محاسبات آماری پیشرفته

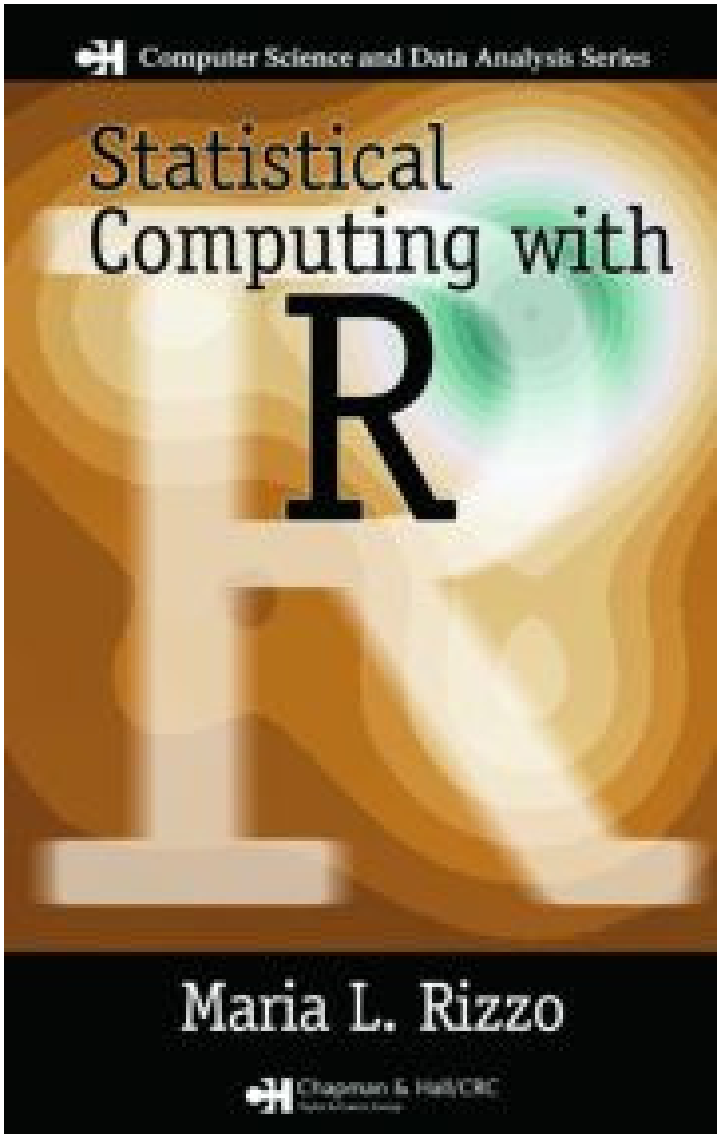
ترم اول سال تحصیلی ۹۳

جلسه اول

حسین باغیشنی

دانشگاه شاهرود

منبع


Computer Science and Data Analysis Series
Statistical Computing with R
Maria L. Rizzo
Chapman & Hall/CRC

محاسبات آماری یا آمار محاسباتی؟

محاسبات آماری یا آمار محاسباتی؟

محاسبات آماری یا آمار محاسباتی؟

آمار محاسباتی به مباحث آماری روش‌های مختلف استنباطی می‌پردازد: به عنوان مثال سازگاری برآوردگرهای به دست آمده در یک مدل خاص به وسیله روش‌های محاسباتی مختلف، یا نرخ همگرایی برآوردگرهای حاصل از روش MCMC.

محاسبات آماری یا آمار محاسباتی؟

محاسبات آماری یا آمار محاسباتی؟

آمار محاسباتی به مباحث آماری روش‌های مختلف استنباطی می‌پردازد: به عنوان مثال سازگاری برآوردگرهای به دست آمده در یک مدل خاص به وسیله روش‌های محاسباتی مختلف، یا نرخ همگرایی برآوردگرهای حاصل از روش MCMC.

محاسبات آماری که مورد نظر ما و این کلاس است، به روش‌های مختلف محاسبه کمیت‌های مورد علاقه در استنباط‌های آماری مربوط می‌شود.

محاسبات آماری یا آمار محاسباتی؟

محاسبات آماری یا آمار محاسباتی؟

آمار محاسباتی به مباحث آماری روش‌های مختلف استنباطی می‌پردازد: به عنوان مثال سازگاری برآوردگرهای به دست آمده در یک مدل خاص به وسیله روش‌های محاسباتی مختلف، یا نرخ همگرایی برآوردگرهای حاصل از روش MCMC.

محاسبات آماری که مورد نظر ما و این کلاس است، به روش‌های مختلف محاسبه کمیت‌های مورد علاقه در استنباط‌های آماری مربوط می‌شود.

البته بعضی از آماردان‌ها معتقدند این هر دو عنوان، و با جامعیت آمار محاسباتی، یکی هستند

چرا آماردان‌های خوب باید برنامه‌نویسی بلد باشند؟

- **استقلال**: در غیر این صورت باید منتظر باشید تا شخص دیگری برنامه‌های مورد نیاز شما را بنویسد و به شما تحویل دهد

چرا آماردان‌های خوب باید برنامه‌نویسی بلد باشند؟

- **استقلال**: در غیر این صورت باید منتظر باشید تا شخص دیگری برنامه‌های مورد نیاز شما را بنویسد و به شما تحویل دهد
- **صداقت**: در غیر این صورت ممکن است نتایج را طوری که مایلید اتفاق بیافتند، دستکاری کنید

چرا آماردان‌های خوب باید برنامه‌نویسی بلد باشند؟

- **استقلال**: در غیر این صورت باید منتظر باشید تا شخص دیگری برنامه‌های مورد نیاز شