 .AND. N .LT. 20)) RETURN
         IFAULT = 5
         DELTA = FLOAT(NCENS)/AN
         IF (DELTA .GT. 0.8) RETURN
   C
   C        If W input as negative, calculate significance level of -W
   C
         IF (W .LT. ZERO) THEN
            W1 = ONE + W
            IFAULT = 0
            GOTO 70
         END IF
   C
   C        Check for zero range
   C
         IFAULT = 6
         RANGE = X(N1) - X(1)
         IF (RANGE .LT. SMALL) RETURN
   C
   C        Check for correct sort order on range - scaled X
   C
         IFAULT = 7
         XX = X(1)/RANGE
         SX = XX
         SA = -A(1)
         J = N - 1
         DO 50 I = 2, N1
            XI = X(I)/RANGE
            IF (XX-XI .GT. SMALL) PRINT *,' ANYTHING'
            SX = SX + XI
```

```
            IF (I .NE. J) SA = SA + SIGN(1, I - J) * A(MIN(I, J))
            XX = XI
            J = J - 1
   50    CONTINUE
         IFAULT = 0
         IF (N .GT. 5000) IFAULT = 2
   C
   C        Calculate W statistic as squared correlation
   C        between data and coefficients
   C
         SA = SA/N1
         SX = SX/N1
         SSA = ZERO
         SSX = ZERO
         SAX = ZERO
         J = N
         DO 60 I = 1, N1
            IF (I .NE. J) THEN
               ASA = SIGN(1, I - J) * A(MIN(I, J)) - SA
            ELSE
               ASA = -SA
            END IF
            XSX = X(I)/RANGE - SX
            SSA = SSA + ASA * ASA
            SSX = SSX + XSX * XSX
            SAX = SAX + ASA * XSX
            J = J - 1
      60 CONTINUE
   C
   C        W1 equals (1-W) claculated to avoid excessive rounding error
   C        for W very near 1 (a potential problem in very large samples)
   C
         SSASSX = SQRT(SSA * SSX)
         W1 = (SSASSX - SAX) * (SSASSX + SAX)/(SSA * SSX)
      70 W = ONE - W1
   C
   C        Calculate significance level for W (exact for N=3)
   C
         IF (N .EQ. 3) THEN
             PW = PI6 * (ASIN(SQRT(W)) - STQR)
             RETURN
         END IF
         Y = LOG(W1)
         XX = LOG(AN)
         M = ZERO
         S = ONE
         IF (N .LE. 11) THEN
            GAMMA = POLY(G, 2, AN)
            IF (Y .GE. GAMMA) THEN
               PW = SMALL
               RETURN
            END IF
            Y = -LOG(GAMMA - Y)
            M = POLY(C3, 4, AN)
            S = EXP(POLY(C4, 4, AN))
         ELSE
            M = POLY(C5, 4, XX)
            S = EXP(POLY(C6, 3, XX))
         END IF
         IF (NCENS .GT. 0) THEN
   C
```

```
C        Censoring by proportion NCENS/N.  Calculate mean and sd
C        of normal equivalent deviate of W.
C
         LD = -LOG(DELTA)
         BF = ONE + XX * BF1
         Z90F = Z90 + BF * POLY(C7, 2, XX90 ** XX) ** LD
         Z95F = Z95 + BF * POLY(C8, 2, XX95 ** XX) ** LD
         Z99F = Z99 + BF * POLY(C9, 2, XX) ** LD
C
C        Regress Z90F,...,Z99F on normal deviates Z90,...,Z99 to get
C        pseudo-mean and pseudo-sd of z as the slope and intercept
C
         ZFM = (Z90F + Z95F + Z99F)/THREE
         ZSD = (Z90*(Z90F-ZFM)+Z95*(Z95F-ZFM)+Z99*(Z99F-ZFM))/ZSS
         ZBAR = ZFM - ZSD * ZM
         M = M + ZBAR * S
         S = S * ZSD
      END IF
      PW = ALNORM((Y - M)/S, UPPER)
C
      RETURN
      END
```

```
      subroutine wext(x, n, ssq, a, n2, eps, w, pw, ifault)
c
c        Algorithm AS 181   Appl. Statist. (1982) Vol. 31, No. 2
c
c        Calculates Shapiro and Wilk's W statistic and its sig. level
c
c     Auxiliary routines required: ALNORM = algorithm AS 66 and NSCOR2
c     from AS 177.
c
      real x(n), a(n2), lamda, wa(3), wb(4), wc(4), wd(6), we(6), wf(7),
     *  c1(5, 3), c2(5, 3), c(5), unl(3), unh(3)
      integer nc1(3), nc2(3)
      logical upper
      data wa(1), wa(2), wa(3)
     *  /0.118898, 0.133414, 0.327907/,
     *     wb(1), wb(2), wb(3), wb(4)
     *  /-0.37542, -0.492145, -1.124332, -0.199422/,
     *     wc(1), wc(2), wc(3), wc(4)
     *  /-3.15805, 0.729399, 3.01855, 1.558776/,
     *     wd(1), wd(2), wd(3), wd(4), wd(5), wd(6)
     *  /0.480385, 0.318828, 0.0, -0.0241665, 0.00879701, 0.002989646/,
     *     we(1), we(2), we(3), we(4), we(5), we(6)
     *  /-1.91487, -1.37888, -0.04183209, 0.1066339, -0.03513666,
     *   -0.01504614/,
     *     wf(1), wf(2), wf(3), wf(4), wf(5), wf(6), wf(7)
     *  /-3.73538, -1.015807, -0.331885, 0.1773538, -0.01638782,
     *   -0.03215018, 0.003852646/
      data c1(1,1), c1(2,1), c1(3,1), c1(4,1), c1(5,1),
     *     c1(1,2), c1(2,2), c1(3,2), c1(4,2), c1(5,2),
     *     c1(1,3), c1(2,3), c1(3,3), c1(4,3), c1(5,3) /
     *     -1.26233, 1.87969, 0.0649583, -0.0475604, -0.0139682,
     *     -2.28135, 2.26186, 0.0, 0.0, -0.00865763,
     *     -3.30623, 2.76287, -0.83484, 1.20857, -0.507590/
      data c2(1,1), c2(2,1), c2(3,1), c2(4,1), c2(5,1),
     *     c2(1,2), c2(2,2), c2(3,2), c2(4,2), c2(5,2),
     *     c2(1,3), c2(2,3), c2(3,3), c2(4,3), c2(5,3) /
     *     -0.287696, 1.78953, -0.180114, 0.0, 0.0,
     *     -1.63638, 5.60924, -3.63738, 1.08439, 0.0,
     *     -5.991908, 21.04575, -24.58061, 13.78661, -2.835295/
      data unl(1), unl(2), unl(3) /-3.8,-3.0, -1.0/,
     *     unh(1), unh(2), unh(3) / 8.6, 5.8,  5.4/
      data nc1(1), nc1(2), nc1(3) /5, 5, 5/,
     *     nc2(1), nc2(2), nc2(3) /3, 4, 5/
      data pi6 /1.90985932/, stqr /1.04719755/, upper /.true./,
     *  zero/0.0/, tqr /0.75/, one /1.0/, onept4 /1.4/, three/3.0/,
     *  five/5.0/
c
      ifault = 1
      pw = one
      w = one
      if(n.le.2) return
      ifault = 3
      if(n/2 .ne. n2) return
      ifault = 2
      if(n.gt.2000) return
c
c        Calculate W
c
      ifault = 0
      w = zero
      an = n
```

```
      i = n
      do 10 j = 1,n2
        w = w+a(j) * (x(i)-x(j))
        i = i-1
   10 continue
      w = w * w / ssq
      if (w .lt. one) goto 20
      w = one
      return
c
c        Get significance level of W
c
   20 if (n .le. 6) goto 100
c
c        N between 7 and 2000 ... Transform W to Y, get mean and sd,
c        standardize and get significance level
c
      if(n.gt.20) goto 30
      al = log(an) - three
      lamda  =  poly(wa,3,al)
      ybar = exp(poly(wb,4,al))
      sdy = exp(poly(wc,4,al))
      goto 40
   30 al = log(an) - five
      lamda = poly(wd,6,al)
      ybar = exp(poly(we,6,al))
      sdy = exp(poly(wf,7,al))
   40 y  =  (one-w) ** lamda
      z  =  (y - ybar) / sdy
      pw  =  alnorm(z,upper)
      return
c
c        Deal with N less than 7 (Exact significance level for N = 3).
c
  100 if(w .le. eps) goto 160
      ww = w
      if(n.eq.3) goto 150
      un = log((w-eps)/(one-w))
      n3 = n-3
      if(un .lt. unl(n3)) goto 160
      if (un .ge. onept4) goto 120
      nc = nc1(n3)
      do 110 i = 1,nc
  110 c(i) = c1(i,n3)
      eu3 = exp(poly(c,nc,un))
      goto 140
  120 if (un .gt. unh(n3)) return
      nc = nc2(n3)
      do 130 i = 1,nc
  130 c(i) = c2(i,n3)
      un = log(un)
      eu3 = exp(exp(poly(c,nc,un)))
  140 ww = (eu3+tqr)/(one+eu3)
  150 pw = pi6 *(atan(sqrt(ww/(one-ww)))-stqr)
      return
  160 pw = zero
      return
      end
c
c
      subroutine wcoef(a, n, n2, eps, ifault)
```

```
c
c       Algorithm AS 181.1   Appl. Statist.  (1982) Vol. 31, No. 2
c
c       Obtain array A of weights for calculating W
c
      real a(n2), c4(2), c5(2), c6(3)
      data c4(1), c4(2) /0.6869, 0.1678/, c5(1), c5(2) /0.6647, 0.2412/,
     *  c6(1), c6(2), c6(3) /0.6431, 0.2806, 0.0875/
      data rsqrt2 /0.70710678/, zero/0.0/, half/0.5/, one/1.0/,
     *  two/2.0/, six/6.0/, seven/7.0/, eight/8.0/, thirt /13.0/
      ifault = 1
      if(n.le.2) return
      ifault = 3
      if(n/2 .ne. n2) return
      ifault = 2
      if(n.gt.2000) return
      ifault = 0
      if(n .le. 6) goto 30
c
c       N .GT. 6   Calculate rankits using approximate routine nscor2
c                  (AS177)
c
      call nscor2(a, n, n2, ifault)
      sastar = zero
      do 10 j = 2,n2
   10 sastar = sastar+a(j)*a(j)
      sastar = sastar*eight
      nn = n
      if (n .le. 20) nn = nn-1
      an = nn
      a1sq = exp(log(six*an+seven)-log(six*an+thirt)
     *  +half*(one+(an-two)*log(an+one)-(an-one)
     *  *log(an+two)))
      a1star = sastar/(one/a1sq-two)
      sastar = sqrt(sastar+two*a1star)
      a(1) = sqrt(a1star)/sastar
      do 20 j = 2,n2
   20 a(j) = two*a(j)/sastar
      goto 70
c
c       n.le.6   Use exact values for weiughts
c
   30 a(1) = rsqrt2
      if(n.eq.3) goto 70
      n3 = n-3
      goto (40,50,60), n3
   40 do 45 j = 1,2
   45 a(j) = c4(j)
      goto 70
   50 do 55 j = 1,2
   55 a(j) = c5(j)
      goto 70
   60 do 65 j = 1,3
   65 a(j) = c6(j)
c
c       Calculate the minimum possible value of W
c
   70 eps = a(1)*a(1)/(one-one/float(n))
      return
      end
c
```

```
c
      function poly(c, nord, x)
c
c
c        Algorithm AS 181.2   Appl. Statist.  (1982) Vol. 31, No. 2
c
c        Calculates the algebraic polynomial of order nored-1 with
c        array of coefficients c.  Zero order coefficient is c(1)
c
      real c(nord)
      poly = c(1)
      if(nord.eq.1) return
      p = x*c(nord)
      if(nord.eq.2) goto 20
      n2 = nord-2
      j = n2+1
      do 10 i = 1,n2
      p = (p+c(j))*x
      j = j-1
   10 continue
   20 poly = poly+p
      return
      end
c
c
c
      subroutine wgp(x, n, ssq, gp, h, a, n2, eps, w, u, p, ifault)
c
c        AS R63 Appl. Statist. (1986) Vol. 35, No.2
c
c        A remark on AS 181
c
c        Calculates Sheppard corrected version of W test.
c
c     Auxiliary functions required: ALNORM = algorithm AS 66, and
c     PPND = algorithm AS 111 (or PPND7 from AS 241).
c
      real x(n), a(n2)
      data twelve /1.2e01/, ten/1.0e1/, five/5.0e0/, one/1.0e0/,
     *     zero/0.0e0/
c
      zbar = zero
      zsd = one
      ifault = 1
      if (n.lt.7) return
c
c        No correction applied if gp=0.
c
      if (gp .le. zero) goto 10
      an1 = float(n-1)
c
c        correct ssq and find standardized grouping interval (h)
c
      ssq = ssq - an1 * gp * gp / twelve
      h = gp / sqrt(ssq / an1)
      ifault = 4
      if (h .gt. 1.5) return
   10 call wext(x, n, ssq, a, n2, eps, w, p, ifault)
      if (ifault .ne. 0) return
      if (p .gt. zero .and. p .lt. one) goto 20
      u = five - ten * p
```

```
      return
   20 if (gp .le. zero) goto 30
c
c        correct u for grouping interval (n .le. 100 and n .gt. 100
c        separately)
c
      hh = sqrt(h)
      if (n .gt. 100) goto 25
      zbar = -h*(1.07457 + hh*(-2.8185 + hh*1.8898))
      zsd = one + h*(0.50933 + hh*(-0.98305 + hh*0.7408))
      goto 30
   25 zbar = -h*(0.96436 + hh*(-2.1300 + hh*1.3196))
      zsd = one + h*(0.2579 + h*0.15225)
c
c        ppnd is AS 111 (Beasley and Springer, 1977)
c
   30 u = (-ppnd(p, ifppnd) - zbar) / zsd
c
c       alnorm is AS 66 (Hill, 1973)
c
      p = alnorm(u, .true.)
      return
      end
```

```
      SUBROUTINE INTBD(G, N, E, ET, DELTA, ITAIL)
C
C        ALGORITHM AS R96 APPL. STATIST. (1996) VOL.45, NO.2
C
C        Interval for trapezoidal rule integration to limit error to
C        E using moment generating function bound for weighted sum of
C        chi-squared(1) random variables
C
      INTEGER N, ITAIL
      REAL DELTA, G(N), E, ET
C
      INTEGER I
      REAL ZERO, TWO, POINT1, POINT5, TWOPI, ULIM, PSI, DPSI, D2PSI,
     *     HB, U, U1, U2, ELOG, EPS, HM, HSD, SGN, ZABS, ZEXP, ZFLOAT,
     *     ZLOG, ZMAX1, ZMIN1, ZSQRT
C
      EXTERNAL LMGFEV
C
      DATA ZERO, TWO, POINT1, POINT5 /0.0E0, 2.0E0, 0.1E0, 0.5E0/
      DATA TWOPI /6.28318530717959E0/
      DATA EPS /0.09531E0/
C
      ZABS(U) = ABS(U)
      ZEXP(U) = EXP(U)
      ZFLOAT(I) = FLOAT(I)
      ZLOG(U) = ALOG(U)
      ZMAX1(U, U1) = AMAX1(U, U1)
      ZMIN1(U, U1) = AMIN1(U, U1)
      ZSQRT(U) = SQRT(U)
C
C        Tail check
C
      SGN = ZFLOAT(ITAIL)
      IF (ITAIL .EQ. 1) THEN
         IF (G(N) .LE. ZERO) RETURN
         ULIM = POINT5 / G(N)
      ELSE
         IF (G(1) .GE. ZERO) RETURN
         ULIM = POINT5 / G(1)
      ENDIF
      HM = ZERO
      HSD = ZERO
      DO 10 I = 1, N
         HM = HM + G(I)
         HSD = HSD + G(I) ** 2
   10 CONTINUE
C
C        Newton search for zero log(m.g.f.) derivative
C
      HSD = ZSQRT(HSD)
      U1 = ZERO
      U2 = ZERO
      IF (SGN * HM .LT. ZERO .AND. -SGN * HM / HSD .GT. TWO) THEN
         DPSI = HM
   20    IF (DPSI * SGN .LT. ZERO) THEN
            U1 = U2
            U2 = POINT5 * (U2 + ULIM)
            CALL LMGFEV (U2, G, N, PSI, DPSI, D2PSI)
            GOTO 20
         ENDIF
   25    IF (DPSI ** 2 .GE. POINT1 * D2PSI) THEN
```

```
               U2 = U2 - DPSI / D2PSI
               CALL LMGFEV(U2, G, N, PSI, DPSI, D2PSI)
            ENDIF
            IF (ZEXP(PSI) .LE. E) RETURN
         ENDIF
C
C        Bound equality
C
         ELOG = - ZLOG(ET)
         CALL LMGFEV(U2, G, N, PSI, DPSI, D2PSI)
         HB = PSI - U2 * DPSI + ELOG
      30 IF (HB .GT. ZERO) THEN
            U1 = U2
            U2 = POINT5 * (U2 + ULIM)
            CALL LMGFEV(U2, G, N, PSI, DPSI, D2PSI)
            HB = PSI - U2 * DPSI + ELOG
            GOTO 30
         ENDIF
C
C        Newton/binary search solution
C
         U = U2
      40 IF (ZABS(HB) .GE. EPS) THEN
            U = U + HB / (U * D2PSI)
            IF (U .LE. ZMIN1(U1, U2) .OR. U .GT. ZMAX1(U1, U2))
     *         U = POINT5 * (U1 + U2)
            CALL LMGFEV(U, G, N, PSI, DPSI, D2PSI)
            HB = PSI - U * DPSI + ELOG
            IF (HB .GT. ZERO) THEN
               U1 = U
            ELSE
               U2 = U
            ENDIF
            GOTO 40
         ENDIF
         IF (ITAIL .EQ. 1 .AND. DPSI .GT. ZERO) THEN
            DELTA = TWOPI / DPSI
            ITAIL = 0
         ELSEIF (ITAIL .EQ. -1 .AND. DPSI .LT. ZERO) THEN
            DELTA = - TWOPI / DPSI
            ITAIL = 0
         ENDIF
C
         RETURN
         END
```

```
      SUBROUTINE ULTRA (BW,M,MORD,MUE,N,NUE,OPTIO1,OPTIO2,OPTIO3,
     *                  XIN,XOU,YIN,IWKAR,WKAR,IFAULT,W2,YOU)
C
C        ALGORITHM AS297.1 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Nonparametric regression using ultraspherical
C        polynomials
C
      INTEGER I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12,
     *        I13, II1, II2, IFLG, IFLG1, IL, ILB, IR, IRB, IZ, M,
     *        IWKAR(2*M), MORD, N, IFAULT, NUE, OPTIO1, OPTIO2, OPTIO3
      DOUBLE PRECISION BW, PHIJU, SUMW2, XA, XAMB, XAMBN, XAPB, XB, XC,
     *                 XX, XIN(N), XOU(M), W2(M),YIN(N), YOU(M), YOUT,
     *                 YOUT1, WKAR(60*N+2000)
C
      EXTERNAL BOUND, ERROR, INIT, PHIJ, SAVE, WEIGHT
C
C        Error checks
C
      CALL ERROR (BW,M,MORD,MUE,N,NUE,OPTIO1,OPTIO2,XIN,XOU,IFAULT)
      IF (IFAULT.EQ.0.OR.IFAULT.EQ.7) GO TO 5
      GO TO 200
C
C        Convert XIN(N) and XOU(M) to interval [-1,1]
C
    5 XA = XIN(1)-(XIN(2)-XIN(1))/2.0D0
      XB = XIN(N)+(XIN(N)-XIN(N-1))/2.0D0
      XAMB = XB-XA
      XAPB = XA+XB
      DO 10 I=1, N
         XIN(I) = 2.0D0*XIN(I)/XAMB-XAPB/XAMB
   10 CONTINUE
      DO 15 I=1, M
         XOU(I) = 2.0D0*XOU(I)/XAMB-XAPB/XAMB
   15 CONTINUE
C
C        Store boundaries of windows and corresponding indices
C
      I0 = 1
      I1 = I0+N
      I2 = I1+2*M
      CALL BOUND (BW,M,N,OPTIO1,OPTIO2,XIN,XOU,IWKAR(I0),
     *            ILB,IRB,WKAR(I1),WKAR(I2))
C
C        Calculate the initial values
C
      I3 = I2+N+1
      I4 = I3+MUE+1
      I5 = I4+(MUE+1)*(MUE+1)
      I6 = I5+(MORD+1)*(MORD+1)
      I7 = I6+MORD+1
      CALL INIT(WKAR(I3),BW,MORD,MUE,WKAR(I4),WKAR(I5),WKAR(I6),
     *          WKAR(I7))
      I8 = I7+MUE+1
      I9 = I8+N+1
      I10 = I9+MORD+1
      I11 = I10+(N+1)*(MORD+2*MUE+1)
      I12 = I11+MORD+1
      I13 = I12+N*(MORD+1)
C
C        Calculations for the:
```

```
C           1 - left boundary (III = 1)
C           2 - right boundary (III = 2)
C           3 - interior (III = 3)
C
      DO  120 III=1, 3
         IF (III.EQ.1.AND.ILB.LE.0) GO TO 120
         IF (III.EQ.2.AND.IRB.LE.0) GO TO 120
         IF (III.EQ.3.AND.ILB.GT.IRB-2) GO TO 120
         IF (III.EQ.3) GO TO 30
         IL = IWKAR(1+(M-1)*(III-1))
         IR = IWKAR(M+1+(M-1)*(III-1))
         IZ = IR-IL
         IF (IZ.LE.0) GO TO 120
         DO 20 I=IL, IR
            WKAR(I8-IL+I) = WKAR(I2+I-1)
   20    CONTINUE
         WKAR(I8) = WKAR(N+1+(M-1)*(III-1))
         WKAR(I8+IZ) = WKAR(N+M+1+(M-1)*(III-1))
         IF (III.EQ.2) GO TO 25
         II1 = 1
         II2 = ILB
         IFLG = 0
         IFLG1 = 0
         XC = WKAR(N+1)+BW
         GO TO 40
   25    IF (ILB.EQ.IRB) IRB = IRB +1
         II1 = IRB
         II2 = M
         IFLG = 0
         IFLG1 = 0
         XC = WKAR(N+2*M)-BW
         GO TO 40
   30    IF ((ILB+1).EQ.IRB) GO TO 120
         II1 = ILB+1
         II2 = IRB -1
   40    DO 110 II=II1, II2
            XX = XOU(II)
            IF (III.NE.3) GO TO 55
            XC = XX
            IL = IWKAR(II)
            IR = IWKAR(M+II)
            IZ = IR-IL
            IF (IZ.LE.0) GO TO 110
            DO 50 I=IL, IR
               WKAR(I8-IL+I) = WKAR(I2+I-1)
   50       CONTINUE
            WKAR(I8) = WKAR(N+II)
            WKAR(I8+IZ) = WKAR(N+M+II)
            GO TO 60
   55       IF (IFLG.EQ.1) GO TO 65
   60       CALL SAVE (BW,WKAR(I5),WKAR(I6),IZ,MORD,MUE,WKAR(I7),
     *                  N,WKAR(I8),XC,WKAR(I10),WKAR(I12))
            IFLG = 1
C
C        Calculate array PHIAR(MORD+1)
C
   65       DO 70 JJ=0, MORD
               CALL PHIJ (BW,WKAR(I5),JJ,WKAR(I6),MORD,NUE,XC,XX,
     *                    WKAR(I9),PHIJU)
               WKAR(I11+JJ) = PHIJU
   70       CONTINUE
```

```
            YOUT = 0.0D0
            IF (III.EQ.3.AND.OPTIO1.EQ.1) GO TO 130
            IF (OPTIO3.EQ.0) GO TO 80
C
C        Sum of squared weights is requested (OPTIO3 = 1)
C
            CALL WEIGHT (IZ,MORD,N,NUE,WKAR(I11),WKAR(I12),SUMW2,
     *                   WKAR(I0))
            DO 75 I=1, IZ
               YOUT = YOUT+WKAR(I0+I-1)*YIN(IL+I-1)
   75       CONTINUE
            YOU(II) = YOUT
            W2(II) = SUMW2
            GO TO 110
C
C        Sum of squared weights not requested (OPTIO3 = 0)
C
   80       IF (IFLG1.EQ.1.AND.III.EQ.1) GO TO 100
            IF (IFLG1.EQ.1.AND.III.EQ.2) GO TO 100
            DO 95 J=NUE, MORD
               YOUT1 = 0.0D0
               DO 90 I=1, IZ
                  YOUT1 = YOUT1+YIN(IL+I-1)*WKAR(I12+I-1+N*J)
   90          CONTINUE
               WKAR(I13+J) = YOUT1
   95       CONTINUE
            IFLG1 = 1
  100       DO 105 J=NUE, MORD
               YOUT = YOUT+WKAR(I13+J)*WKAR(I11+J)
  105       CONTINUE
            YOU(II) = YOUT
  110    CONTINUE
  120 CONTINUE
      GO TO 160
C
C        Calculation for the interior in the equidistant case
C
  130 CALL WEIGHT (IZ,MORD,N,NUE,WKAR(I11),WKAR(I12),SUMW2,
     *             WKAR(I0))
      DO 150 II=II1, II2
         YOUT = 0.0D0
         IL = IWKAR(II)
         DO 140 I=1, IZ
            YOUT = YOUT+WKAR(I0+I-1)*YIN(IL+I-1)
  140    CONTINUE
         YOU(II) = YOUT
         W2(II) = SUMW2
  150 CONTINUE
C
C        Convert XIN(N), XOU(M) and YOU(M) to the original scales
C
  160 CONTINUE
      DO 170 I=1, N
         XIN(I) = (XAMB*XIN(I)+XAPB)/2.0D0
  170 CONTINUE
      XAMBN = 1.0D0
      IF (NUE.EQ.0) GO TO 185
      DO 180 I=1, NUE
         XAMBN = (2.0D0/XAMB)*XAMBN
  180 CONTINUE
  185 DO 190 I=1, M
```

```
            XOU(I) = (XAMB*XOU(I)+XAPB)/2.0D0
            YOU(I) = XAMBN*YOU(I)
  190 CONTINUE
  200 CONTINUE
       RETURN
       END
C
      SUBROUTINE ERROR (BW,M,MORD,MUE,N,NUE,OPTIO1,OPTIO2,XIN,
     *                  XOU,IFAULT)
C
C        ALGORITHM AS297.2 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Error checking
C
      INTEGER M, MO, MORD, MUE, N, IFAULT, NU, NUE, OPTIO1, OPTIO2
      DOUBLE PRECISION BW, ERR, X1, X2, XIN(N), XOU(M)
C
      IFAULT = 0
C
C        Error code messages
C
      IF (N.GE.4.AND.M.GE.1) GO TO 10
      IFAULT = 1
   10 IF (MORD.LE.30.AND.MUE.LE.10.AND.NUE.LE.3.AND.NUE.LE.MORD)
     *   GO TO 20
      IFAULT = 2
   20 DO 30 I=1, N-1
         IF (XIN(I).LE.XIN(I+1)) GO TO 30
         IFAULT = 3
   30 CONTINUE
      DO 40 I=1, M-1
         IF (XOU(I).LE.XOU(I+1)) GO TO 40
         IFAULT = 4
   40 CONTINUE
      IF (BW.LE.0.AND.OPTIO2.NE.1)  IFAULT=5
      IF (OPTIO1.NE.1) GO TO 70
      ERR = 0.00001D0
      DO 50 I=1, N
         IF (DABS(XOU(I)-XIN(I)).LT.ERR) GO TO 50
         IFAULT = 6
   50 CONTINUE
      DO 60 I=1, N-2
         X1 = XIN(I+1)-XIN(I)
         X2 = XIN(I+2)-XIN(I+1)
         IF (DABS(X1-X2).LT.ERR) GO TO 60
         IFAULT = 6
   60 CONTINUE
   70 IF (IFAULT.NE.0) GO TO 80
C
C        Warning message (error code 7)
C
      MO = (MORD/2)*2
      NU = (NUE/2)*2
      IF (MO.EQ.MORD.AND.NU.EQ.NUE.OR.MO.NE.MORD.AND.NU.NE.NUE)
     *   GO TO 80
      IFAULT = 7
   80 CONTINUE
      RETURN
      END
C
      SUBROUTINE BOUND (BW,M,N,OPTIO1,OPTIO2,XIN,XOU,IBND,ILB,
```

```
     *                  IRB,BND,S)
C
C        ALGORITHM AS297.3 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculate left and right boundaries and indices of windows
C        midpoints
C
      INTEGER IL, IR, IZ, ILB, IRB, M,
     *        IBND(M,2), N, OPTIO1, OPTIO2
      DOUBLE PRECISION B1, B11, B2, B22, BND(M,2), BW, S(N+1),
     *                 XIN(N), XOU(M)
C
      S(1) = XIN(1)-(XIN(2)-XIN(1))/2.0D0
      S(N+1) = XIN(N)+(XIN(N)-XIN(N-1))/2.0D0
      DO 10 I=1, N-1
         S(I+1) = (XIN(I+1)+XIN(I))/2.0D0
   10 CONTINUE
      IF (OPTIO2.EQ.1) BW = (S(N+1)-S(1))/2.0D0
      B1 = S(1)+BW
      B11 = B1+BW
      B2 = S(N+1)-BW
      B22 = B2-BW
C
C        Find the indices of the boundaries for the output points
C
       IF (OPTIO1.EQ.0) GO TO 15
C
C        Equidistant case
C
      IZ = BW/(XIN(2)-XIN(1))
      ILB = 1+IZ
      IRB = N-IZ
      GO TO 25
C
C        Non-equidistant case
C
   15 DO 20 I=1, M
         IF (XOU(I).LE.B1) ILB = I
         IF (XOU(M-I+1).GE.B2) IRB = M-I+1
   20 CONTINUE
C
C        Find the indices of the boundaries for the input points
C
       IF (OPTIO1.EQ.0) GO TO 30
C
C        Equidistant case
C
   25 IZ = INT(2*BW/(XIN(2)-XIN(1)))
      IL = 1+IZ
      IR = N+1-IZ
      GO TO 40
C
C        Non-equidistant case
C
   30 DO 35 I=1, N
         IF (S(I).LT.B11)    IL = I
         IF (S(N-I+2).GT.B22) IR = N-I+2
   35 CONTINUE
      IL = IL+1
      IF (IR.GT.1) IR = IR-1
      IZ = IR-IL
```

```
C
C        Record the boundaries and boundary indices
C
   40 IF (XOU(1).GT.B1) GO TO 45
      IBND(1,1) = 1
      IBND(1,2) = IL
      BND(1,1) = S(1)
      BND(1,2) = B11
   45 IF (XOU(M).LT.B2) GO TO 50
      IBND(M,1) = IR
      IBND(M,2) = N+1
      BND(1,1) = S(1)
      BND(1,2) = B11
      BND(M,1) = B22
      BND(M,2) = S(N+1)
   50 IF (OPTIO2.EQ.1) GO TO 110
      IF (B1.EQ.B2) GO TO 100
      IF (XOU(M).LT.B2) IRB = M+1
      IF (ILB.GT.IRB-2) GOTO 110
      DO 90 I=ILB+1, IRB-1
         XL = XOU(I)-BW
         XR = XOU(I)+BW
C
C        Computation of relevant indices
C
         IF (OPTIO1.EQ.0) GO TO 55
C
C        Equidistant case
C
         IZ = INT(BW/(XIN(2)-XIN(1)))
         IL = I-IZ
         IR = I+IZ+1
         GO TO 85
C
C        Non-equidistant case
C
   55    IL = 1
   60    IR = IL
         IF (S(IL).GT.XL) GO TO 65
         IL = IL+1
         GO TO 60
   65    IR = IL-1
   70    IR = IR+1
         IF (IR.EQ.N+1) GO TO 80
         IF (S(IR).LT.XR) GO TO 70
   80    IF (IL.GT.1)  IL = IL-1
         IZ = IR-IL
C
C        Record the indices
C
   85    IBND(I,1) = IL
         IBND(I,2) = IR
         BND(I,1) = XL
         BND(I,2) = XR
   90 CONTINUE
      GO TO 110
  100 OPTIO2 = 1
  110 RETURN
      END
C
      SUBROUTINE INIT (BIN,BW,MORD,MUE,WKM1,COEF,HJAR,M1BIN)
```

```
C
C       ALGORITHM AS297.4 APPL. STATIST. (1995) VOL.44, NO.2
C
C       Computes necessary constants
C
      INTEGER JM, MORD, MUE
      DOUBLE PRECISION AA, BB, BCD, BIN(MUE+1), BW, CC,
     *                 COEF(MORD+1,MORD+1), DD, EE, FF, HJAR(MORD+1),
     *                 HJMU, M1BIN(MUE+1), MUP1, POW2, RR,
     *                 WKM1(MUE+1,MUE+1)
C
C       Calculate multiple of binomial coefficients
C
      EXTERNAL BINOM, HJ, RGAMMA
      INTRINSIC DSQRT
C
      CALL BINOM (MUE,WKM1,BIN)
      DO 5 K=0, MUE
         M1BIN(K+1) = ((-1)**K)*BIN(K+1)
    5 CONTINUE
C
C       Calculate HJAR(MORD+1)
C
      MUP1 = (1.0D0)*MUE+1.0D0
      POW2 = 1.0D0
      DO 10 I=1,2*MUE+1
         POW2 = 2.0D0*POW2
   10 CONTINUE
      POW2 = DSQRT(POW2)
      CALL RGAMMA (2*MUP1-1,MUP1,RR)
      DO 15 JJ=0, MORD
         CALL HJ (JJ,MUE,HJMU)
         HJAR(JJ+1) = HJMU*RR/POW2
   15 CONTINUE
C
C       Calculate COEF(MORD+1,MORD+1)
C
      DO 80 JJ=0, MORD
         DO 70 MM=0, JJ/2
            JM = JJ-2*MM
            IF (JJ.GE.3) GO TO 85
            BB = 1.0D0
            IF (MM.EQ.0) GO TO 30
            DO 20 I=1, MM
               BB = BB*(2.0D0*I-1)/(2.0D0*I)
   20       CONTINUE
   30       CC = 1.0D0
            DO 40 UU=1, MUE
               CC = CC*(2*MM+2*UU-1.0D0)/(2*UU-1.0D0)
   40       CONTINUE
            DD = 1.0D0
            IF (JM.EQ.0) GO TO 60
            DO 50 K=1, JM
               DD = DD*(2*MM+2*MUE+2*K-1.0D0)/FLOAT(K)
   50       CONTINUE
   60       DD = ((-1.0D0)**MM)*DD
            BCD = BB*CC*DD
            AA = 1.0D0
            COEF(JJ+1,MM+1) = AA*BCD*DSQRT(BW)
   70    CONTINUE
   80 CONTINUE
```

```
     85 IF (MORD.LT.3) GO TO 130
        DO 120 JJ=3, MORD
           DO 110 MM=0, JJ/2
              JM = JJ-2*MM
              EE = COEF(JJ,MM+1)
              IF (MM.EQ.0) GO TO 90
              FF = COEF(JJ-1,MM)
              GO TO 100
     90       FF = 0.0D0
    100       BCD = 2*((JJ+MUE-0.5D0)/FLOAT(JJ))*EE
       *            -((JJ+2*MUE-1.0D0)/FLOAT(JJ))*FF
              COEF (JJ+1,MM+1) = BCD
    110    CONTINUE
    120 CONTINUE
    130 RETURN
        END
C
        SUBROUTINE SAVE (BW,COEF,HJAR,IZ,MORD,MUE,M1BIN,N,S1,
       *                 XC,WKM2,RINT)
C
C       ALGORITHM AS297.5 APPL. STATIST. (1995) VOL.44, NO.2
C
C       Saves integrals for the boundaries given in array S1(IZ)
C
        INTEGER  IZ, JMK1, N, MORD, MO2, MUE
        DOUBLE PRECISION AA, BB, BW, CC, COEF(MORD+1,MORD+1),
       *                 HJAR(MORD+1), M1BIN(MUE+1), RINT(N,MORD+1),
       *                 S1(N+1), SUM, XC, WKM2(N+1,MORD+2*MUE+1)
C
C       Calculate the powers
C
        MO2 = MORD+2*MUE+1
        DO 30 I=1, IZ+1
           WKM2(I,1) = (S1(I)-XC)/BW
           IF (WKM2(I,1).GT.1.0D0) WKM2(I,1) = 1.0D0
           IF (WKM2(I,1).LT.-1.0D0) WKM2(I,1) = -1.0D0
           DO 20 J=2, MO2
              WKM2(I,J) = WKM2(I,1)*WKM2(I,J-1)
    20    CONTINUE
    30 CONTINUE
C
C       Calculate the integral
C
        DO 70 II=1, IZ
           DO 60 JJ=0, MORD
              AA = HJAR(JJ+1)
              SUM = 0.0D0
              DO 50 MM=0, JJ/2
                 BB = COEF(JJ+1,MM+1)
                 CC = 0.0D0
                 DO 40 K=0, MUE
                    JMK1 = JJ-2*MM+2*K+1
                    CC = CC+(M1BIN(K+1)*(WKM2(II+1,JMK1)
       *                     -WKM2(II,JMK1)))/FLOAT(JMK1)
    40           CONTINUE
                 SUM = SUM +BB*CC
    50       CONTINUE
             RINT(II,JJ+1) = AA*SUM
    60    CONTINUE
    70 CONTINUE
       RETURN
```

```
      END
C
      SUBROUTINE WEIGHT (IZ,MORD,N,NUE,PHIAR,RINT,SUMW2,W)
C
C        ALGORITHM AS297.6 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculate weights W(N) corresponding to estimating NUE-th
C        derivative at point XX using ultraspherical polynomial of
C        order MORD on the interval [XX-BW,XX+BW]
C
      INTEGER IZ, MORD, N, NUE
      DOUBLE PRECISION PHIAR(MORD+1), RINT(N,MORD+1), SUM,
     *                 SUMW2, W(N)
C
      SUMW2 = 0
      DO 10 I=1, N
         W(I) = 0
   10 CONTINUE
      DO 30 I=1, IZ
         SUM = 0.0D0
         DO 20 J=NUE, MORD
            SUM = SUM +PHIAR(J+1)*RINT(I,J+1)
   20    CONTINUE
         W(I) = SUM
         SUMW2 = SUMW2+SUM*SUM
   30 CONTINUE
      RETURN
      END
C
      SUBROUTINE PHIJ (BW,COEF,JJ,HJAR,MORD,NUE,XC,XX,WKAR1,PHIJU)
C
C        ALGORITHM AS297.7 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculate NUE-th derivative of the JJ-th ultraspherical
C        polynomial with degree of smoothness MUE, on the interval
C        [XC-BW,XC+BW], evaluated at point XX
C
      INTEGER JJ, JM, MORD, MM, NUE
      DOUBLE PRECISION AA, BB, BW, BWN, CC, CENX, COEF(MORD+1,MORD+1),
     *                 HJAR(MORD+1), HJMU, PHIJU, SUM, XC, XX,
     *                 WKAR1(MORD+1)
C
      CENX = (XX-XC)/BW
      WKAR1(1) = 1.0D0
      DO 10 I=2, JJ+1
         WKAR1(I) = CENX*WKAR1(I-1)
   10 CONTINUE
      BWN = 1.0D0
      DO 20 I=0, NUE
         BWN = BW*BWN
   20 CONTINUE
      SUM = 0.0D0
      DO 50 MM=0, (JJ-NUE)/2
         JM = JJ-2*MM
         BB = COEF(JJ+1,MM+1)
         IF (NUE.EQ.0) GO TO 40
         DO 30 KK=1, NUE
            BB = BB*(JM-KK+1)
   30    CONTINUE
   40    CC = WKAR1(JM-NUE+1)
         AA = BB*CC
```

```
            SUM = SUM+AA
      50 CONTINUE
         HJMU = HJAR(JJ+1)
         PHIJU = HJMU*SUM/BWN
         RETURN
         END
C
          SUBROUTINE HJ(JJ,MUE,HJMU)
C
C        ALGORITHM AS297.8 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculates part of the normalizing constant for
C        ultraspherical polynomials of order JJ, constant for
C        JJ-th polynomial
C
         INTEGER JJ, MUE
         DOUBLE PRECISION AA, BB, JJP1, JP1MU, HJMU
C
         EXTERNAL RGAMMA
         INTRINSIC DSQRT
C
         JJP1 = FLOAT(JJ+1)
         JP1MU = JJP1+2*MUE
         CALL RGAMMA (JJP1,JP1MU,BB)
         AA = JJ+JP1MU
         HJMU = DSQRT(AA*BB)
         RETURN
         END
C
         SUBROUTINE GAMMA (NN,GM)
C
C        ALGORITHM AS297.9 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculate gamma function at argument which is an
C        integer or integer + 0.5
C
         INTEGER N
         DOUBLE PRECISION ABSN, GM, PI, NN
C
         INTRINSIC DABS, DASIN, DSQRT
C
         PI = 2.0D0*DASIN(1.0D0)
         GM = 1.0D0
         IF (NN.EQ.0.5D0) GO TO 20
         N = NINT(NN+0.00001D0)
         DO 10 I=1, N-1
            GM = GM*(NN-I)
      10 CONTINUE
         ABSN = DABS(N-NN)
         IF (ABSN.GT.0.1D0) GO TO 20
         GO TO 30
      20 GM = GM*DSQRT(PI)
      30 RETURN
         END
C
         SUBROUTINE RGAMMA (MM,NN,RGM)
C
C        ALGORITHM AS297.10 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Calculate the ratio of two gamma functions
C
```

```
      DOUBLE PRECISION MM, NN, RGM
C
      RGM = 1.0D0
      IF (MM.EQ.NN) GO TO 40
      IF (MM.LT.NN) GO TO 20
      DO 10 I=1, NINT(MM-NN)
         RGM = RGM*(MM-I)
   10 CONTINUE
      GO TO 40
   20 DO 30 I=1, NINT(NN-MM)
         RGM = RGM*(NN-I)
   30 CONTINUE
      RGM = 1.0D0/RGM
   40 RETURN
      END
C
      SUBROUTINE BINOM (P,WKM1,BIN)
C
C        ALGORITHM AS297.11 APPL. STATIST. (1995) VOL.44, NO.2
C
C        Binomial coefficients for power P using the method of
C        Pascal triangle
C
      INTEGER P, P1
      DOUBLE PRECISION BIN(P+1), WKM1(P+1,P+1)
C
      P1 = P+1
      DO 20 I=1, P1
         DO 10 J=1, P1
            WKM1(I,J) = 1.0D0
   10    CONTINUE
   20 CONTINUE
      IF (P.LT.2) GO TO 50
      DO 40 I=3, P1
         DO 30 J=2, I-1
            WKM1(I,J) = WKM1(I-1,J-1)+WKM1(I-1,J)
   30    CONTINUE
   40 CONTINUE
   50 DO 60 J=1, P1
         BIN(J) = WKM1(P1,J)
   60 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE HYBRID(NPARAM,PARAM,N,T0,RHO,NT,BL,BU,NP,POINTS,LOW,
     +                  IFAULT)
C     .. Scalar Arguments ..
      DOUBLE PRECISION RHO,T0
      INTEGER IFAULT,N,NP,NPARAM,NT
C     ..
C     .. Array Arguments ..
      DOUBLE PRECISION BL(NPARAM),BU(NPARAM),LOW(NPARAM+1),
     +                 PARAM(NPARAM),POINTS(NPARAM,N)
C     ..
C     .. Local Scalars ..
      DOUBLE PRECISION BST,DIFF,NEWOBJ,OBJF,ONE,P,T,U,ZERO
      INTEGER I,I1,I2
C     ..
C     .. External Functions ..
      DOUBLE PRECISION RANDOM
      EXTERNAL RANDOM
C     ..
C     .. External Subroutines ..
      EXTERNAL FUNCT1,TRADNL
C     ..
C     .. Intrinsic Functions ..
      INTRINSIC DEXP
C
C      Define the constants ZERO and ONE and set
C      initial values for BST and IFAULT
C
C     .. Data statements ..
      DATA ZERO,ONE,BST/0.0D0,1.0D0,1.0D+10/
C     ..
      IFAULT = 0
C
C      ****************************************
C      *** Start of Annealing component to    ***
C      *** generate a set of starting points. ***
C      ****************************************
C
C      Initialise the temperature
C
      T = T0
C
C      Choose an initial point
C
      DO 10 I = 1,NPARAM
          PARAM(I) = RANDOM(BL(I),BU(I))
   10 CONTINUE
C
C      Calculate the function value for the initial point
C
      CALL FUNCT1(NPARAM,PARAM,OBJF)
C
C      Begin the loop over I1 until Nt Temp. reductions.
C
      DO 90 I1 = 1,NT
C
C      Set the counter for the No of points accepted to one
C
         NP = 1
C
C      Begin the loop over I2 until N points have been considered.
C
```

```
            DO 80 I2 = 1,N
C
C       Record the present point
C
                DO 20 I = 1,NPARAM
                    POINTS(I,NP+1) = PARAM(I)
   20           CONTINUE
C
C       Select a new point
C
                DO 30 I = 1,NPARAM
                    PARAM(I) = RANDOM(BL(I),BU(I))
   30           CONTINUE
C
C       Calculate the function value of this new point
C
                CALL FUNCT1(NPARAM,PARAM,NEWOBJ)
C
C       Calculate the difference
C
                DIFF = (NEWOBJ-OBJF)
C
C       If the new point is better than the old one then increase the
C       Np counter, and record it as a possible starting point. Move
C       to this new point.
C
                IF (DIFF.LT.ZERO) THEN
                    NP = NP + 1
                    DO 40 I = 1,NPARAM
                        POINTS(I,NP) = PARAM(I)
   40               CONTINUE
                    OBJF = NEWOBJ
                    IF (NEWOBJ.LT.BST) THEN
                        BST = NEWOBJ
                        DO 50 I = 1,NPARAM
                            POINTS(I,1) = PARAM(I)
   50                   CONTINUE
                    END IF
C
C       Otherwise use the Metropolis Criterion
C
                ELSE
C
C       P is the comparison statistic. U is a standard Uniform variate.
C
                    P = DEXP(-DIFF/T)
                    U = RANDOM(ZERO,ONE)
C
C       If U < P then accept this point. Increase the Np counter,
C       Record this point as a possible starting point, and move
C       to this new point.
C
                    IF (U.LT.P) THEN
                        NP = NP + 1
                        OBJF = NEWOBJ
                        DO 60 I = 1,NPARAM
                            POINTS(I,NP) = PARAM(I)
   60                   CONTINUE
C
C       If U >= P then reject this point and return to the old point.
C
```

```
                  ELSE
                      DO 70 I = 1,NPARAM
                          PARAM(I) = POINTS(I,NP+1)
   70                 CONTINUE
                  END IF
C
C      Both alternatives for the comparison of the new and old point
C      Have been considered.
C
              END IF
C
C      Continue with the loop until N new points have been considered
C
   80     CONTINUE
C
C      Set the error indicator to 1 if the algorithm appears to have
C      converged.
C
           IF (NP.EQ.1) THEN
               IFAULT = I1
           END IF
C
C      Once N points have been considered, lower the Temperature.
C
           T = T*RHO
C
C      Continue with the loop until the Temp. has been reduced Nt times
C
   90 CONTINUE
C
C       ****************************************
C       *** End of Annealing component, NP    ***
C       *** starting points are now stored in ***
C       *** the array POINTS.                 ***
C       ****************************************
C
C       Set an initially high value of LOW(NPARAM+1)
C
       LOW(NPARAM+1) = 1.0D+10
C
C       ****************************************
C       *** Start of the traditional component, ***
C       *** which loops over the NP starting    ***
C       *** points calling the traditional      ***
C       *** minimisation routine for each.      ***
C       ****************************************
C
C       Begin the loop over the Np starting points.
C
       DO 130 I1 = 1,NP
C
C       Set the PARAM array to be the values of the I1st start point
C
           DO 110 I2 = 1,NPARAM
               PARAM(I2) = POINTS(I2,I1)
  110      CONTINUE
C
C       Call traditional minimisation routine
C
           CALL TRADNL(NPARAM,PARAM,BL,BU,OBJF)
C
```

```
C       Check to see if this starting point gives the
C       lowest function value to date
C
          IF (OBJF.LT.LOW(NPARAM+1)) THEN
              DO 120 I = 1,NPARAM
                  LOW(I) = PARAM(I)
  120         CONTINUE
              LOW(NPARAM+1) = OBJF
          END IF
C
C       End the Np loop
C
  130 CONTINUE
C
C       ************************************
C       *** End of the second, traditional ***
C       *** component.                     ***
C       ************************************
C
C       Return to calling routine
C
      RETURN
C
C       End of subroutine HYBRID
C
      END
```

```
      SUBROUTINE SPP (DI, NIN, SPEC, LARGE, NOUT, XSTORE, XOUT, ISEED,
     *                RANNUM, IFAULT)
C
C        ALGORITHM AS 300.1 APPL. STATIST. (1996), VOL.45, NO.1
C
C        Efficient algorithm for determining a simulated percentile
C        point with a fixed degree of accuracy
C
      INTEGER DI, NIN(4), NOUT(3), ISEED, IFAULT
      REAL SPEC(4), XSTORE(DI), XOUT(3), RANNUM,
      LOGICAL LARGE
C
      INTEGER HIGH, LOW, N, N1, N2, NBND, NCHK, NDEX1, NDEX2, NSTORE
      REAL AN, ANP, C1, C2, D, ERR, HALF, MAXERR, ONE, P, PQ, TWO,
     *     THREE, XNEW, ZERO
C
      PARAMETER (ONE = 1.0E0, TWO = 2.0E0, THREE = 3.0E0, HALF = 0.5,
     *           ZERO = 0.0E0)
C
      EXTERNAL INSERT, RANNUM
C
C        Initialise local variables and check for input failures
C
      IFAULT = 0
      N1 = NIN(1)
      IF (N1 .LE. 2) IFAULT = IFAULT + 1
      N2 = NIN(2)
      IF (N2 .LE. N1) IFAULT = IFAULT + 2
      NCHK = NIN(3)
      IF (NCHK .LE. N2) IFAULT = IFAULT + 4
      NBND = NIN(4)
      IF (NBND .LE. NCHK) IFAULT = IFAULT + 8
      C1 = SPEC(1)
      IF (C1 .LE. ZERO) IFAULT = IFAULT + 16
      C2 = SPEC(2)
      IF (C2 .LE. C1) IFAULT = IFAULT + 32
      MAXERR = SPEC(3)
      IF (MAXERR .LE. ZERO) IFAULT = IFAULT + 64
      P = SPEC(4)
      IF (P .LE. ZERO .OR. P .GT. HALF) IFAULT = IFAULT + 128
      IF (DI .LT. TWO * C2 * SQRT(NBND * P * (ONE - P)) + THREE)
     *   IFAULT = IFAULT + 256
      IF (IFAULT .NE. 0) RETURN
      PQ = P * (ONE - P)
      NDEX1= 1
      NDEX2 = MAX0(2, N1)
C
C        Start iterating simulated values
C
      DO 50 N = 1, NBND
         XNEW = RANNUM(ISEED)
         IF (N .EQ. 1) THEN
            XSTORE(1) = XNEW
         ELSE IF (N .EQ. 2) THEN
            XSTORE(2) = XNEW
            NSTORE = 2
         IF (XNEW .GT. XSTORE(1) .EQV. LARGE) THEN
            XSTORE(2) = XSTORE(1)
            XSTORE(1) = XNEW
         ENDIF
C
```

```
C        Stage 1: sort and store all of the first N1 simulated values
C
         ELSE IF (N .LE. N1) THEN
            CALL INSERT (NSTORE, DI, XNEW, XSTORE, 0, LARGE)
C
C        Stage 2: sort and store only N1 of the first N2 simulated
C                 values
C
         ELSE IF (N .LE. N2) THEN
            IF (XNEW .GT. XSTORE(NDEX2) .EQV. LARGE) THEN
            CALL INSERT (NSTORE, DI, XNEW, XSTORE, 2, LARGE)
         ENDIF
C
C        Stage 3: sort and store only a probabilistically determined
C                 number of simulated values
C             3.1: do not store simulated value but change index
C                  numbers
C
         ELSE IF (XNEW .GT. XSTORE(1) .EQV. LARGE) THEN
            NDEX1 = NDEX1 + 1
            NDEX2 = NDEX2 + 1
C
C             3.2: sort and store the simulated value
C
         ELSE IF (XNEW .GT. XSTORE(NSTORE) .EQV. LARGE) THEN
            AN = N
            ANP = AN * P
            D = C2 * SQRT(AN * PQ)
            LOW = ANP - D
C
C           3.2.1: delete the simulated value that had been in the first
C                  storage location
C
            IF (NDEX1 .LT. LOW) THEN
               CALL INSERT (NSTORE, DI, XNEW, XSTORE, 1, LARGE)
               NDEX1 = NDEX1 + 1
               NDEX2 = NDEX2 + 1
C
C           3.2.2: no previously stored simulated value is to be
C                  deleted
C
            ELSE
               HIGH = ANP + D + 1
               IF (NDEX2 .LT. HIGH) THEN
                  NDEX2 = NDEX2 + 1
                  CALL INSERT (NSTORE, DI, XNEW, XSTORE, 0, LARGE)
C
C           3.2.3: delete the simulated value that had been in the last
C                  storage location
C
            ELSE
               CALL INSERT (NSTORE, DI, XNEW, XSTORE, 2, LARGE)
            ENDIF
         ENDIF
C
C             3.3: do not store simulated value and do not change index
C                  numbers
C
      ENDIF
C
C        Stage 4: check to see if the desired precision has been
```

```
C                achieved at the end of each NCHK iterations
C
      IF (N .EQ. (N / NCHK) * NCHK .OR. N .EQ. NBND) THEN
         AN = N
         ANP = AN * P
         D = C1* SQRT(AN * PQ)
         LOW = ANP - D + HALF
         HIGH = ANP + D + ONE + HALF
C
C            4.1: the desired order statistics are not in storage
C                 (abnormal termination)
C
         IF (NDEX1 .GT. LOW .OR. NDEX2 .LT. HIGH) THEN
            IFAULT = 999
            GOTO 100
         ENDIF
C
C            4.2: the desired order statistics are in storage
C
         LOW = LOW - NDEX1 + 1
         HIGH = HIGH - NDEX1 + 1
         IF (LARGE) THEN
            XOUT(3) = XSTORE(LOW)
            XOUT(2) = XSTORE(HIGH)
         ELSE
            XOUT(3) = XSTORE(HIGH)
            XOUT(2) = XSTORE(LOW)
         ENDIF
            ERR = (XOUT(3) - XOUT(2)) / TWO
            LOW = ANP - NDEX1 + 1
            XOUT(1) = XSTORE(LOW)
C
C            4.3: the desired precision has been achieved when ERR is
C                 less than MAXERR
C
            IF (ERR .LT. MAXERR) GOTO 100
         ENDIF
   50 CONTINUE
C
C            4.4: the desired precision is not achieved within the
C                 fixed number of iterations
C
      N = N - 1
  100 NOUT(1) = N
      NOUT(2) = NSTORE
      NOUT(3) = NDEX1
C
      RETURN
      END
C
      SUBROUTINE INSERT (NSTORE, DI, XNEW, XSTORE, CONTRL, LARGE)
C
C        ALGORITHM AS 300.2 APPL. STATIST. (1996), VOL.45, NO.1
C
C        Divide and conquer algorithm to sort and store a new value
C        (XNEW) among a set of previously sorted and stored values
C
      INTEGER NSTORE, DI, CONTRL
      REAL XNEW, XSTORE(DI)
      LOGICAL LARGE
C
```

```
      INTEGER I, IBND1, IBND2, ITRY
      REAL XHOLD
C
C        Initialise local variables
C
      IBND1 = 0
      IBND2 = NSTORE + 1
C
C        Find the appropriate position to insert XNEW into the
C        sorted set
C
   50 ITRY = (IBND1 + IBND2) / 2
      IF (ITRY.LT.0 .OR. ITRY.GT.NSTORE) RETURN
      IF (ITRY.EQ.IBND1) GOTO 100
      IF (XSTORE(ITRY) .GT. XNEW .EQV. LARGE) THEN
         IBND1 = ITRY
      ELSE
         IBND2 = ITRY
      ENDIF
      GOTO 50
  100 IF (CONTRL .NE. 1) THEN
C
C        If CONTRL equals 0, add XNEW to the list
C
         IF (CONTRL .EQ. 0) NSTORE = NSTORE + 1
C
C        If CONTRL equals 2, add XNEW to the list and delete the value
C        in the last storage location
C
         DO 150 I = IBND2, NSTORE
            XHOLD = XSTORE(I)
            XSTORE(I) = XNEW
            XNEW = XHOLD
  150    CONTINUE
C
C        If CONTRL equals 1, add XNEW to the list and delete the
C        first storage location
C
      ELSE
         DO 200 I = IBND1, 1, -1
            XHOLD = XSTORE(I)
            XSTORE(I) = XNEW
            XNEW = XHOLD
  200    CONTINUE
      ENDIF
C
      RETURN
      END
```

```
      SUBROUTINE LOGF1(CELL, DELTA, EPS, INQCOL, INQROW, LQ, R, RS,
     *     S, UQ, COL, IFAULT, LFCELL, LFCOL, LFROW, LOGEF1, LOGPH,
     *     LOGPHP, MU1, N, QCELL, QCOL, QROW, ROW)
C
C        ALGORITHM AS 301.1 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes the logarithms of F1, P(H) and P(H')
C
      INTEGER R, RS, S, CELL(RS), COL(S), IFAULT, N, ROW(R)
      REAL DELTA, EPS, INQCOL(S), INQROW(R), LQ, UQ, LOGEF1, LOGPH,
     *     LOGPHP, MU1, QCELL(RS), QCOL(S), QROW(R)
      DOUBLE PRECISION LFCELL(RS), LFCOL(S), LFROW(R)
C
      INTEGER I, J, K
      DOUBLE PRECISION LFM
      REAL C1, C2, INTOFG, LGPHI1, LGPHI2, LGPHI3, LOGFY,
     *     MLOGG, RPI, XMAX
C
      EXTERNAL MRGINS, KAPPA0, PARRHO, LIMITS, MXLOGG
C
      IFAULT = 0
      CALL MRGINS(CELL, INQCOL, INQROW, R, RS, S, COL, IFAULT,
     *            LFCELL, LFCOL, LFM, LFROW, N, ROW)
      IF (IFAULT .EQ. 0) THEN
         CALL KAPPA0(N, DELTA, EPS, LFM, LFROW,
     *               ROW, INQROW, R, IFAULT, QROW)
         CALL KAPPA0(N, DELTA, EPS, LFM, LFCOL,
     *               COL, INQCOL, S, IFAULT, QCOL)
         CALL PARRHO(LQ, UQ, IFAULT, MU1, RPI)
         IF (IFAULT .NE. 1) THEN
C
C        Compute log(Fisher-Yates probability) and the cell
C        probabilities
C
            LOGFY = -SNGL(LFM)
            K = 0
            DO 10 I = 1, R
               LOGFY = LOGFY + SNGL(LFROW(I))
               DO 5 J = 1, S
                  K = K + 1
                  QCELL(K) = QROW(I)*QCOL(J)
                  LOGFY = LOGFY - SNGL(LFCELL(K))
    5          CONTINUE
   10       CONTINUE
            DO 15 J = 1, S
   15       LOGFY = LOGFY + SNGL(LFCOL(J))
C
C        Compute log PHI1 = log INTOFG((N(I, J)), EXP(X)(Q(I.)Q(.J)))
C
            CALL LIMITS(N, LFM, LFCELL, CELL, QCELL, RS,    C1, C2)
            CALL MXLOGG(N, C1, C2, LFM, LFCELL, CELL,
     *                  QCELL, RS, 1, 1, MLOGG, XMAX)
            LGPHI1 = ALOG(RPI*INTOFG(N, C1, C2, DELTA, EPS, LFM, LFCELL,
     *              CELL, MLOGG, QCELL, RS, 1, XMAX, IFAULT)) + MLOGG
C
C        Compute log PHI2 = log INTOFG((N(I.)), EXP(X)(Q(I.)))
C
            CALL LIMITS(N, LFM, LFROW, ROW, QROW, R,    C1, C2)
            CALL MXLOGG(N, C1, C2, LFM, LFROW, ROW,
     *                  QROW, R, 1, 1, MLOGG, XMAX)
            LGPHI2 = ALOG(RPI*INTOFG(N, C1, C2, DELTA, EPS, LFM,
```

```
     *          LFROW, ROW, MLOGG, QROW, R, 1, XMAX, IFAULT)) + MLOGG
C
C        Compute log PHI3 = log INTOFG((N(.J)), EXP(X)(Q(.J)))
C
            CALL LIMITS(N, LFM, LFCOL, COL, QCOL, S,    C1, C2)
            CALL MXLOGG(N, C1, C2, LFM, LFCOL, COL,
     *                  QCOL, S, 1, 1, MLOGG, XMAX)
            LGPHI3 = ALOG(RPI*INTOFG(N, C1, C2, DELTA, EPS, LFM,
     *          LFCOL, COL, MLOGG, QCOL, S, 1, XMAX, IFAULT)) + MLOGG
C
C        Compute the logarithms of F1, P(H) and P(H')
C
            LOGEF1 = LGPHI1 - (LOGFY + LGPHI2 + LGPHI3)
            LOGPH = LOGFY + LGPHI3
            LOGPHP = LGPHI1 - LGPHI2
         ENDIF
      ENDIF
C
      RETURN
      END
C
      SUBROUTINE MRGINS(CELL, INQCOL, INQROW, R, RS, S, COL, IFAULT,
     *                  LFCELL, LFCOL, LFM, LFROW, N, ROW)
C
C        ALGORITHM AS 301.2 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes the table total N, the row and column totals, N(I.)
C        and N(.J), log N(I.)!, log N(.J)!, and checks whether (Q(I.))
C        and (Q(.J)) sum to 1, CELL(I) < 0 and N < 1 or N > 10E+09
C
      INTEGER R, RS, S, CELL(RS), COL(S), IFAULT, N, ROW(R)
      REAL INQCOL(S), INQROW(R)
      DOUBLE PRECISION LFCELL(RS), LFCOL(S), LFM, LFROW(R)
C
      DOUBLE PRECISION LFACT
      REAL APPONE, NINES, SUMQ, ZERO
      INTEGER I, J, K, L
C
      DATA APPONE, NINES, ZERO /1.000001, 0.999999, 0.0/
C
      DO 5 I = 1, RS
         IF (CELL(I) .LT. 0) IFAULT = 2
    5 CONTINUE
      IF (IFAULT .NE. 2) THEN
         SUMQ = ZERO
         N = 0
         DO 15 I = 1, R
            SUMQ = SUMQ + INQROW(I)
            ROW(I) = 0
            L = (I - 1)*S
            DO 10 J = 1, S
               K = L + J
               ROW(I) = ROW(I) + CELL(K)
               LFCELL(K) = LFACT(CELL(K))
   10       CONTINUE
            LFROW(I) = LFACT(ROW(I))
            N = N + ROW(I)
   15    CONTINUE
         IF (N .GE. 1 .AND. N .LE. 1000000000) THEN
            IF (SUMQ .LE. NINES .OR. SUMQ .GE. APPONE) THEN
               IFAULT = 3
```

```
               ELSE
                  SUMQ = ZERO
                  LFM = LFACT(N)
                  DO 25 J = 1, S
                     SUMQ = SUMQ + INQCOL(J)
                     COL(J) = 0
                     K = J
                     DO 20 I = 1, R
                        COL(J) = COL(J) + CELL(K)
                        K = K + S
   20                CONTINUE
                     LFCOL(J) = LFACT(COL(J))
   25             CONTINUE
                  IF (SUMQ .LE. NINES .OR. SUMQ .GE. APPONE) IFAULT = 3
               ENDIF
            ELSE
               IFAULT = 4
            ENDIF
         ENDIF
C
         RETURN
         END
C
         DOUBLE PRECISION FUNCTION LFACT(N)
C
C        ALGORITHM AS 301.3 APPL. STATIST. (1996), VOL.45, NO.2
C
C       Computes log Gamma(N + 1) = log N!
C
         INTEGER N
C
         DOUBLE PRECISION C1, C2, LF(0:4), L2PID2, ONE, ONED2, U
C
         DATA C1, C2 /0.08333333333333333D0, 0.002777777777777778D0/,
        *     L2PID2, LF(0), LF(1) /0.9189385332046728D0, 0.0D0, 0.0D0/,
        *     LF(2), LF(3) /0.6931471805599454D0, 1.791759469228055D0/,
        *     LF(4), ONE, ONED2 /3.178053830347946D0, 1.0D0, 0.5D0/
C
         IF (N .LE. 4) THEN
            LFACT = LF(N)
         ELSE
            U = DBLE(N + 1)
         LFACT = (U - ONED2)*DLOG(U) - U + L2PID2 + C1/U - C2*(ONE/U)**3
         ENDIF
C
         RETURN
         END
C
         SUBROUTINE KAPPA0(BIGM, DELTA, EPS, LFM, LOGFAC,
        *                  M, PRQ, T, IFAULT, Q)
C
C        ALGORITHM AS 301.4 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes K0 and then updates (Q(I.)) and (Q(.J))
C
         INTEGER BIGM, T, M(T), IFAULT
         REAL DELTA, EPS, PRQ(T), Q(T)
         DOUBLE PRECISION LFM, LOGFAC(T)
C
         REAL C1, C2, DUMMY1, DUMMY2, FIFTY, INTOFG, K0,
        *     LGNUM, LGDNOM, MLOGG, TEN, XMAX, ZERO
```

```
      INTEGER I
C
      DATA FIFTY, TEN, ZERO /50.0, 10.0, 0.0/
C
      EXTERNAL PARRHO, LIMITS, MXLOGG
C
      CALL PARRHO(TEN, FIFTY, IFAULT, DUMMY1, DUMMY2)
      CALL LIMITS(BIGM, LFM, LOGFAC, M, PRQ, T, C1, C2)
      CALL MXLOGG(BIGM, ZERO, C2, LFM, LOGFAC, M,
     *            PRQ, T, 2, 2, MLOGG, XMAX)
      LGNUM = ALOG(INTOFG(BIGM, ZERO, C2, DELTA, EPS, LFM,
     *            LOGFAC, M, MLOGG, PRQ, T, 2, XMAX, IFAULT)) + MLOGG
      CALL MXLOGG(BIGM, C1/BIGM, ZERO, LFM, LOGFAC, M,
     *            PRQ, T, 3, 3, MLOGG, XMAX)
      LGDNOM = ALOG(INTOFG(BIGM, C1/BIGM, ZERO, DELTA, EPS, LFM,
     *            LOGFAC, M, MLOGG, PRQ, T, 3, XMAX, IFAULT)) + MLOGG
      K0 = EXP(LGNUM - LGDNOM)
      DO 5 I = 1, T
         Q(I) = (FLOAT(M(I)) + PRQ(I)*K0)/(BIGM + K0)
    5 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE LIMITS(BIGM, LFM, LOGFAC, M, Q, T, C1, C2)
C
C        ALGORITHM AS 301.5 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes the logarithms of the limits C1 and C2
C
      INTEGER BIGM, T, M(T)
      DOUBLE PRECISION LFM, LOGFAC(T)
      REAL Q(T), C1, C2
C
      DOUBLE PRECISION DPC2
      REAL ZERO
      INTEGER I
C
      DATA ZERO /0.0/
C
      C1 = ZERO
      DPC2 = LFM
      DO 5 I = 1, T
         DPC2 = DPC2 + DBLE(M(I))*DLOG(DBLE(Q(I))) - LOGFAC(I)
         IF (M(I) .EQ. BIGM) C1 = ALOG(Q(I))
    5 CONTINUE
      C2 = SNGL(DPC2)
C
      RETURN
      END
C
      SUBROUTINE MXLOGG(BIGM, C1, C2, LFM, LOGFAC, M,
     *                  Q, T, TAU, ZETA, MLOGG, XMAX)
C
C        ALGORITHM AS 301.6 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Finds the approximate maximum of log G(X),
C        log G(X)/(1 + M/EXP(X)) or log G(X)/(M + EXP(X)) where
C        M denotes BIGM.
C
      INTEGER BIGM, T, M(T), TAU, ZETA
```

```
      REAL C1, C2, Q(T), MLOGG, XMAX
      DOUBLE PRECISION LFM, LOGFAC(T)
C
      DOUBLE PRECISION D1, D2, DPZERO, EPS, LOGDM
      REAL CONDTN, KMAX, LMB, LMB2, LMU1, LXLMU1, ONE,
     *     ONED2, RHO, RLMB, SAVEK, THRTY2, U, Y, ZERO
      INTEGER I
C
      COMMON /RHOPAR/ LMB, LMB2, LMU1, RLMB
C
      DATA DPZERO, ONE, ONED2, EPS, THRTY2, ZERO
     *     /0.0D0, 1.0, 0.5, 0.001D0, 32.0, 0.0/
C
      RHO(U) = LMB/(LMB2 + (U - LMU1)**2)
      LXLMU1 = SNGL(LOGDM(BIGM, LFM, LOGFAC, M, Q, T, TAU, LMU1)) +
     *         ALOG(RHO(LMU1))
      IF (C1 .NE. ZERO) THEN
         IF (C1 + ALOG(RHO(-THRTY2)) .LT. LXLMU1) THEN
            XMAX = LMU1
            MLOGG = LXLMU1
         ELSE
            XMAX = -THRTY2
            MLOGG = C1 + ALOG(RHO(-THRTY2))
         ENDIF
      ELSE
         CONDTN = ONE
         DO 5 I = 1, T
            Y = Q(I)*BIGM
            CONDTN = CONDTN + ((FLOAT(M(I)) - Y)**2 - FLOAT(M(I)))/Y
    5    CONTINUE
         IF (CONDTN .LE. ZERO .AND. C2 .NE. ZERO) THEN
            IF (C2 + ALOG(RHO(THRTY2)) .LT. LXLMU1) THEN
               XMAX = LMU1
               MLOGG = LXLMU1
            ELSE
               XMAX = THRTY2
               MLOGG = C2 + ALOG(RHO(THRTY2))
            ENDIF
         ELSE
C
C           Find the zero of d(log F(X))/dx,
C           d(log F(X)/(1 + M/EXP(X)))/dx or d(log F(X)/(M + EXP(X)))/dx
C           by Newton-Raphson's method where M denotes BIGM.
C
            KMAX = ONE
            SAVEK = KMAX
            CALL DERIV(BIGM, M, Q, T, KMAX, ZETA,    D1, D2)
   10       IF (DABS(D1) .LT. EPS .OR. ABS(ALOG(KMAX)) .GT .THRTY2)
     *          GO TO 15
               IF (D2 .LT. DPZERO) THEN
                  KMAX = KMAX - D1/D2
                  IF (KMAX .LT. ZERO) THEN
                     SAVEK = ONED2*SAVEK
                     KMAX = SAVEK
                  ENDIF
               ELSE
                  SAVEK = ONED2*SAVEK
                  KMAX = SAVEK
               ENDIF
               CALL DERIV(BIGM, M, Q, T, KMAX, ZETA,    D1, D2)
               GO TO 10
```

```
   15       XMAX = ALOG(KMAX)
            IF (XMAX .GT. THRTY2) XMAX = THRTY2
            MLOGG = SNGL(LOGDM(BIGM, LFM, LOGFAC, M, Q, T, TAU, XMAX))
     *              + ALOG(RHO(XMAX))
            IF (MLOGG .LT. LXLMU1) THEN
               XMAX = LMU1
               MLOGG = LXLMU1
            ENDIF
            IF (TAU .LT. 3 .AND. MLOGG .LT. C2 + ALOG(RHO(THRTY2))) THEN
               XMAX = THRTY2
               MLOGG = C2 + ALOG(RHO(THRTY2))
            ENDIF
         ENDIF
      ENDIF
C
      RETURN
      END
C
      SUBROUTINE DERIV(BIGM, M, Q, T, K, ZETA, D1, D2)
C
C        ALGORITHM AS 301.7 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes the derivatives of d(log F(X))/dx,
C        d(log F(X)/(1 + M/EXP(X)))/dx or d(log F(X)/(M + EXP(X)))/dx
C        where M denotes BIGM.
C
      INTEGER BIGM, T, M(T), ZETA
      REAL Q(T), K
      DOUBLE PRECISION D1, D2
C
      INTEGER ALPHA, I
      DOUBLE PRECISION C1, C2, ONE, ONED2, PSIAPP, PSIPAP,
     *                 TEN, TWO, U, Z, ZERO
C
      DATA C1, C2 /0.08333333333333333D0, 0.1666666666666667D0/,
     * ONE, ONED2, TEN, TWO, ZERO /1.0D0, 0.5D0, 10.0D0, 2.0D0, 0.0D0/
C
      PSIAPP(U) = DLOG(U) - ONED2/U - C1*(ONE/U)**2
      PSIPAP(U) = ONE/U + ONED2*(ONE/U)**2 + C2*(ONE/U)**3
      D1 = ZERO
      D2 = ZERO
      DO 15 ALPHA = 1, T
         IF (M(ALPHA) .GT. 0) THEN
            Z = DBLE(Q(ALPHA)*K)
            I = 0
    5       IF (I + Z .GT. TEN .OR. I .EQ. M(ALPHA)) GO TO 10
               D1 = D1 + Q(ALPHA)/(I + Z)
               D2 = D2 - (Q(ALPHA)/(I + Z))**2
               I = I + 1
               GO TO 5
   10       IF (I .LT. M(ALPHA)) THEN
               D1 = D1 + Q(ALPHA)*(PSIAPP(M(ALPHA) + Z) - PSIAPP(I + Z))
               D2 = D2 + Q(ALPHA)**2*(PSIPAP(M(ALPHA) + Z) -
     *              PSIPAP(I + Z))
            ENDIF
         ENDIF
   15 CONTINUE
      I = 0
   20 IF (DBLE(I + K) .GT. TEN .OR. I .EQ. BIGM) GO TO 25
         D1 = D1 - ONE/(I + K)
         D2 = D2 + (ONE/(I + K))**2
```

```
            I = I + 1
            GO TO 20
         25 IF (I .LT. BIGM) THEN
            D1 = D1 - PSIAPP(DBLE(BIGM + K)) + PSIAPP(DBLE(I + K))
            D2 = D2 - PSIPAP(DBLE(BIGM + K)) + PSIPAP(DBLE(I + K))
           ENDIF
           IF (ZETA .EQ. 2) THEN
            D1 = D1 + ONE/(K*(ONE + K/BIGM))
            D2 = D2 - (ONE + TWO*K/BIGM)*(ONE/(K*(ONE + K/BIGM)))**2
           ELSE IF (ZETA .EQ. 3) THEN
            D1 = D1 - ONE/(BIGM + K)
            D2 = D2 + (ONE/(BIGM + K))**2
           ENDIF
      C
           RETURN
           END
      C
           REAL FUNCTION INTOFG(BIGM, C1, C2, DELTA, EPS, LFM, LGF,
          *                     M, MLOGG, Q, T, TAU, XMAX, IFAULT)
      C
      C       ALGORITHM AS 301.8 APPL. STATIST. (1996), VOL.45, NO.2
      C
      C       Computes the integral of G(X), G(X)/(1 + M/EXP(X)) or
      C       G(X)/(M + EXP(X)) where M denotes BIGM.
      C
           INTEGER BIGM, T, M(T), TAU, IFAULT
           REAL C1, C2, DELTA, EPS, MLOGG, Q(T), XMAX
           DOUBLE PRECISION LFM, LGF(T)
      C
           REAL C, CONST, ONE,
          *     QG8, THRTY2, XLIMIT, YL, YU, ZERO
           INTEGER CVERG, I
      C
           DATA CONST, ONE, THRTY2, ZERO /-23.0, 1.0, 32.0, 0.0/
      C
           EXTERNAL CHKCVG
      C
           INTOFG = ZERO
      C
      C       Integrate to the left from XMAX and then to the right
      C       from XMAX
      C
           XLIMIT = -THRTY2
           C = C1
           YL = -(ONE - ONE/DELTA)
           YU = ZERO
           DO 15 I = 1, 2
            CVERG = 0
            IF (XMAX .EQ. XLIMIT) CALL CHKCVG(BIGM, C, MLOGG + CONST,
           *     DELTA, EPS, LFM, LGF, M, MLOGG, Q, I, T, TAU, XMAX, CVERG,
           *     IFAULT, INTOFG, ZERO, ZERO)
          5    IF (CVERG .EQ. 1) GO TO 10
               INTOFG = INTOFG + QG8(BIGM, LFM, LGF, M, MLOGG, Q, T,
           *                         TAU, XMAX + YL, XMAX + YU)
               IF (YL .LT. ZERO) THEN
               CALL CHKCVG(BIGM, C, MLOGG + CONST, DELTA, EPS, LFM, LGF,
           *      M, MLOGG, Q, I, T, TAU, XMAX, CVERG, IFAULT, INTOFG,
           *      YU, YL)
               ELSE
               CALL CHKCVG(BIGM, C, MLOGG + CONST, DELTA, EPS, LFM, LGF,
           *      M, MLOGG, Q, I, T, TAU, XMAX, CVERG, IFAULT, INTOFG,
```

```
     *         YL, YU)
               ENDIF
               GO TO 5
   10    XLIMIT = THRTY2
         C = C2
         YL = ZERO
         YU = ONE - ONE/DELTA
   15 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE CHKCVG(BIGM, C, CRITER, DELTA, EPS, LFM, LOGFAC, M,
     *   MLOGG, Q, SIDE, T, TAU, XMAX, CVERG, IFAULT, INTOFG, YL, YU)
C
C        ALGORITHM AS 301.9 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Checks for the convergence to the limit C and updates the
C        end-points of the integration intervals
C
      INTEGER BIGM, T, M(T), SIDE, TAU, CVERG, IFAULT
      REAL C, CRITER, DELTA, EPS, MLOGG, Q(T), XMAX, INTOFG, YL, YU
      DOUBLE PRECISION LFM, LOGFAC(T)
C
      DOUBLE PRECISION LOGDM
      REAL LDIRMT, LMB, LMB2,
     *     LMU1, LTERM, ONE, PID2, RHO, RLMB,
     *     THRTY2, U, V, ZERO
C
      COMMON /RHOPAR/ LMB, LMB2, LMU1, RLMB
C
      DATA ONE, PID2, THRTY2, ZERO /1.0, 1.5707963, 32.0, 0.0/
C
      RHO(U) = LMB/(LMB2 + (U - LMU1)**2)
      LDIRMT = SNGL(LOGDM(BIGM, LFM, LOGFAC, M, Q, T, TAU, XMAX + YU))
      IF (C .EQ. ZERO) THEN
         IF (LDIRMT + ALOG(RHO(XMAX + YU)) .LT. CRITER) CVERG = 1
      ELSE
         IF (ABS(LDIRMT - C) .LT. ALOG(ONE + EPS)) CVERG = 1
      ENDIF
      IF (CVERG .EQ. 0) THEN
         IF (ABS(XMAX + DELTA*YU) .GE. THRTY2) THEN
            CVERG = 1
            IF (TAU .EQ. 1) IFAULT = 5
         ELSE
            YL = YU
            YU = DELTA*YU
         ENDIF
      ENDIF
      IF (CVERG .EQ. 1) THEN
         IF (C .NE. ZERO) THEN
C
C           LTERM is assigned the value of the integral of DX
C
            V = ATAN((XMAX + YU - LMU1)*RLMB)
            LTERM = ZERO
            IF (SIDE .EQ. 1) THEN
               IF (V + PID2 .GT. ZERO) LTERM = V + PID2
            ELSE
               IF (PID2 - V .GT. ZERO) LTERM = PID2 - V
            ENDIF
```

```
            IF (LTERM .GT. ZERO) INTOFG = INTOFG +
     *                                 EXP(C + ALOG(LTERM) - MLOGG)
          ENDIF
        ENDIF
C
        RETURN
        END
C
        REAL FUNCTION QG8(BIGM, LFM, LOGFAC, M, MLOGG, Q, T, TAU, XL, XU)
C
C          ALGORITHM AS 301.10 APPL. STATIST. (1996), VOL.45, NO.2
C
C          Integrates from XL to XU by eight-point Gaussian quadrature
C
        INTEGER BIGM, T, M(T), TAU
        DOUBLE PRECISION LFM, LOGFAC(T)
        REAL MLOGG, Q(T), XL, XU
C
        DOUBLE PRECISION LOGDM
        REAL C(8), CONST, G, LMB, LMB2, LMU1, ONED2, RHO, RLMB, U, X,
     *       Y, Z, ZERO
        INTEGER I, J
C
        COMMON /RHOPAR/ LMB, LMB2, LMU1, RLMB
C
        DATA C /0.48014493, 0.050614268, 0.39833324, 0.11119052,
     *          0.26276620, 0.15685332, 0.091717321, 0.18134189/,
     *       CONST, ONED2, ZERO /-23.0, 0.5, 0.0/
C
        RHO(U) = LMB/(LMB2 + (U - LMU1)**2)
        Y = ONED2*(XL + XU)
        QG8 = ZERO
        DO 10 I = 2, 8, 2
           Z = C(I - 1)*(XU - XL)
           X = Y - Z
           DO 5 J = 1, 2
              G = SNGL(LOGDM(BIGM, LFM, LOGFAC, M, Q, T, TAU, X))
     *            + ALOG(RHO(X)) - MLOGG
              IF (G .GT. CONST) QG8 = QG8 + C(I)*EXP(G)
              X = Y + Z
    5      CONTINUE
   10   CONTINUE
        QG8 = (XU - XL)*QG8
C
        RETURN
        END
C
        DOUBLE PRECISION FUNCTION LOGDM(BIGM, LFM, LOGFAC, M, Q, T,
     *                                  TAU, X)
C
C          ALGORITHM AS 301.11 APPL. STATIST. (1996), VOL.45, NO.2
C
C          Computes log F(X), log F(X)/(1 + M/EXP(X)) or
C          log F(X)/(M + EXP(X)) where M denotes BIGM.
C
        INTEGER BIGM, T, M(T), TAU, X
        DOUBLE PRECISION LFM, LOGFAC(T)
        REAL Q(T)
C
        DOUBLE PRECISION C, K, LGAPP, L2PID2, ONE, ONED2, TEN,
     *                   U, Z
```

```
      INTEGER ALPHA, I
C
      DATA C /0.08333333333333333D0/, L2PID2, ONE, ONED2, TEN
     * /0.9189385332046728D0, 1.0D0, 0.5D0, 10.0D0/
C
      LGAPP(U) = (U - ONED2)*DLOG(U) - U + L2PID2 + C/U
      K = DEXP(DBLE(X))
      LOGDM = LFM
      DO 15 ALPHA = 1, T
         IF (M(ALPHA) .GT. 0) THEN
            LOGDM = LOGDM - LOGFAC(ALPHA)
            Z = Q(ALPHA)*K
            I = 0
    5       IF (I + Z .GT. TEN .OR. I .EQ. M(ALPHA)) GO TO 10
               LOGDM = LOGDM + DLOG(I + Z)
               I = I + 1
               GO TO 5
   10       IF (I .LT. M(ALPHA)) LOGDM = LOGDM +
     *                            LGAPP(M(ALPHA) + Z) - LGAPP(I + Z)
         ENDIF
   15 CONTINUE
      I = 0
   20 IF (I + K .GT. TEN .OR. I .EQ. BIGM) GO TO 25
         LOGDM = LOGDM - DLOG(I + K)
         I = I + 1
         GO TO 20
   25 IF (I .LT. BIGM) LOGDM = LOGDM - LGAPP(BIGM + K) + LGAPP(I + K)
      IF (TAU .EQ. 2) THEN
         LOGDM = LOGDM - DLOG(ONE + BIGM/K)
      ELSE IF (TAU .EQ. 3) THEN
         LOGDM = LOGDM - DLOG(BIGM + K)
      ENDIF
C
      RETURN
      END
C
      SUBROUTINE PARRHO(LQ, UQ, IFAULT, MU1, RPI)
C
C        ALGORITHM AS 301.12 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Computes the parameter values of the log-Cauchy density
C
      REAL LQ, UQ, MU1, RPI
      INTEGER IFAULT
C
      REAL LMB, LMB2, LMU1, ONE, PI, RLMB, TENE9, TWO, ZERO
C
      COMMON /RHOPAR/ LMB, LMB2, LMU1, RLMB
C
      DATA ONE, PI, TENE9, TWO, ZERO
     *     /1.0, 3.1415927, 1.0E+09, 2.0, 0.0/
C
      IF (LQ .LE. ZERO .OR. LQ .GE. UQ .OR. UQ .GE. TENE9) THEN
         IFAULT = 1
      ELSE
         MU1 = SQRT(LQ*UQ)
         LMU1 = ALOG(MU1)
         LMB = ALOG(UQ/LQ)/TWO
         LMB2 = LMB**2
         RLMB = ONE/LMB
         RPI = ONE/PI
```

```
      ENDIF
C
      RETURN
      END
```

```
      SUBROUTINE MISRE(K, IP, IPP, NH, INV, COV, X, EPS, ITMAX,
     *                 IB, COVINV, XA, Y, Z, AII, A, W, WK, IT, SITA,
     *                 STAT, IFAULT)
C
C        ALGORITHM AS 302.1 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Subroutine to compute a multiple isotonic regression
C        for umbrella ordering with known peak
C
      INTEGER I, IFAULT, II, II1, INC, IP, IP1, IPP, IT, ITMAX, J,
     *        J1, K, L, NULLTY
      REAL DET, EPS, RABS, RMAX, STAT
      REAL A(IP), AII(K), COV(IPP, K), COVINV(IPP, K), SITA(IP, K),
     *     W(IP), WK(K, 6), X(IP, K), XA(K), Y(K), Z(K)
      INTEGER IB(IP), NH(IP)
C
      EXTERNAL ISRE, SUBINV
C
      IFAULT = 0
      IF (IP .LE. 1) IFAULT = 1
      IF (IPP .NE. IP * (IP + 1) / 2) IFAULT = 6
      IF (IFAULT .NE. 0) RETURN
C
C        Find inversions of covariances
C
      IF (INV .EQ. 1)THEN
        DO 10 I = 1, IP
          IB(I) = I
   10   CONTINUE
        DO 20 I = 1, K
          CALL SUBINV(COV(1, I), IP, IB, IP, COVINV(1, I), W, NULLTY,
     *     IFAULT, IPP, DET)
          IF (IFAULT .NE. 0) RETURN
   20   CONTINUE
        INV = 0
      ENDIF
C
      IT = 1
      IP1 = IP - 1
      DO 40 I = 2, IP
        DO 30 J = 1, K
          SITA(I, J) = X(I, J)
   30   CONTINUE
   40 CONTINUE
C
C        Find multivariate isotonic regression iteratively
C
   50 RMAX = 0.0
      DO 120 I = 1, IP
        II = I * (1 + I) / 2
C
C        For the i-th univariate isotonic regression
C
        DO 100 L = 1, K
          AII(L) = COVINV(II, L)
C
C        Set A(1),..., A(IP - 1) to the elements of A21
C
          IF (I .GT. 1) THEN
            II1 = II - I
            DO 60 J = 1, I - 1
```

```
              II1 = II1 + 1
              A(J) =  COVINV(II1, L)
   60       CONTINUE
          ENDIF
          IF (I .LT. IP) THEN
            II1 = II + I
            INC = I + 1
            DO 70 J = I, IP1
              A(J) = COVINV(II1, L)
              II1 = II1 + INC
              INC = INC + 1
   70       CONTINUE
          ENDIF
C
C        Prepare the input data Z(1),...,Z(IP - 1) for
C        univariate isotonic regression
C
          J1 = 1
          DO 80 J = 1, IP
            IF (J .NE. I) THEN
              Y(J1) = X(J, L) - SITA(J, L)
              J1 = J1 + 1
            ENDIF
   80     CONTINUE
          Z(L) = X(I, L)
          DO 90 J = 1, IP1
            Z(L) = Z(L) + Y(J) * A(J) / AII(L)
   90     CONTINUE
  100   CONTINUE
C
C        Find a univariate isotonic regression
C
        CALL ISRE(K, NH(I), Z, AII, WK(1, 1), WK(1, 2), WK(1, 3),
     *            WK(1, 4), WK(1, 5), WK(1, 6), XA, IFAULT)
        IF (IFAULT .NE. 0) THEN
          IFAULT = IFAULT + 2
          RETURN
        ENDIF
        DO 110 J = 1, K
          IF (IT .NE. 1) THEN
            RABS = ABS(XA(J) - SITA(I,J))
            IF (RABS .GT. RMAX) RMAX = RABS
          ENDIF
          SITA(I, J) = XA(J)
  110   CONTINUE
  120 CONTINUE
      IT = IT + 1
      IF (IT .EQ. 2 .OR. RMAX .GT. EPS .AND. IT .LE. ITMAX) GOTO 50
      IF (RMAX .GT. EPS .AND. IT .GT. ITMAX) IFAULT = 7
C
C       Calculate likelihood ratio statistics
C
      STAT = 0.0
      DO 150 L = 1, K
        DO 140 I = 1, IP
          II = I * (1 + I) / 2
          STAT = STAT + COVINV(II, L) * (X(I, L) - SITA(I, L)) ** 2
          IF (I .GT. 1) THEN
            II1 = II - I
            DO 130 J = 1, I - 1
              STAT = STAT + 2.0 * COVINV(II1+J,L) * (X(I,L) - SITA(I,L))
```

```
     *                          * (X(J, L) - SITA(J, L))
  130       CONTINUE
            ENDIF
  140   CONTINUE
  150 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE MISREU(K, IP, IPP, KP, NH, COV, X, EPS, ITMAX, IB,
     *                  COVINV, XA, Y, Z, AII, A, W, WK, ITM, SITA,
     *                  SITAMIN, STATMIN, NHMIN, STATMAT, IFAULT)
C
C        ALGORITHM AS 302.2 APPL. STATIST. (1996), VOL.45, NO.2
C
C        Subroutine to compute a multiple isotonic regression
C        for umbrella ordering with unknown peak
C
      INTEGER I, IFAULT, II, II1, INC, INV, IP, IP1, IPP, IT, ITM,
     *        ITMAX, J, J1, K, KP, L, NULLTY
      REAL DET, EPS, RABS, RMAX, STAT, STATMIN
      REAL A(IP), AII(K), COV(IPP, K), COVINV(IPP, K), SITA(IP, K),
     *     SITAMIN(IP,K), STATMAT(KP), W(IP), WK(K, 6), X(IP, K),
     *     XA(K), Y(K), Z(K)
      INTEGER IB(IP), NH(IP), NHMIN(IP)
C
      EXTERNAL MISRE
C
      IF (KP .LT. K ** IP) THEN
        IFAULT = 8
        RETURN
      ENDIF
C
      INV = 1
      ITM = 0
      STATMIN = 0.99E10
      NUM = 1
      DO 10 I = 1, IP
        NH(I) = 1
   10 CONTINUE
C
C        Calculate multiple isotonic regression for all possible
C        peaks
C
   20 CALL MISRE(K, IP, IPP, NH, INV, COV, X, EPS, ITMAX,
     *           IB, COVINV, XA, Y, Z, AII, A, W, WK, IT, SITA, STAT,
     *           IFAULT)
      IF (IFAULT .NE. 0) RETURN
      STATMAT(NUM) = STAT
      NUM = NUM + 1
      IF (IT .GT. ITM) ITM = IT
C
C        Save the best regression and peak vector
C
      IF (STAT .LE. STATMIN) THEN
        STATMIN = STAT
        DO 40 I = 1, IP
          DO 30 J = 1, K
            SITAMIN(I, J) = SITA(I, J)
   30     CONTINUE
          NHMIN(I) = NH(I)
```

```
   40   CONTINUE
      ENDIF
C
C        Set the next peak
C
      DO 50 I = 1, IP
        NH(I) = NH(I) + 1
        IF (NH(I) .LE. K) GOTO 20
        NH(I) = 1
   50 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE OPART(N, K, NI, MCOUNT, IFAULT)
C
C        ALGORITHM AS 303.1 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Generation of ordered multinomial frequencies.  Given integers
C        N (N >= 0) and K (K >= 1), generates all possible integer arrays
C        NI of size K, whose elements are non-negative, non-decreasing
C        and sum to N.
C
      INTEGER N, K, NI(K), MCOUNT, IFAULT
C
      INTEGER I, I0, I1, J, K1, KSUM, NSUM
C
      EXTERNAL JOB
C
C        Perform domain checks.  Initialize MCOUNT, NI, K1 and KSUM.
C        Set NI(I0) = 0; for I0 = 1 to K-1.
C
      IFAULT = 1
      MCOUNT = 0
      IF(K .LT. 1) RETURN
      IF(N .LT. 0) RETURN
      IFAULT = 0
      K1 = K - 1
      IF(K .EQ. 1) K1 = 1
      KSUM = 0
C
      DO 10 I = 1, K1
         NI(I) = 0
   10 CONTINUE
C
C        Begin with pivot element in position K-1 of array NI
C
      I0 = K1
   20 IF(I0 .EQ. K1) GO TO 40
C
C        Set NI(J); for J = I0+1 to K-1, equal to pivot cell
C        frequency.  Update KSUM.
C
      I1 = I0 + 1
C
      DO 30 J = I1, K1
         NI(J) = NI(I0)
         KSUM = KSUM + NI(I0)
   30 CONTINUE
C
C        Assign NI(K)
C
   40 MCOUNT = MCOUNT + 1
      NSUM = NI(K1)
      NI(K) = N - KSUM
C
      CALL JOB(K, NI, MCOUNT)
C
C        Reset pivot cell to K-1 and update NSUM.  Check upper limit
C        for NI(I0).
C
      I0 = K1
      NSUM = NSUM + NI(K)
   50 IF( NI(I0) .LT. NSUM / (K - I0 + 1) ) GO TO 60
C
```

```
C        Move pivot I0 to next lower cell.  Update KSUM and NSUM.
C
      I0 = I0 - 1
      IF(I0 .LE. 0) RETURN
      KSUM = KSUM - NI(I0+1)
      NSUM = NSUM + NI(I0)
      GO TO 50
C
C        Increment NI(I0) by 1 and update KSUM.
C
   60 NI(I0) = NI(I0) + 1
      KSUM = KSUM + 1
      GO TO 20
      END
C
      SUBROUTINE JOB(K, NI, MCOUNT)
C
C        ALGORITHM AS 303.2 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Evaluates C(X) for X = NI and writes out ordered K-tuple
C
      INTEGER K, NI(K), MCOUNT
C
      REAL SIZE, TIES
      INTEGER J, K1, KTEMP, NTEMP, RK
C
      K1 = K-1
      NTEMP = NI(1)
      SIZE = K
      KTEMP = K
      TIES = 1.0D0
C
      DO 30 J = 1, K1
         KTEMP = KTEMP - 1
         RK = KTEMP
         IF(NI(J + 1) .EQ. NTEMP) GO TO 10
         NTEMP = NI(J + 1)
         TIES = 1.0D0
         GO TO 20
   10    TIES = TIES + 1.0D0
   20    SIZE = SIZE * RK / TIES
   30 CONTINUE
C
      WRITE(*, 40) MCOUNT, SIZE, (NI(J), J = 1, K)
   40 FORMAT(1X,I5,1X,F15.0,7X,10I4/
     *       1X,21X,10I4)
      RETURN
      END
```

```
      SUBROUTINE FISHER (X, M, Y, N, TOTAL, POSSIB, P, IFAULT)
C
C        ALGORITHM AS 304.1 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Fisher's non-parametric randomization test for two small
C        independent random samples
C
      INTEGER M, N, TOTAL, POSSIB, IFAULT
      REAL X(*), Y(*), P
C
      INTEGER MAXSAM, MAXSIZ
      PARAMETER (MAXSAM = 14, MAXSIZ = 3432)
C
C        Important : set MAXSIZ >= COMB(MAXSAM, MAXSAM / 2)
C
      INTEGER K, SIZE1, SIZE2, WS3(MAXSAM), COUNT
      REAL SUMX, SUMY, WS1(MAXSIZ), WS2(MAXSIZ), SUMM
C
      INTEGER COMB, TRADES
      REAL MEAN, SUM
      EXTERNAL COMB, SUM, TRADES
C
      EXTERNAL CMPLMT, EXCHNG, KTRADE
C
      MEAN(SUMM, COUNT) = SUMM / REAL(COUNT)
C
      IF (COMB(MAXSAM, MAXSAM/2) .GT. MAXSIZ) THEN
         IFAULT = 1
      ELSE IF (M .GT. MAXSAM .OR. N .GT. MAXSAM) THEN
              IFAULT = 2
           ELSE
              IFAULT = 0
              SUMX = SUM(X, M)
              SUMY = SUM(Y, N)
              IF (MEAN(SUMX, M) .GT. MEAN(SUMY, N)) THEN
                 CALL EXCHNG(X, M, Y, N, SUMX, SUMY)
           END IF
           TOTAL = 0
           IF (M .EQ. N) THEN
              DO 10 K = 1, (M-1)/2
                 CALL KTRADE(X, M, WS1, SIZE1, WS3, K)
                 CALL KTRADE(Y, N, WS2, SIZE2, WS3, K)
                 TOTAL = TOTAL + TRADES(WS1, SIZE1, WS2, SIZE2)
                 CALL CMPLMT(WS1, SIZE1, SUMX)
                 CALL CMPLMT(WS2, SIZE2, SUMY)
                 TOTAL = TOTAL + TRADES(WS1, SIZE1, WS2, SIZE2)
   10         CONTINUE
              IF (MOD(M, 2) .EQ. 0) THEN
                 CALL KTRADE(X, M, WS1, SIZE1, WS3, K)
                 CALL KTRADE(Y, N, WS2, SIZE2, WS3, K)
                 TOTAL = TOTAL + TRADES(WS1, SIZE1, WS2, SIZE2)
              END IF
         ELSE
            DO 20 K = 1, MIN(M, N)
               CALL KTRADE(X, M, WS1, SIZE1, WS3, K)
               CALL KTRADE(Y, N, WS2, SIZE2, WS3, K)
               TOTAL = TOTAL + TRADES(WS1, SIZE1, WS2, SIZE2)
   20       CONTINUE
         END IF
         POSSIB = COMB(M+N, M)
         P = REAL(TOTAL + 1) / REAL(POSSIB)
```

```
      END IF
C
      RETURN
      END
C
      SUBROUTINE EXCHNG (X, M, Y, N, SX, SY)
C
C        ALGORITHM AS 304.2 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Exchanges the sample data.  Assumes both X and Y have been
C        previously dimensioned to at least max(M, N) elements
C
      INTEGER M, N
      REAL X(*), Y(*), SX, SY
C
      INTEGER C, K
      REAL TEMP
C
      TEMP = SX
      SX = SY
      SY = TEMP
C
      C = MIN(M, N)
      DO 10 K = 1, C
         TEMP = X(K)
         X(K) = Y(K)
         Y(K) = TEMP
   10 CONTINUE
      IF (M .GT. N) THEN
         DO 20 K = C+1, M
            Y(K) = X(K)
   20    CONTINUE
         N = M
         M = C
      ELSE IF (M .LT. N) THEN
         DO 30 K = C+1, N
            X(K) = Y(K)
   30    CONTINUE
         M = N
         N = C
      END IF
C
      RETURN
      END
C
      SUBROUTINE KTRADE (W, K, WPRIME, KPRIME, WS, R)
C
C        ALGORITHM AS 304.3 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Generates and sorts the sums of the R-combinations of the
C        elements of W
C
      INTEGER K, KPRIME, WS(*), R
      REAL W(*), WPRIME(*)
C
      INTEGER COMB
      REAL SUM
      EXTERNAL COMB, SUM
C
      EXTERNAL CMPLMT, GENER, SORT
C
```

```
      KPRIME = COMB(K, R)
      IF (R .LE. K - R .OR. R .EQ. K) THEN
         CALL GENER(W, K, WPRIME, KPRIME, WS, R)
         CALL SORT(WPRIME, KPRIME)
      ELSE
         CALL GENER(W, K, WPRIME, KPRIME, WS, K - R)
         CALL SORT(WPRIME, KPRIME)
         CALL CMPLMT(WPRIME, KPRIME, SUM(W, K))
      ENDIF
C
      RETURN
      END
C
      INTEGER FUNCTION TRADES (XPRIME, MPRIME, YPRIME, NPRIME)
C
C        ALGORITHM AS 304.4 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Returns the number of 1-for-1 trades that refutes the null
C        hypothesis.  Assumes that XPRIME has the smaller mean and
C        that both arrays are sorted in ascending order.
C
      INTEGER MPRIME, NPRIME
      REAL XPRIME(*), YPRIME(*)
C
      INTEGER I, J
C
      TRADES = 0
      I = 1
      J = 1
   10 IF (J .GT. NPRIME) GOTO 40
   20 IF (XPRIME(I) .GE. YPRIME(J)) GOTO 30
      I = I + 1
      IF (I .LE. MPRIME) GOTO 20
   30 TRADES = TRADES + (MPRIME - I + 1)
      J = J + 1
      IF (I .LE. MPRIME) GOTO 10
C
   40 RETURN
      END
C
      SUBROUTINE CMPLMT (WPRIME, KPRIME, SUM)
C
C        ALGORITHM AS 304.5 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Reverse and complement the data in WPRIME
C
      INTEGER KPRIME
      REAL WPRIME(*), SUM
C
      INTEGER I, J
      REAL TEMP
C
      J = KPRIME
      DO 10 I = 1, KPRIME / 2 + MOD(KPRIME, 2)
         TEMP = WPRIME(I)
         WPRIME(I) = REAL(DBLE(SUM) - DBLE(WPRIME(J)))
         WPRIME(J) = REAL(DBLE(SUM) - DBLE(TEMP))
         J = J - 1
   10 CONTINUE
C
      RETURN
```

```
      END
C
      SUBROUTINE GENER (W, N, WPRIME, NPRIME, INDEX, R)
C
C        ALGORITHM AS 304.6 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Computes an array of sums of the various R-combinations of
C        the elements of W
C
      INTEGER N, NPRIME, R, INDEX(R)
      REAL W(N), WPRIME(NPRIME)
C
      INTEGER I, J
      DOUBLE PRECISION SUM
      LOGICAL INIT
C
      EXTERNAL NEXT
C
      INIT = .TRUE.
C
      DO 20 I = 1, NPRIME
         CALL NEXT(INDEX, R, N, INIT)
         SUM = 0.0D0
         DO 10 J = 1, R
            SUM = SUM + DBLE(W(INDEX(J)))
   10    CONTINUE
         WPRIME(I) = REAL(SUM)
   20 CONTINUE
C
      RETURN
      END
C
      SUBROUTINE NEXT (RCOMBO, R, N, INIT)
C
C        ALGORITHM AS 304.7 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Accepts some R-combination of the first N integers and then
C        computes the next R-combination in the lexicographic
C        ordering of the N! / (R! * (N - R)!) such R-combinations.
C        Returns the first R-combination if the initialization
C        indicator is .true. and then resets the indicator.
C
      INTEGER R, N, RCOMBO(R)
      LOGICAL INIT
C
      INTEGER I, J, D
C
      IF (INIT) THEN
         DO 10 I = 1, R
            RCOMBO(I) = I
   10    CONTINUE
         INIT = .FALSE.
      ELSE
         D = N - R
         J = R
C
C        The counter J is not prevented from going out of bounds
C        which will happen if there is no next R-combination
C
   20    IF (RCOMBO(J) .LT. D + J) GOTO 30
         J = J - 1
```

```
            GOTO 20
      30    RCOMBO(J) = RCOMBO(J) + 1
            DO 40 I = J + 1, R
               RCOMBO(I) = RCOMBO(I - 1) + 1
      40    CONTINUE
         END IF
C
         RETURN
         END
C
C        General purpose subroutines
C
         SUBROUTINE SORT (X, N)
C
C        ALGORITHM AS 304.8 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Sorts the N values stored in array X in ascending order
C
         INTEGER N
         REAL X(N)
C
         INTEGER I, J, INCR
         REAL TEMP
C
         INCR = 1
C
C        Loop : calculate the increment
C
      10 INCR = 3 * INCR + 1
         IF (INCR .LE. N) GOTO 10

C
C        Loop : Shell-Metzner sort
C
      20 INCR = INCR / 3
         I = INCR + 1
      30 IF (I .GT. N) GOTO 60
         TEMP = X(I)
         J = I
      40 IF (X(J - INCR) .LT. TEMP) GOTO 50
         X(J) = X(J - INCR)
         J = J - INCR
         IF (J .GT. INCR) GOTO 40
      50 X(J) = TEMP
         I = I + 1
         GOTO 30
      60 IF (INCR .GT. 1) GOTO 20
C
         RETURN
         END
C
         INTEGER FUNCTION COMB (N, K)
C
C        ALGORITHM AS 304.9 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Returns the number of combinations of N things taken K at a
C        time or 0 if the parameters are incompatible
C
         INTEGER N, K
C
         INTEGER M, I
```

```
      REAL NUMER, DENOM, FACT
      EXTERNAL FACT
C
      IF (K .LT. 0 .OR. K .GT. N) THEN
         COMB = 0
      ELSE
         M = N - K
         NUMER = 1.0
         DO 10 I = N, 1 + MAX(K, M), -1
            NUMER = NUMER * REAL(I)
   10    CONTINUE
         DENOM = FACT(MIN(K, M))
         COMB = NINT(NUMER / DENOM)
      END IF
C
      RETURN
      END
C
      REAL FUNCTION FACT (N)
C
C        ALGORITHM AS 304.10 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Returns the factorial of N or 0.0 if the parameter is
C        negative
C
      INTEGER N
C
      INTEGER I
C
      IF (N .LT. 0) THEN
         FACT = 0.0
      ELSE
        FACT = 1.0
        DO 10 I = 2, N
           FACT = FACT * REAL(I)
   10   CONTINUE
      END IF
C
      RETURN
      END
C
      REAL FUNCTION SUM (X, N)
C
C        ALGORITHM AS 304.11 APPL.STATIST. (1996), VOL.45, NO.3
C
C        Returns the sum of the values stored in X
C
      INTEGER N
      REAL X(N)
C
      INTEGER I
      DOUBLE PRECISION ACCUM
C
      ACCUM = 0.0D0
      DO 10 I = 1, N
         ACCUM = ACCUM + DBLE(X(I))
   10 CONTINUE
      SUM = REAL(ACCUM)
C
      RETURN
      END
```

```
      SUBROUTINE ARLM(K,H,U,DELTA,ARL,IFAULT)
C
C        ALGORITHM AS 305.1 APPL. STATIST. (1996), VOL.45, NO.4
C
C        Compute the ARL of a CUSUM mean chart with linear drift
C
      DOUBLE PRECISION K,H,U,DELTA,ARL
      INTEGER IFAULT
C
      DOUBLE PRECISION SIGMA
      INTEGER NN, NSIDE, NTYPE
C
      EXTERNAL COMARL
C
      NTYPE=1
      NSIDE=1
      CALL COMARL(NTYPE,NSIDE,NN,K,H,U,DELTA,SIGMA,ARL,IFAULT)
C
      RETURN
      END
C
      SUBROUTINE ARLV(NSIDE,NN,K,H,U,SIGMA,ARL,IFAULT)
C
C        ALGORITHM AS 305.2 APPL. STATIST. (1996), VOL.45, NO.4
C
C        Compute the ARL of a CUSUM variance chart
C
      INTEGER NSIDE, NN, IFAULT
      DOUBLE PRECISION K,H,U,SIGMA,ARL
C
      INTEGER NTYPE
      DOUBLE PRECISION DELTA
C
      EXTERNAL COMARL
C
      NTYPE=2
      CALL COMARL(NTYPE,NSIDE,NN,K,H,U,DELTA,SIGMA,ARL,IFAULT)
C
      RETURN
      END

C
      SUBROUTINE COMARL(NTYPE,NSIDE,NN,K,H,U,DELTA,SIGMA,ARL,IFAULT)
C
C        ALGORITHM AS 305.3 APPL. STATIST. (1996), VOL.45, NO.4
C
C        Compute the ARL of a CUSUM chart or a CUSUM variance chart
C
      INTEGER NTYPE, NSIDE, NN, IFAULT
      DOUBLE PRECISION K,H,U,DELTA,SIGMA,ARL
C
      INTEGER I, IA, IB, IP1, IFLAG, IPIVOT(25), J, JJ, JP1, JP2,
     *        MMM, MP1, ND2, NPTS, NQM1, NQUAD, NSIDE2
      DOUBLE PRECISION A(25, 25), ALPHA, B(25), BETA, DMEAN(3000),
     *                 P(25), SIDE, SK, SS, W(25), WK(25), X(25),
     *                 XOLD(25)
C
      DOUBLE PRECISION CDFLG, PDFLG, PDFN
      REAL ALNORM, DNORM
C
      EXTERNAL FACTOR, SUBST, SETUP
```

```
C
C        Gaussian quadrature abscissas
C
      P(1)=0.9951872199970213D0
      P(2)=0.9747285559713095D0
      P(3)=0.9382745520027327D0
      P(4)=0.8864155270044010D0
      P(5)=0.8200019859739029D0
      P(6)=0.7401241915785543D0
      P(7)=0.6480936519369755D0
      P(8)=0.5454214713888395D0
      P(9)=0.4337935076260451D0
      P(10)=0.3150426796961634D0
      P(11)=0.1911188674736163D0
      P(12)=0.0640568928626056D0
C
C        Gaussian quadrature weights
C
      W(1)=0.0123412297999872D0
      W(2)=0.0285313886289337D0
      W(3)=0.0442774388174198D0
      W(4)=0.0592985849154368D0
      W(5)=0.0733464814110803D0
      W(6)=0.0861901615319533D0
      W(7)=0.0976186521041139D0
      W(8)=0.1074442701159656D0
      W(9)=0.1155056680537256D0
      W(10)=0.1216704729278034D0
      W(11)=0.1258374563468283D0
      W(12)=0.1279381953467521D0
C
C        Initialise variables
C
      NPTS=25
      NQUAD=NPTS-1
      ND2=NQUAD/2
      SIDE=NSIDE
      SS=NN
      ALPHA=(SS-1.0D0)/2.0D0
      BETA=(SIGMA*SIGMA)/ALPHA
      SK=SIDE*K
C
C        Set IFAULT to be 0, 1, 2, 3, 4, 5, 6
C
      IFAULT=0
      IF(NTYPE.EQ.1.AND.(DELTA.GE.0.0D0.AND.
     *DELTA.LT.0.0001D0)) IFAULT=1
      IF(NTYPE.EQ.1.AND.DELTA.LT.0.0D0) IFAULT=2
      NSIDE2=NSIDE*NSIDE
      IF(NTYPE.EQ.2.AND.NSIDE2.NE.1) IFAULT=3
      IF(NTYPE.EQ.2.AND.SIGMA.LE.0.0D0) IFAULT=4
      IF(NTYPE.EQ.1.AND.(U.LT.0.0D0.OR.U.GT.H)) IFAULT=5
      IF(NTYPE.EQ.2.AND.NSIDE.EQ.1.AND.
     *(U.LT.0.0D0.OR.U.GT.H)) IFAULT=5
      IF(NTYPE.EQ.2.AND.NSIDE.EQ.-1.AND.
     *(U.LT.-H.OR.U.GT.0.0D0)) IFAULT=5
      IF(NTYPE.EQ.2.AND.NN.LE.1) IFAULT=6
      IF(IFAULT.GE.2) RETURN
C
C        Obtain the full Gaussian abscissas and weights
C
```

```
        DO 2 I=1,ND2
          P(NPTS-I)=-P(I)
          W(NPTS-I)=W(I)
    2   CONTINUE
C
C         Transform the Gaussian abscissas and weights
C         from the interval (-1,1) to (0,h) or (-h,0)
C
        DO 3 I=1,NQUAD
          W(I)=H*W(I)/2.0D0
          P(I)=H*P(I)/2.0D0 + SIDE*H/2.0D0
    3   CONTINUE
C
C         Compute the ARL of a CUSUM variance chart under step shift
C
        IF(NTYPE.EQ.2) THEN
C
C         Replace the integral equation by a system of linear
C         equations AX=B
C
          B(NPTS)=-1.D0
          A(1,1)=-1.0D0+CDFLG(SK,ALPHA,BETA)
          IF(NSIDE.EQ.-1) A(1,1)=-1.0D0-A(1,1)
          DO 4 I=1,NQUAD
            A(1,I+1)=W(I)*PDFLG(SK+P(I),ALPHA,BETA)
            B(I)=-1.D0
    4     CONTINUE
          DO 5 I=1,NQUAD
            IP1=I+1
            A(IP1,1)=CDFLG(SK-P(I),ALPHA,BETA)
            IF(NSIDE.EQ.-1) A(IP1,1)=1.0D0-A(IP1,1)
    5     CONTINUE
          DO 7 I=1,NQUAD
            DO 6 J=1,NQUAD
              IP1=I+1
              JP1=J+1
              IF(I.EQ.J) THEN
                A(IP1,JP1)=W(I)*PDFLG(SK,ALPHA,BETA)-1.0D0
              ELSE
                A(IP1,JP1)=W(J)*PDFLG(P(J)-P(I)+SK,
     *          ALPHA,BETA)
              ENDIF
    6       CONTINUE
    7     CONTINUE
C
C         Call the subroutines FACTOR and SUBST to solve the system
C         of equations for X. Set IFAULT to be 7 if matrix A is
C         singular.
C
          CALL FACTOR(A,NPTS,WK,IPIVOT,IFLAG)
          IF(IFLAG.EQ.0) THEN
            IFAULT=7
            RETURN
          ENDIF
          CALL SUBST(A,IPIVOT,B,NPTS,X)
C
C         Compute L(u) of a CUSUM variance chart
C
          ARL=CDFLG(SK-U,ALPHA,BETA)
          IF(NSIDE.EQ.-1) ARL=1.0D0-ARL
          ARL=1.D0+X(1)*ARL+W(1)*X(2)*PDFLG(P(1)+SK-U,
```

```
     *  ALPHA,BETA)
        NQM1=NQUAD-1
        DO 8 J=1,NQM1
          JP1=J+1
          JP2=J+2
          ARL=ARL+W(JP1)*X(JP2)*PDFLG(P(JP1)+SK-U,
     *    ALPHA,BETA)
    8   CONTINUE
C
C        Compute the ARL of a CUSUM mean chart under linear drift
C
      ELSE
C
C        Compute the process mean mu_0, mu_1, ..., mu_m
C
        CALL SETUP(DELTA,DMEAN,MP1)
C
C        Replace the integral equation by a system of linear
C        equations AX=B
C
        DO 20 I=1,NPTS
          B(I)=-1.0D0
          DO 10 J=1,NPTS
            IF(I.LE.NQUAD.AND.J.EQ.I) THEN
              A(I,J)=W(J)*PDFN(P(J)+K-P(I),
     *        DMEAN(MP1),1.0D0)-1.0D0
            ELSEIF(I.LE.NQUAD.AND.J.LE.NQUAD) THEN
              A(I,J)=W(J)*PDFN(P(J)+K-P(I),
     *        DMEAN(MP1),1.0D0)
            ELSEIF(I.LE.NQUAD.AND.J.EQ.NPTS) THEN
              DNORM=K-P(I)-DMEAN(MP1)
              A(I,J)=ALNORM(DNORM,.FALSE.)
            ELSEIF(I.EQ.NPTS.AND.J.LT.I) THEN
              A(I,J)=W(J)*PDFN(P(J)+K,
     *        DMEAN(MP1),1.0D0)
            ELSE
              DNORM=K-DMEAN(MP1)
              A(I,J)=ALNORM(DNORM,.FALSE.)-1.0D0
            ENDIF
   10     CONTINUE
   20   CONTINUE
C
C        Call the subroutines FACTOR and SUBST to solve the system
C        of equations for X.  Set IFAULT to be 7 if matrix A is
C        singular.
C
        CALL FACTOR(A,NPTS,WK,IPIVOT,IFLAG)
        IF(IFLAG.EQ.0) THEN
          IFAULT=7
          RETURN
        ENDIF
        CALL SUBST(A,IPIVOT,B,NPTS,X)
C
C        Compute intermediate ARL functions
C
        MMM=MP1-1
        DO 80 J=2,MMM
          JJ=MP1-J+1
          DO 60 I=1,NPTS
            XOLD(I)=X(I)
            X(I)=1.0D0
```

```
   60     CONTINUE
          DO 75 IA=1,NPTS
            IF(IA.EQ.NPTS) THEN
              DNORM = K-DMEAN(JJ)
            ELSE
              DNORM = K-P(IA)-DMEAN(JJ)
            ENDIF
            X(IA) = X(IA)+ALNORM(DNORM,.FALSE.)*XOLD(NPTS)
            DO 70 IB=1,NQUAD
              IF(IA.EQ.NPTS) THEN
                X(IA) = X(IA)+W(IB)*XOLD(IB)*
     *          PDFN(P(IB)+K,DMEAN(JJ),1.0D0)
              ELSE
                X(IA) = X(IA)+W(IB)*XOLD(IB)*
     *          PDFN(P(IB)+K-P(IA),DMEAN(JJ),1.0D0)
              ENDIF
   70       CONTINUE
   75     CONTINUE
   80   CONTINUE
C
C        Compute L_0(u,0)
C
        ARL=1.0D0
        DNORM = K-U-DMEAN(1)
        ARL = ARL+ALNORM(DNORM,.FALSE.)*X(NPTS)
        DO 90 J=1,NQUAD
          ARL=ARL+W(J)*PDFN(P(J)+K-U,DMEAN(1),1.0D0)*X(J)
   90   CONTINUE
      ENDIF
C
      RETURN
      END
C
      SUBROUTINE SETUP(DELTA,DMEAN,MP1)
C
C        ALGORITHM AS 305.4 APPL. STATIST. (1996), VOL.45, NO.4
C
C
C        Determine MP1 and DMEAN where
C        DMEAN(j) = process mean at (j-1)th sample;
C                   for j=1, 2, ...,MP1
C
      DOUBLE PRECISION DELTA, DMEAN(3000)
      INTEGER MP1
C
      INTEGER J, JM1
C
      IF(DELTA.LT.0.0001) THEN
        MP1=1
      ELSEIF(DELTA.LE.0.001) THEN
        MP1=3000
      ELSEIF(DELTA.LE.0.002) THEN
        MP1=2000
      ELSEIF(DELTA.LE.0.005) THEN
        MP1=1000
      ELSEIF(DELTA.LE.0.010) THEN
        MP1=500
      ELSEIF(DELTA.LE.0.015) THEN
        MP1=300
      ELSEIF(DELTA.LE.0.020) THEN
        MP1=200
```

```
      ELSEIF(DELTA.LE.0.025) THEN
        MP1=150
      ELSEIF(DELTA.LE.0.040) THEN
        MP1=100
      ELSEIF(DELTA.LE.0.050) THEN
        MP1=70
      ELSEIF(DELTA.LE.0.080) THEN
        MP1=60
      ELSEIF(DELTA.LE.0.150) THEN
        MP1=40
      ELSEIF(DELTA.LE.0.200) THEN
        MP1=30
      ELSEIF(DELTA.LE.0.400) THEN
        MP1=20
      ELSEIF(DELTA.LE.1.000) THEN
        MP1=15
      ELSE
        MP1=10
      ENDIF
      DMEAN(1)=0.0D0
      IF(MP1.GE.2) THEN
        DO 10 J=2,MP1
          JM1=J-1
          DMEAN(J)=DMEAN(JM1)+DELTA
   10   CONTINUE
      ENDIF
C
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION PDFN(X,XMU,SIGMA)
C
C        ALGORITHM AS 305.5 APPL. STATIST. (1996), VOL.45, NO.4
C
C        Compute the normal density function at X where XMU is the
C        mean and SIGMA is the sandard deviation
C
      DOUBLE PRECISION X, XMU, SIGMA
C
      DOUBLE PRECISION ARG
C
      ARG=-0.5D0*(X-XMU)*(X-XMU)/(SIGMA*SIGMA)
      PDFN=(3.989422804014327D-1/SIGMA)*DEXP(ARG)
C
      RETURN
      END
```

```
      SUBROUTINE BIB(AMAT,A,B,BMAT,C,D,NEWMAT,FLAG)
C
C        ALGORITHM AS 306 APPL.STATIST. (1996), VOL.45, NO.4
C
C        Calculation of a BIB product operation on any two matrices
C
      INTEGER A, B, C, D
      REAL AMAT(A,B), BMAT(C,D), NEWMAT(A,B*D)
      LOGICAL FLAG
C
      INTEGER CT, I, J, K, PT
C
C     This block of code sets the logical variable FLAG to F and exits
C     the subroutine if the number of non-zero elements in each column
C     of A does not equal the number of rows of B
C
      DO 20 I = 1, B
         CT=0
         DO 10 J=1, A
            IF (AMAT(J,I).NE.0.0) CT = CT + 1
   10    CONTINUE
         FLAG=CT.EQ.C
         IF(.NOT.FLAG) GOTO 70
   20 CONTINUE
C
C        This block of code computes the BIB product
C
      DO 60 I = 1, B
         PT = 1
         DO 50 J = 1, A
            IF(AMAT(J,I).EQ.0.0) THEN
               DO 30 K = 1, D
                 NEWMAT(J,(I*D)-D+K)=0.0
   30          CONTINUE
            ELSE
               DO 40 K = 1, D
                  NEWMAT(J,(I*D)-D+K)=AMAT(J,I)*BMAT(PT,K)
   40          CONTINUE
               PT=PT+1
            END IF
   50    CONTINUE
   60 CONTINUE
   70 RETURN
      END
```

```
      SUBROUTINE DEPTH(U, V, N, X, Y, ALPHA, F, SDEP, HDEP)
C
C        ALGORITHM AS 307.1 APPL.STATIST. (1996), VOL.45, NO.4
C
C        Calculation of the simplicial depth and the halfspace
C        depth
C
      INTEGER N, F(N)
      REAL U, V, X(N), Y(N), ALPHA(N), SDEP, HDEP
C
      INTEGER GI, I, J, JA, JB, KI, NBAD, NF, NN, NN2, NT, NU, NUMH,
     *        NUMS
      REAL ALPHK, ANGLE, BETAK, D, EPS, P, P2, XU, YU
C
      EXTERNAL K, SORT
C
      NUMS = 0
      NUMH = 0
      SDEP = 0.0
      HDEP = 0.0
C
      IF (N.LT.1) RETURN
C
      P = ACOS(-1.0)
      P2 = P * 2.0
      EPS = 0.000001
      NT = 0
C
C        Construct the array ALPHA
C
      DO 10 I = 1, N
          D = SQRT((X(I)-U) * (X(I)-U) + (Y(I)-V) * (Y(I)-V))
          IF (D.LE.EPS) THEN
              NT = NT + 1
          ELSE
              XU = (X(I) - U) / D
              YU = (Y(I) - V) / D
              IF (ABS(XU).GT.ABS(YU)) THEN
                  IF (X(I).GE.U) THEN
                      ALPHA(I-NT) = ASIN(YU)
                      IF(ALPHA(I-NT).LT.0.0) THEN
                          ALPHA(I-NT) = P2 + ALPHA(I-NT)
                      ENDIF
                  ELSE
                      ALPHA(I-NT) = P - ASIN(YU)
                  ENDIF
              ELSE
                  IF (Y(I).GE.V) THEN
                      ALPHA(I-NT) = ACOS(XU)
                  ELSE
                      ALPHA(I-NT) = P2 - ACOS(XU)
                  ENDIF
              ENDIF
              IF (ALPHA(I-NT).GE.(P2 - EPS)) ALPHA(I-NT) = 0.0
          ENDIF
   10 CONTINUE
      NN = N - NT
      IF (NN.LE.1) GOTO 60
C
C        Sort the array ALPHA
C
```

```
      CALL SORT(ALPHA, NN)
C
C         Check whether theta=(U,V) lies outside the data cloud
C
      ANGLE = ALPHA(1) - ALPHA(NN) + P2
      DO 20 I = 2, NN
          ANGLE = AMAX1(ANGLE, (ALPHA(I) - ALPHA(I-1)))
   20 CONTINUE
      IF (ANGLE.GT.(P + EPS)) GOTO 60
C
C        Make smallest alpha equal to zero, and
C        compute NU = number of alpha < pi
C
      ANGLE = ALPHA(1)
      NU = 0
      DO 30 I = 1, NN
          ALPHA(I) = ALPHA(I) - ANGLE
          IF (ALPHA(I).LT.(P - EPS)) NU = NU + 1
   30 CONTINUE
      IF (NU.GE.NN) GOTO 60
C
C        Mergesort the alpha with their antipodal angles beta, and
C        at the same time update I, F(I), and NBAD
C
      JA = 1
      JB = 1
      ALPHK = ALPHA(1)
      BETAK = ALPHA(NU+1) - P
      NN2 = NN * 2
      NBAD = 0
      I = NU
      NF = NN
      DO 40 J = 1, NN2
          IF ((ALPHK + EPS).LT.BETAK) THEN
              NF = NF + 1
              IF (JA.LT.NN) THEN
                  JA = JA + 1
                  ALPHK = ALPHA(JA)
              ELSE
                  ALPHK = P2 + 1.0
              ENDIF
          ELSE
              I = I + 1
              IF (I.EQ.(NN + 1)) THEN
                  I = 1
                  NF = NF - NN
              ENDIF
              F(I) = NF
              NBAD = NBAD + K((NF-I), 2)
              IF (JB.LT.NN) THEN
                  JB = JB + 1
                  IF ((JB + NU).LE.NN) THEN
                      BETAK = ALPHA(JB+NU) - P
                  ELSE
                      BETAK = ALPHA(JB+NU-NN) + P
                  ENDIF
              ELSE
                  BETAK = P2 + 1.0
              ENDIF
          ENDIF
   40 CONTINUE
```

```
      NUMS = K(NN, 3) - NBAD
C
C        Computation of NUMH for halfspace depth
C
      GI = 0
      JA = 1
      ANGLE = ALPHA(1)
      NUMH = MIN0(F(1), (NN - F(1)))
      DO 50 I = 2, NN
          IF(ALPHA(I).LE.(ANGLE + EPS)) THEN
              JA = JA + 1
          ELSE
              GI = GI + JA
              JA = 1
              ANGLE = ALPHA(I)
          ENDIF
          KI = F(I) - GI
          NUMH = MIN0(NUMH, MIN0(KI, (NN-KI)))
   50 CONTINUE
C
C        Adjust for the number NT of data points equal to theta
C
   60 NUMS = NUMS + K(NT, 1)*K(NN, 2) + K(NT, 2)*K(NN, 1) + K(NT, 3)
      IF (N.GE.3) SDEP = (NUMS + 0.0) / (K(N, 3) + 0.0)
      NUMH = NUMH + NT
      HDEP = (NUMH + 0.0) / (N + 0.0)
C
      RETURN
      END
C
      INTEGER FUNCTION K(M, J)
C
C        ALGORITHM AS 307.2 APPL.STATIST. (1996), VOL.45, NO.4
C
C        Returns the value zero if M < J; otherwise computes the
C        number of combinations of J out of M
C
      INTEGER M, J
C
      IF (M.LT.J) THEN
         K = 0
      ELSE
         IF (J.EQ.1) K = M
         IF (J.EQ.2) K = (M * (M - 1)) / 2
         IF (J.EQ.3) K = (M * (M - 1) * (M - 2)) / 6
      ENDIF
C
      RETURN
      END
```

```
      REAL FUNCTION NCBETA (A, B, LAMBDA, X, ERRMAX, IFAULT)
C
C       ALGORITHM AS 310 APPL. STATIST. (1997), VOL. 46, NO. 1
C
C       Computes the cumulative distribution function of a
C       non-central beta random variable
C
      REAL A, B, LAMBDA, X, ERRMAX
      INTEGER IFAULT
C
      INTEGER XJ, M, I, J, ITER1, ITER2, ITERLO, ITERHI
C
C       Local variable XJ gives the number of iterations taken
C
      REAL FX, GX, TEMP, FTEMP, EBD, ERRBD,
     *  Q, R, S, T, S0, S1, T0, T1, SUM, PSUM,
     *  C, HALF, ONE, ZERO, FIVE, BETA,
     *  ALNGAM, BETAIN, BETANC, GAMMAD
C
      EXTERNAL ALNGAM, BETAIN, BETANC, GAMMAD
C
      DATA ZERO, HALF, ONE, FIVE /
     *  0.0E+00, 0.5E+00, 1.0E+00, 5.0E+00 /
C
      NCBETA = X
C
C       Check for admissibility of parameters
C
      IFAULT = 3
      IF (LAMBDA .LE. ZERO .OR. A .LE. ZERO .OR. B .LE. ZERO) RETURN
      IFAULT = 2
      IF (X .LT. ZERO .OR. X .GT. ONE) RETURN
      IFAULT = 1
      IF (X .EQ. ZERO .OR. X .EQ. ONE) RETURN
      IFAULT = 0
C
      C = LAMBDA * HALF
      XJ = ZERO
C
      IF (LAMBDA .LT. 54.0) THEN
C
C       AS 226 as it stands is sufficient in this situation
C
        NCBETA = BETANC(X, A, B, LAMBDA, IFAULT)
        RETURN
      ELSE
        M = INT(C + HALF)
        ITERLO = M - FIVE * SQRT(M)
        ITERHI = M + FIVE * SQRT(M)
        T = - C + M * ALOG(C) - ALNGAM(M + ONE)
        Q = EXP(T)
        R = Q
        PSUM = Q

        BETA = ALNGAM(A+ M) + ALNGAM(B) - ALNGAM(A+ M +B)
        S1 = (A + M) * ALOG(X) + B * ALOG(ONE - X) - ALOG(A + M) - BETA
        GX = EXP(S1)
        FX = GX
        TEMP = BETAIN(X, A + M, B, BETA, IFAULT)
        FTEMP= TEMP
        XJ = XJ + ONE
```

```
        SUM = Q - TEMP
        ITER1= M
c
c      The first set of iterations starts from M and goes downwards
c
   20   IF (ITER1 .LT. ITERLO) GO TO 30
        IF (Q .LT. ERRMAX) GO TO 30
        Q = Q - ITER1 / C
        XJ = XJ + ONE
        GX = (A + ITER1) / (X * (A + B + ITER1 - ONE)) * GX
        ITER1= ITER1- ONE
        TEMP =TEMP + GX
        PSUM = PSUM + Q
        SUM = SUM + Q * TEMP
        GO TO 20
   30   T0 = ALNGAM(A + B) - ALNGAM(A + ONE) - ALNGAM(B)
        S0 = A * ALOG(X) + B * ALOG(ONE - X)
c
        DO 40 I=1, ITER1
          J = I - ONE
          S = S + EXP(T0 + S0 + J * ALOG(X))
          T1 = ALOG(A + B + J) - ALOG(A + ONE + J) + T0
          T0 = T1
   40   CONTINUE
c
c       Compute the first part of error bound
c
        ERRBD=(ONE- GAMMAD(C,FLOAT(ITER1),IFAULT))*(TEMP + S)
        Q = R
        TEMP = FTEMP
        GX = FX
        ITER2 = M
   50   EBD = ERRBD + (ONE - PSUM) * TEMP
        IF (EBD .LT. ERRMAX .OR. ITER2 .GE. ITERHI) GO TO 60
        ITER2 = ITER2 + ONE
        XJ = XJ + ONE
        Q = Q * C / ITER2
        PSUM = PSUM + Q
        TEMP = TEMP - GX
        GX = X * (A + B + ITER2 - ONE) / (A + ITER2) * GX
        SUM = SUM + Q * TEMP
        GO TO 50
   60   CONTINUE
      END IF
   70 NCBETA = SUM
c
      RETURN
      END



      REAL FUNCTION BETANC(X, A, B, LAMBDA, IFAULT)
C
C     ALGORITHM AS226 APPL. STATIST. (1987) VOL. 36, NO. 2
C     Incorporates modification AS R84 from AS vol. 39, pp311-2, 1990
C
C     Returns the cumulative probability of X for the non-central beta
C     distribution with parameters A, B and non-centrality LAMBDA
C
C     Auxiliary routines required: ALOGAM - log-gamma function (ACM
C     291 or AS 245), and BETAIN - incomplete-beta function (AS 63)
```

```
C
      REAL A, AX, B, BETA, C, ERRBD, ERRMAX, GX, HALF, LAMBDA, ONE, Q,
     *     SUMQ, TEMP, X, XJ, ZERO
      REAL A0, X0, UALPHA
C
C     Change ERRMAX and ITRMAX if desired ...
C
      DATA ERRMAX, ITRMAX /1.0E-6, 100/, UALPHA /5.0/
C
      DATA ZERO, HALF, ONE /0.0, 0.5, 1.0/
C
      BETANC = X
C
      IFAULT = 2
      IF (LAMBDA .LT. ZERO .OR. A .LE. ZERO .OR. B .LE. ZERO) RETURN
      IFAULT = 3
      IF (X .LT. ZERO .OR. X .GT. ONE) RETURN
      IFAULT = 0
      IF (X .EQ. ZERO .OR. X .EQ. ONE) RETURN
C
      C = LAMBDA * HALF
C
C     Initialize the series ...
C
      X0 = INT( MAX(C - UALPHA*SQRT(C), ZERO) )
      A0 = A + X0
      BETA = ALOGAM(A0, IFAULT) + ALOGAM(B, IFAULT) -
     *       ALOGAM(A0+B, IFAULT)
      TEMP = BETAIN(X, A0, B, BETA, IFAULT)
      GX = EXP(A0 * LOG(X) + B * LOG(ONE - X) - BETA - LOG(A0))
      IF (A0 .GT. A) THEN
        Q = EXP(-C + X0*LOG(C)) - ALOGAM(X0 + ONE, IFAULT)
      ELSE
        Q = EXP(-C)
      END IF
      XJ = ZERO
      AX = Q * TEMP
      SUMQ = ONE - Q
      BETANC = AX
C
C     Recur over subsequent terms until convergence is achieved...
C
   10 XJ = XJ + ONE
      TEMP = TEMP - GX
      GX = X * (A + B + XJ - ONE) * GX / (A + XJ)
      Q = Q * C / XJ
      SUMQ = SUMQ - Q
      AX = TEMP * Q
      BETANC = BETANC + AX
C
C     Check for convergence and act accordingly...
C
      ERRBD = (TEMP - GX) * SUMQ
      IF ((INT(XJ) .LT. ITRMAX) .AND. (ERRBD .GT. ERRMAX)) GO TO 10
      IF (ERRBD .GT. ERRMAX) IFAULT = 1
C
      RETURN
      END
```

```
      SUBROUTINE INVMOD(MAT,IMAT,RMOD,CMOD,IWK,NROW,IFAULT)
C
C        ALGORITHM AS 314.1 APPL. STATIST. (1997), VOL.46, NO.2
C
C        Inverts matrix with contents subject to modulo arithmetic
C
      INTEGER   IFAULT,NROW
      INTEGER   CMOD(NROW),IMAT(NROW*NROW),IWK(2*NROW),
     *          MAT(NROW*NROW),RMOD(NROW)
C
      INTEGER   I,IR,J,K,KIR,KJR,N
C
      EXTERNAL  MSORT,MUSORT
C
      INTRINSIC MOD
C
C        Check elements in 'mixed-moduli' positions are all zero
C
      N=0
      DO 26 I=1,NROW
        DO 24 J=1,NROW
          N=N+1
          IF ((RMOD(I).NE.CMOD(J)).AND.(MAT(N).GT.0)) GO TO 1002
          IF ((MAT(N).GT.RMOD(I)).OR.(MAT(N).LT.0)) GO TO 1001
          IMAT(N)=0
   24   CONTINUE
   26 CONTINUE
C
C        Sort rows and columns into ascending order of moduli
C
      CALL MSORT(MAT,IMAT,RMOD,CMOD,IWK,IWK(NROW+1),NROW)
C
C        Complete initialisation of inverse matrix
C
      DO 28 N=1,NROW*NROW,NROW+1
        IMAT(N)=1
   28 CONTINUE
C
C        Invert the matrix
C
      DO 190 IR=1,NROW
        KIR=(IR-1)*NROW
        IF (MAT(KIR+IR).EQ.0) THEN
C
C         Find a row JR below IR such that K(JR,IR)>0
C
          DO 112 KJR=KIR+NROW+IR,NROW*NROW,NROW
            IF (MAT(KJR).GT.0) GO TO 115
  112     CONTINUE
C
C         Column IR contains all zeros in rows IR or below:
C         look for a row above with zeros to left of column IR
C         and K(JR,IR)>0
C
          DO 114 KJR=IR,KIR,NROW
            IF (MAT(KJR).GT.0) THEN
              DO 113 I=KJR-IR+1,KJR-1
                IF (MAT(I).GT.0) GO TO 1003
  113         CONTINUE
              GO TO 115
            ENDIF
```

```
  114     CONTINUE
C
C        Column IR contains all zeros
C
         GO TO 190
C
C        Switch row JR with row IR
C
  115     KJR=KJR-IR
          DO 116 I=1,NROW
            K=MAT(KIR+I)
            MAT(KIR+I)=MAT(KJR+I)
            MAT(KJR+I)=K
            K=IMAT(KIR+I)
            IMAT(KIR+I)=IMAT(KJR+I)
            IMAT(KJR+I)=K
  116     CONTINUE
        END IF
C
C        Find multiplier N such that N*MAT(IR,IR)=1 mod(P{IR})
C
        K=MAT(KIR+IR)
        DO 122 N=1,RMOD(IR)-1
          IF (MOD(N*K,RMOD(IR)).EQ.1) GO TO 125
  122   CONTINUE
C
C        Multiply row IR by N
C
  125   IF (N.GT.1) THEN
          DO 126 I=KIR+1,IR*NROW
            MAT(I)=MAT(I)*N
            IMAT(I)=IMAT(I)*N
  126     CONTINUE
        END IF
C
C        Subtract MAT(JR,IR) * row IR from each row JR
C
        DO 136 KJR=0,NROW*NROW-1,NROW
          N=RMOD(IR)-MAT(KJR+IR)
          IF ((KJR.NE.KIR).AND.(N.NE.0)) THEN
            DO 132 I=1,NROW
              MAT(KJR+I)=MOD(MAT(KJR+I)+N*MAT(KIR+I),CMOD(I))
              IMAT(KJR+I)=MOD(IMAT(KJR+I)+N*IMAT(KIR+I),CMOD(I))
  132       CONTINUE
          END IF
  136   CONTINUE
  190 CONTINUE
C
C        Check inversion was possible - that result has
C        non-zero elements only on diagonal
C
      IFAULT=0
      DO 202 N=1,NROW*NROW,NROW+1
C
C        Zero diagonal element => left inverse formed
C
        IF (MAT(N).EQ.0) IFAULT=-1
        MAT(N)=-MAT(N)
  202 CONTINUE
      DO 204 N=1,NROW*NROW
        IF (MAT(N).GT.0) GO TO 1003
```

```
  204 CONTINUE
      DO 206 N=1,NROW*NROW,NROW+1
        MAT(N)=-MAT(N)
  206 CONTINUE
C
C        Sort rows and columns back into original order
C
      CALL MUSORT(MAT,IMAT,RMOD,CMOD,IWK,IWK(NROW+1),NROW)
C
      RETURN
C
C        Invalid entry (< 0 or > modulus)
C
 1001 IFAULT=1
      RETURN
C
C        Non-zero entry in position where row mod does not
C        equal column mod
C
 1002 IFAULT=2
      RETURN
C
C        Matrix cannot be inverted
C
 1003 IFAULT=3
C
      RETURN
      END
C
      SUBROUTINE MSORT(MAT,IMAT,RMOD,CMOD,RSORT,CSORT,NROW)
C
C        ALGORITHM AS 314.2 APPL. STATIST. (1997), VOL.46, NO.2
C
C        Sorts the matrix and associated information to put
C        rows and columns in ascending order of moduli
C
      INTEGER   NROW
      INTEGER   CMOD(NROW),CSORT(NROW),IMAT(*),MAT(*),RMOD(NROW),
     *          RSORT(NROW)
C
      INTEGER   I,IRC,J,JRC,KIRC,KJRC,P
C
C        Initialise row and column addresses
C
      DO 22 I=1,NROW
        RSORT(I)=I
        CSORT(I)=I
   22 CONTINUE
C
C        Sort rows
C
      DO 48 IRC=1,NROW
C
C        Find next row
C
        JRC=IRC
        P=RMOD(IRC)
        DO 42 I=IRC+1,NROW
          IF (RMOD(I).LT.P) THEN
            P=RMOD(I)
            JRC=I
```

```
            END IF
      42   CONTINUE
           IF (IRC.NE.JRC) THEN
             I=RMOD(IRC)
             RMOD(IRC)=RMOD(JRC)
             RMOD(JRC)=I
             I=RSORT(IRC)
             RSORT(IRC)=RSORT(JRC)
             RSORT(JRC)=I
C
C           Switch rows
C
             KIRC=(IRC-1)*NROW
             KJRC=(JRC-1)*NROW
             DO 44 J=1,NROW
               I=MAT(KIRC+J)
               MAT(KIRC+J)=MAT(KJRC+J)
               MAT(KJRC+J)=I
      44     CONTINUE
           END IF
      48 CONTINUE
C
C          Sort columns
C
         DO 68 IRC=1,NROW
C
C          Find next column
C
           JRC=IRC
           P=CMOD(IRC)
           DO 62 I=IRC+1,NROW
             IF (CMOD(I).LT.P) THEN
               P=CMOD(I)
               JRC=I
             END IF
      62   CONTINUE
           IF (IRC.NE.JRC) THEN
             I=CMOD(IRC)
             CMOD(IRC)=CMOD(JRC)
             CMOD(JRC)=I
             I=CSORT(IRC)
             CSORT(IRC)=CSORT(JRC)
             CSORT(JRC)=I
C
C           Switch columns
C
             DO 64 J=0,NROW*NROW-1,NROW
               I=MAT(IRC+J)
               MAT(IRC+J)=MAT(JRC+J)
               MAT(JRC+J)=I
      64     CONTINUE
           END IF
      68 CONTINUE
C
         RETURN
         END
C
         SUBROUTINE MUSORT(MAT,IMAT,RMOD,CMOD,RSORT,CSORT,NROW)
C
C          ALGORITHM AS 314.3 APPL. STATIST. (1997), VOL.46, NO.2
C
```

```
C        Sorts the inverse matrix and associated information into the
C        original order of rows and columns
C
      INTEGER   NROW
      INTEGER   CMOD(NROW),CSORT(NROW),IMAT(NROW*NROW),
     *          MAT(NROW*NROW),RMOD(NROW),RSORT(NROW)
C
      INTEGER   I,IRC,J,JRC,KIRC,KJRC
C
C        Sort rows of inverse (=columns of original)
C
      DO 48 IRC=1,NROW
C
C        Find next row
C
        IF (CSORT(IRC).NE.IRC) THEN
          DO 42 JRC=IRC+1,NROW
            IF (CSORT(JRC).EQ.IRC) GO TO 43
   42     CONTINUE
   43     I=CMOD(IRC)
          CMOD(IRC)=CMOD(JRC)
          CMOD(JRC)=I
          I=CSORT(IRC)
          CSORT(IRC)=CSORT(JRC)
          CSORT(JRC)=I
C
C        Switch rows
C
          KIRC=(IRC-1)*NROW
          KJRC=(JRC-1)*NROW
          DO 44 J=1,NROW
            I=IMAT(KIRC+J)
            IMAT(KIRC+J)=IMAT(KJRC+J)
            IMAT(KJRC+J)=I
   44     CONTINUE
        END IF
   48 CONTINUE
C
C        Sort columns of inverse (= rows of original)
C
      DO 68 IRC=1,NROW
C
C        Find next column
C
        IF (RSORT(IRC).NE.IRC) THEN
          DO 62 JRC=IRC+1,NROW
            IF (RSORT(JRC).EQ.IRC) GO TO 63
   62     CONTINUE
   63     I=RMOD(IRC)
          RMOD(IRC)=RMOD(JRC)
          RMOD(JRC)=I
          I=RSORT(IRC)
          RSORT(IRC)=RSORT(JRC)
          RSORT(JRC)=I
C
C        Switch columns
C
          DO 64 J=0,NROW*NROW-1,NROW
            I=IMAT(IRC+J)
            IMAT(IRC+J)=IMAT(JRC+J)
            IMAT(JRC+J)=I
```

```
   64     CONTINUE
C
C        Switch diagonal elements of MAT (others are zero)
C
          KIRC=(IRC-1)*NROW+IRC
          KJRC=(JRC-1)*NROW+JRC
          I=MAT(KIRC)
          MAT(KIRC)=MAT(KJRC)
          MAT(KJRC)=I
        END IF
   68 CONTINUE
C
      RETURN
      END
```

Below are 2 versions of this algorithm, the first as submitted
to the RSS in Fortran 77; the second is my Fortran 90 version.
The author has given permission for his version to be submitted
to the apstat collection.

As a `title' in the index I suggest:
Unconstrained variable metric function minimization without derivatives.

The two versions are separated by a line of !!!!!!!!'s

Alan
P.S. I have tested the F90 version. As it was derived from the F77
version, that should work too!

C Algorithm AS 319 and test program

```
C-----------------------------------------------------------------------
        PROGRAM  VAR
C-----------------------------------------------------------------------
C       A PROGRAM TO IMPLEMENT A QUASI-NEWTON METHOD.
C       USING NEW ALGORITHM VARMET   JULY 1994
C-----------------------------------------------------------------------
C
      COMMON /FUNERR/LER
      COMMON /TEST/IG,IFN
      EXTERNAL FUN
      LOGICAL LER
C
      INTEGER N, NMAX
      PARAMETER (N=2, NMAX=50)
      INTEGER IG,IFN,GRADTL,MESS,MAXFN,IER
      PARAMETER (GRADTL = 12, MAXFN = 1000, MESS = 6)
      DOUBLE PRECISION X,XTMP,W,FP
      DIMENSION X(N),XTMP(N),W(225)
C
      WRITE(*,*)'  '
      WRITE(*,*) 'INPUT YOUR STARTING GUESS'
      DO 12 I=1,N
        READ(*,*) XTMP(I)
   12 CONTINUE
      WRITE(*,*)' '
      WRITE(*,*)'INITIALIZATION COMPLETE.'
      WRITE(*,*)'****************************************'
C
        DO 24 J=1,N
  24      X(J)=XTMP(J)
        IFN = 0
        CALL VARME(FUN,N,X,FP,W,GRADTL,MAXFN,MESS,IER)
C
      WRITE(*,*)' '
      WRITE(*,*)' THE NUMBER OF FUNCTION EVALUATIONS IS ',IFN
      WRITE(*,*)' '
      WRITE(*,*)'THE MINIMUM FOUND IS',(X(I),I=1,N)
      WRITE(*,*)' '
      CALL FUN(N,X,FP)
      WRITE(*,*)'THE FUNCTION VALUE IS: ',FP
      STOP
      END
C-----------------------------------------------------------------------
```

```
      SUBROUTINE FUN(NORD,BP,Q)
C-----------------------------------------------------------------------
      COMMON /FUNERR/LER
      COMMON /TEST/IG,IFN
      DIMENSION BP(*)
      LOGICAL LER
      DOUBLE PRECISION BP,Q
      Q=100.*(BP(2)-BP(1)**2)**2+(BP(1)-1.)**2
      IFN = IFN + 1
      LER = .FALSE.
      RETURN
      END
C-----------------------------------------------------------------------
      SUBROUTINE VARME(FUN,NPAR,B,F0,W,NSIG,MAXFN,IOUT,IER)
C-----------------------------------------------------------------------
C
C     CALLING SUBROUTINE FOR SUBROUTINE VARMET
C
C     ALLOWS FOR SETUP OF DEFAULT PARAMETERS
C     AND EFFICIENT USE OF STORAGE
C     AS WELL AS WRITING OF ERROR MESSAGES
C
C    VERSION 0.1
C    CODED BY JOHN J. KOVAL
C    MARCH 1986
C
C    VERSION 0.2
C    CODED BY JOHN J. KOVAL
C    JULY 1988
C
C    VERSION 0.26
C    CODED BY MURRAY ALEXANDER FOR JOHN J. KOVAL
C    JULY 1989
C
C    VERSION 0.27
C    MODIFIED BY NAZIH HASSAN, JULY 1993
C
C    VERSION 0.28
C    MODIFIED BY JOHN KOVAL, JUNE 1996
C    BECAUSE OF COMMENTS FROM REVIEWER FOR APPLIED STATISTICS
C    CHANGES TO ORDER OF PARAMETERS IN GRAD
C
C    PARAMETERS              MEANING                        DEFAULT
C    ----------              -------                        -------
C
C     FUN         NAME OF FUNCTION TO BE MINIMIZED
C
C     NPAR        ORDER OF PARAMETER VECTOR
C                 (NUMBER OF UNKNOWNS)
C
C     B           ARRAY CONTAINING INITIAL ESTIMATES
C                 ON OUTPUT CONTAINING FINAL ESTIMATES
C
C     F0          VALUE OF FUNCTION AT MINIMUM
C
C     W           WORK ARRAY OF LENGTH FO (NPAR+5)*NPAR
C
C     NSIG        MACHINE ACCURACY AS NEGATIVE POWER       10 OR 5
C                 OF TEN
C
C     MAXFN       MAXIMUM NUMBER OF FUNCTION EVALUATIONS     1000
```

```
C                (DOES INCLUDE EVALUATIONS BY SUBROUTINE
C                GRAD WHICH CALCULATES APPROXIMATE GRADIENT)
C
C     IOUT       OUTPUT CHANNEL FOR ERROR MESSAGES           0
C                (IF 0, THEN MESSAGES NOT WRITTEN)
C
C      IER       ERROR INDICATOR                             0
C                INTEGER
C
C-----------------------------------------------------------------------
      DOUBLE PRECISION W,B,F0,EPD,GRADTL
      EXTERNAL FUN
      DIMENSION B(*),W(*)
      PARAMETER (MAXF = 1000, EPD = 1.0D-05, MSIG = 10)
C
C     INITIALIZE
C
      IER=0
C
      IF(NSIG.EQ.0) NSIG = MSIG
      GRADTL = 1.0/(10.0**(NSIG))
C
      IF(GRADTL.LT.0.0) THEN
        IF(IOUT.GT.0) WRITE(IOUT,300) NSIG, GRADTL
 300    FORMAT(' NSIG VALUE OF ',I3,' CREATES NEGATIVE VALUE OF',
     1' GRADTL, NAMELY, ',G12.5)
        GRADTL = 1.0/(10**(MSIG))
        IF(IOUT.GT.0) WRITE(IOUT,310) MSIG, GRADTL
 310    FORMAT(' PROGRAM SUBSTITUTES NSIG VALUE OF ',I3,' WHICH',
     1' GIVES GRADTL VALUE OF ',G12.5)
      ENDIF
C
      IF(MAXFN.EQ.0) MAXFN = MAXF
C
C      NOW WE ARE READY TO CALL THE MINIMIZATION SUBROUTINE
C
      I1 = NPAR*NPAR + 1
      I2 = I1 + NPAR
      I3 = I2 + NPAR
      I4 = I3 + NPAR
C
      CALL VARMET(FUN,NPAR,B,F0,W(I3),W,W(I1),W(I2),W(I4),
     1 GRADTL,MAXFN,IER)
C
      IF(IER.GT.0.AND.IOUT.GT.0)THEN
        WRITE(IOUT,30) IER
   30   FORMAT(/' SUBROUTINE VARMET ERROR NUMBER ',I3)
        IF(IER.EQ.1) THEN
          WRITE(IOUT,40)
        ELSE IF(IER.EQ.2) THEN
          WRITE(IOUT,60)
        ELSE IF(IER.EQ.3) THEN
          WRITE(IOUT,70)
        ELSE IF(IER.EQ.4) THEN
          WRITE(IOUT,80)
        ENDIF
   40   FORMAT(' FUNCTION UNDEFINED AT INITIAL VALUE         ')
   60   FORMAT(' GRADIENT UNDEFINED IN TOO MANY DIMENSIONS   ')
   70   FORMAT(' FUNCTON NOT MINIMIZED BUT'
     1  /' UNABLE TO FIND MINIMUM IN DIRECTION OF SEARCH')
   80   FORMAT(' TOO MANY FUNCTION EVALUATIONS REQUIRED      ')
```

```
C
      ENDIF
C
  200 RETURN
      END

C-----------------------------------------------------------------------
      SUBROUTINE VARMET(FUN,NPAR,B,F0,G,H,C,D,T,GRADTL,MAXFN,IFAULT)
C
C       ALGORITHM AS 319 APPL.STATIST. (1997), VOL.46, NO.4
C
C               VARIABLE METRIC FUNCTION MINIMISATION
C
      EXTERNAL FUN
      COMMON/FUNERR/LER
      DIMENSION B(NPAR),G(NPAR),H(NPAR,NPAR),C(NPAR),D(NPAR),T(2*NPAR)
      DOUBLE PRECISION B,F0,G,H,C,D,T,GRADTL,W,TOLER,D1,S,CK,F1,D2
      INTEGER IFN,IG
      LOGICAL LER
      PARAMETER (ICMAX=20, TOLER=0.00001, W=0.2)
C
      IG = 0
      IFN = 0
      LER = .FALSE.
      IFAULT = 0
      NP = NPAR + 1
C
      IF (MAXFN.EQ.0) MAXFN = 1000
      IF (GRADTL.EQ.0.0) GRADTL = 1.0D-10
C
      CALL FUN(NPAR,B,F0)
      IF(LER) THEN
         IFAULT = 1
         RETURN
      ENDIF
      IFN = IFN + 1
C
      CALL GRAD(FUN,NPAR,B,F0,G,T(NP),GRADTL,IFAULT)
      IF(IFAULT.GT.0) RETURN
C
      IG = IG + 1
      IFN = IFN + NPAR
      IF(IFN.GT.MAXFN) THEN
         IFAULT = 4
         RETURN
      ENDIF
C
   10 DO 30 K = 1,NPAR
         DO 20 L = 1,NPAR
            H(K,L) = 0.0
   20    CONTINUE
         H(K,K) = 1.00
   30 CONTINUE
      ILAST = IG
C
   40 DO 50 I = 1,NPAR
         D(I) = B(I)
         C(I) = G(I)
   50 CONTINUE
C
      D1 = 0.0
```

```
      DO 70 I = 1,NPAR
         S = 0.0
         DO 60 J = 1,NPAR
            S = S - H(I,J)*G(J)
   60    CONTINUE
         T(I) = S
         D1 = D1 - S*G(I)
   70 CONTINUE
C
      IF(D1.LE.0.0) THEN
         IF(ILAST.EQ.IG) THEN
            RETURN
         ENDIF
         GO TO 10
      ELSE
         CK = 1.0
         IC = 0
   90    ICOUNT = 0
         DO 100 I = 1,NPAR
            B(I) = D(I) + CK*T(I)
            IF(B(I).EQ.D(I)) THEN
               ICOUNT = ICOUNT + 1
            ENDIF
  100    CONTINUE
C
         IF(ICOUNT.GE.NPAR) THEN
            IF(ILAST.EQ.IG) THEN
               RETURN
            ENDIF
            GO TO 10
         ELSE
            CALL FUN(NPAR,B,F1)
C
            IFN = IFN + 1
            IF(IFN.GT.MAXFN) THEN
               IFAULT = 4
               RETURN
            ELSE IF(LER) THEN
               CK = W * CK
               IC = IC+1
               IF(IC.GT.ICMAX) THEN
                  IFAULT = 3
                  RETURN
               ENDIF
               GO TO 90
C
            ELSE IF(F1.GE.F0 - D1*CK*TOLER) THEN
               CK = W * CK
               GO TO 90
            ELSE
               F0 = F1
               CALL GRAD(FUN,NPAR,B,F0,G,T(NP),GRADTL,IFAULT)
               IF(IFAULT.GT.0) THEN
                  RETURN
               ENDIF
               IG = IG + 1
               IFN = IFN + NPAR
               IF(IFN.GT.MAXFN) THEN
                  IFAULT = 4
                  RETURN
               ENDIF
```

```
C
               D1 = 0.0
               DO 130 I = 1,NPAR
                  T(I) = CK*T(I)
                  C(I) = G(I) - C(I)
                  D1 = D1 + T(I)*C(I)
  130          CONTINUE
C
               IF(D1.LE.0.0) THEN
                  GOTO 10
               ENDIF
C
               D2 = 0.0
               DO 150 I = 1,NPAR
                  S = 0.0
                  DO 140 J = 1,NPAR
                     S = S + H(I,J)*C(J)
  140             CONTINUE
                  D(I) = S
                  D2 = D2 + S*C(I)
  150          CONTINUE
               D2 = 1.0 + D2/D1
C
               DO 170 I = 1,NPAR
                  DO 170 J = 1,NPAR
                     H(I,J) = H(I,J) - (T(I)*D(J) + D(I)*T(J) -
     1               D2*T(I)*T(J))/D1
  170          CONTINUE
            ENDIF
         ENDIF
      ENDIF
      GO TO 40
      END

      SUBROUTINE GRAD(F,NPAR,B,F0,G,SA,ER,IFAULT)
C
C     CALCULATE APPROXIMATE GRADIENT
C
      DIMENSION B(NPAR),G(NPAR),SA(NPAR)
      DOUBLE PRECISION B,F0,G,SA,ER,H,F1
      COMMON/FUNERR/LER
      LOGICAL LER
C
      JCMAX=NPAR-2
      JC = 0
C
      DO 20 I = 1,NPAR
         H =(DABS(B(I)) +DSQRT(ER)) *DSQRT(ER)
         SA(I) = B(I)
         B(I) = B(I) + H
         CALL F(NPAR,B,F1)
         B(I) = SA(I)
C
         IF(LER) THEN
            F1 = F0 + H
            JC = JC + 1
         ENDIF
C
         G(I) = (F1 -F0)/H
   20 CONTINUE
C
```

```
      IF(JC.GT.JCMAX) IFAULT = 2
      RETURN
      END

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Algorithm AS 319 and test program
! Converted to Fortran 90 free-format style by Alan Miller
! e-mail: Alan.Miller @ vic.cmis.csiro.au
! URL: www.ozemail.com.au/~milleraj

MODULE as319
IMPLICIT NONE

! COMMON /funerr/ler
! COMMON /test/ig,ifn
LOGICAL, SAVE      :: ler
INTEGER, SAVE      :: ig, ifn

INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(14, 50)

END MODULE as319



!-----------------------------------------------------------------------

SUBROUTINE varme(fun, npar, b, f0, nsig, maxfn, iout, ier)
!-----------------------------------------------------------------------

!      CALLING SUBROUTINE FOR SUBROUTINE VARMET

!      ALLOWS FOR SETUP OF DEFAULT PARAMETERS
!      AND EFFICIENT USE OF STORAGE
!      AS WELL AS WRITING OF ERROR MESSAGES

!     VERSION 0.1
!     CODED BY JOHN J. KOVAL
!     MARCH 1986

!     VERSION 0.2
!     CODED BY JOHN J. KOVAL
!     JULY 1988

!     VERSION 0.26
!     CODED BY MURRAY ALEXANDER FOR JOHN J. KOVAL
!     JULY 1989

!     VERSION 0.27
!     MODIFIED BY NAZIH HASSAN, JULY 1993

!     VERSION 0.28
!     MODIFIED BY JOHN KOVAL, JUNE 1996
!     BECAUSE OF COMMENTS FROM REVIEWER FOR APPLIED STATISTICS
!     CHANGES TO ORDER OF PARAMETERS IN GRAD

!     PARAMETERS             MEANING                       DEFAULT
!     ----------             -------                       -------

!      FUN          NAME OF FUNCTION TO BE MINIMIZED
```

```
!     NPAR        ORDER OF PARAMETER VECTOR
!                 (NUMBER OF UNKNOWNS)

!     B           ARRAY CONTAINING INITIAL ESTIMATES
!                 ON OUTPUT CONTAINING FINAL ESTIMATES

!     F0          VALUE OF FUNCTION AT MINIMUM

!     NSIG        MACHINE ACCURACY AS NEGATIVE POWER       10 OR 5
!                 OF TEN

!     MAXFN       MAXIMUM NUMBER OF FUNCTION EVALUATIONS     1000
!                 (DOES INCLUDE EVALUATIONS BY SUBROUTINE
!                 GRAD WHICH CALCULATES APPROXIMATE GRADIENT)

!     IOUT        OUTPUT CHANNEL FOR ERROR MESSAGES            0
!                 (IF 0, THEN MESSAGES NOT WRITTEN)

!      IER        ERROR INDICATOR                              0
!                 INTEGER

!-----------------------------------------------------------------------

USE as319
IMPLICIT NONE
INTEGER, INTENT(IN)               :: npar
REAL (dp), INTENT(IN OUT)         :: b(:)
REAL (dp), INTENT(IN OUT)         :: f0
INTEGER, INTENT(IN OUT)           :: nsig
INTEGER, INTENT(IN OUT)           :: maxfn
INTEGER, INTENT(IN OUT)           :: iout
INTEGER, INTENT(OUT)              :: ier

INTERFACE
  SUBROUTINE fun(nord, bp, q)
    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)           :: nord
    REAL (dp), INTENT(IN)         :: bp(:)
    REAL (dp), INTENT(OUT)        :: q
  END SUBROUTINE fun
END INTERFACE

REAL (dp) :: gradtl

INTEGER, PARAMETER :: maxf = 1000, msig = 10

!     INITIALIZE

ier=0

IF(nsig == 0) nsig = msig
gradtl = 1.0 / (10.0**(nsig))

IF(gradtl < 0.0) THEN
  IF(iout > 0) WRITE(iout, 300) nsig, gradtl
  300    FORMAT(' NSIG VALUE OF ', i3, ' CREATES NEGATIVE VALUE OF',  &
      ' GRADTL, NAMELY, ', g12.5)
  gradtl = 1.0/(10**(msig))
  IF(iout > 0) WRITE(iout, 310) msig, gradtl
  310    FORMAT(' PROGRAM SUBSTITUTES NSIG VALUE OF ', i3, ' WHICH',  &
```

```
      ' GIVES GRADTL VALUE OF ', g12.5)
END IF

IF(maxfn == 0) maxfn = maxf

!      NOW WE ARE READY TO CALL THE MINIMIZATION SUBROUTINE

CALL varmet(fun, npar, b, f0, gradtl, maxfn, ier)

IF(ier > 0.AND.iout > 0)THEN
  WRITE(iout, 30) ier
  30   FORMAT(/' SUBROUTINE VARMET ERROR NUMBER ', i3)
  IF(ier == 1) THEN
    WRITE(iout, 40)
  ELSE IF(ier == 2) THEN
    WRITE(iout, 60)
  ELSE IF(ier == 3) THEN
    WRITE(iout, 70)
  ELSE IF(ier == 4) THEN
    WRITE(iout, 80)
  END IF
  40   FORMAT(' FUNCTION UNDEFINED AT INITIAL VALUE         ')
  60   FORMAT(' GRADIENT UNDEFINED IN TOO MANY DIMENSIONS   ')
  70   FORMAT(' FUNCTON NOT MINIMIZED BUT'  &
      /' UNABLE TO FIND MINIMUM IN DIRECTION OF SEARCH')
  80   FORMAT(' TOO MANY FUNCTION EVALUATIONS REQUIRED      ')

END IF

RETURN

CONTAINS


!------------------------------------------------------------------------

SUBROUTINE varmet(fun, npar, b, f0, gradtl, maxfn, ifault)

!       ALGORITHM AS 319 APPL.STATIST. (1997), VOL.46, NO.4

!               VARIABLE METRIC FUNCTION MINIMISATION

INTEGER, INTENT(IN)               :: npar
REAL (dp), INTENT(IN OUT)         :: b(:)
REAL (dp), INTENT(OUT)            :: f0
REAL (dp), INTENT(OUT)            :: gradtl
INTEGER, INTENT(OUT)              :: maxfn
INTEGER, INTENT(OUT)              :: ifault

INTERFACE
  SUBROUTINE fun(nord, bp, q)
    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)           :: nord
    REAL (dp), INTENT(IN)         :: bp(:)
    REAL (dp), INTENT(OUT)        :: q
  END SUBROUTINE fun
END INTERFACE

REAL (dp)            :: d1, s, ck, f1, d2
INTEGER              :: i, ic, icount, ifn, ig, ilast, j, k, np
```

```
  INTEGER, PARAMETER   :: icmax=20
  REAL (dp), PARAMETER :: toler=0.00001, w=0.2
  REAL (dp)            :: g(npar), h(npar,npar), c(npar), d(npar), t(2*npar)

  ig = 0
  ifn = 0
  ler = .false.
  ifault = 0
  np = npar + 1

  IF (maxfn == 0) maxfn = 1000
  IF (gradtl == 0.0) gradtl = 1.0D-10

  CALL fun(npar, b, f0)
  IF(ler) THEN
    ifault = 1
    RETURN
  END IF
  ifn = ifn + 1

  CALL grad(fun, npar, b, f0, g, t(np:), gradtl, ifault)
  IF(ifault > 0) RETURN

  ig = ig + 1
  ifn = ifn + npar
  IF(ifn > maxfn) THEN
    ifault = 4
    RETURN
  END IF

  10 DO k = 1, npar
    h(k,1:npar) = 0.0
    h(k,k) = 1.00
  END DO
  ilast = ig

  40 DO i = 1, npar
    d(i) = b(i)
    c(i) = g(i)
  END DO

  d1 = 0.0
  DO i = 1, npar
    s = - DOT_PRODUCT( h(i,1:npar), g(1:npar) )
    t(i) = s
    d1 = d1 - s*g(i)
  END DO

  IF(d1 <= 0.0) THEN
    IF(ilast == ig) THEN
      RETURN
    END IF
    GO TO 10
  ELSE
    ck = 1.0
    ic = 0
    90    icount = 0
    DO i = 1,npar
      b(i) = d(i) + ck*t(i)
      IF(b(i) == d(i)) THEN
        icount = icount + 1
```

```
      END IF
    END DO

    IF(icount >= npar) THEN
      IF(ilast == ig) THEN
        RETURN
      END IF
      GO TO 10
    ELSE
      CALL fun(npar, b, f1)

      ifn = ifn + 1
      IF(ifn > maxfn) THEN
        ifault = 4
        RETURN
      ELSE IF(ler) THEN
        ck = w * ck
        ic = ic+1
        IF(ic > icmax) THEN
          ifault = 3
          RETURN
        END IF
        GO TO 90

      ELSE IF(f1 >= f0 - d1*ck*toler) THEN
        ck = w * ck
        GO TO 90
      ELSE
        f0 = f1
        CALL grad(fun, npar, b, f0, g, t(np:), gradtl, ifault)
        IF(ifault > 0) THEN
          RETURN
        END IF
        ig = ig + 1
        ifn = ifn + npar
        IF(ifn > maxfn) THEN
          ifault = 4
          RETURN
        END IF

        d1 = 0.0
        DO i = 1, npar
          t(i) = ck*t(i)
          c(i) = g(i) - c(i)
          d1 = d1 + t(i)*c(i)
        END DO

        IF(d1 <= 0.0) THEN
          GO TO 10
        END IF

        d2 = 0.0
        DO i = 1, npar
          s = 0.0
          DO j = 1, npar
            s = s + h(i,j)*c(j)
          END DO
          d(i) = s
          d2 = d2 + s*c(i)
        END DO
        d2 = 1.0 + d2/d1
```

```
      DO i = 1, npar
        DO j = 1, npar
          h(i,j) = h(i,j) - (t(i)*d(j) + d(i)*t(j) - d2*t(i)*t(j))/d1
        END DO
      END DO
    END IF
  END IF
END IF
GO TO 40
END SUBROUTINE varmet


SUBROUTINE grad(f, npar, b, f0, g, sa, er, ifault)

!     CALCULATE APPROXIMATE GRADIENT

INTEGER, INTENT(IN)           :: npar
REAL (dp), INTENT(IN OUT)     :: b(:)
REAL (dp), INTENT(IN)         :: f0
REAL (dp), INTENT(OUT)        :: g(:)
REAL (dp), INTENT(OUT)        :: sa(:)
REAL (dp), INTENT(IN)         :: er
INTEGER, INTENT(OUT)          :: ifault

INTERFACE
  SUBROUTINE f(nord, bp, q)
    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)           :: nord
    REAL (dp), INTENT(IN)         :: bp(:)
    REAL (dp), INTENT(OUT)        :: q
  END SUBROUTINE f
END INTERFACE

REAL (dp) :: h, f1
INTEGER   :: i, jc, jcmax

jcmax=npar - 2
jc = 0

DO i = 1, npar
  h =(ABS(b(i)) + SQRT(er)) * SQRT(er)
  sa(i) = b(i)
  b(i) = b(i) + h
  CALL f(npar, b, f1)
  b(i) = sa(i)

  IF(ler) THEN
    f1 = f0 + h
    jc = jc + 1
  END IF

  g(i) = (f1 - f0)/h
END DO

IF(jc > jcmax) ifault = 2
RETURN
END SUBROUTINE grad

END SUBROUTINE varme
```

```
!-----------------------------------------------------------------------
PROGRAM  var
!-----------------------------------------------------------------------
!       A PROGRAM TO IMPLEMENT A QUASI-NEWTON METHOD.
!       USING NEW ALGORITHM VARMET   JULY 1994
!-----------------------------------------------------------------------

USE as319
IMPLICIT NONE
INTEGER, PARAMETER :: n=2
INTEGER            :: ier
INTEGER, SAVE      :: gradtl = 12, maxfn = 1000, mess = 6
REAL (dp)          :: x(n), xtmp(n), fp

INTERFACE
  SUBROUTINE fun(nord, bp, q)
    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)           :: nord
    REAL (dp), INTENT(IN)         :: bp(:)
    REAL (dp), INTENT(OUT)        :: q
  END SUBROUTINE fun

  SUBROUTINE varme(fun, npar, b, f0, nsig, maxfn, iout, ier)
    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)              :: npar
    REAL (dp), INTENT(IN OUT)        :: b(:)
    REAL (dp), INTENT(IN OUT)        :: f0
    INTEGER, INTENT(IN OUT)          :: nsig
    INTEGER, INTENT(IN OUT)          :: maxfn
    INTEGER, INTENT(IN OUT)          :: iout
    INTEGER, INTENT(OUT)             :: ier
    INTERFACE
      SUBROUTINE fun(nord, bp, q)
        USE as319
        IMPLICIT NONE
        INTEGER, INTENT(IN)          :: nord
        REAL (dp), INTENT(IN)        :: bp(:)
        REAL (dp), INTENT(OUT)       :: q
      END SUBROUTINE fun
    END INTERFACE
  END SUBROUTINE varme
END INTERFACE

WRITE(*,*)'  '
WRITE(*,*) 'INPUT YOUR STARTING GUESS: '
READ(*,*) xtmp(1:n)
WRITE(*,*)' '
WRITE(*,*)'INITIALIZATION COMPLETE.'
WRITE(*,*)'****************************************'

x(1:n)=xtmp(1:n)
ifn = 0
CALL varme(fun, n, x, fp, gradtl, maxfn, mess, ier)

WRITE(*,*)' '
IF (ier /= 0) WRITE(*, *) '** IER =', ier, ' **'
```

```
    WRITE(*,*)' THE NUMBER OF FUNCTION EVALUATIONS IS ', ifn
    WRITE(*,*)' '
    WRITE(*,*)'THE MINIMUM FOUND IS ', x(1:n)
    WRITE(*,*)' '
    CALL fun(n, x, fp)
    WRITE(*, *)'THE FUNCTION VALUE IS: ', fp
    STOP
    END PROGRAM  var
    !------------------------------------------------------------------------

    SUBROUTINE fun(nord, bp, q)
    !------------------------------------------------------------------------

    USE as319
    IMPLICIT NONE
    INTEGER, INTENT(IN)           :: nord
    REAL (dp), INTENT(IN)         :: bp(:)
    REAL (dp), INTENT(OUT)        :: q

    q=100.*(bp(2) - bp(1)**2)**2 + (bp(1) - 1.)**2
    ifn = ifn + 1
    ler = .false.
    IF (nord < 1) WRITE(*, *)'** NORD must be > 0, actual value =', nord
    RETURN
    END SUBROUTINE fun
```

Pantelis Vlachos

Pantelis Vlachos (vlachos@stat.cmu.edu) is a Research Scientist, in the Department of Statistics, at Carnegie Mellon University. He took over the StatLib service in June of 1998 from Mike Meyer.

Mike started the StatLib, service in April of 1989. By 1995 StatLib had grown to a collection of about 150 Mbytes and the StatLib server was servicing about 60,000 transactions per month.

The people who really make StatLib a success are those who contribute software, datasets, and other information. Thanks to those who have already contributed, and keep those contributions coming.

Back to StatLib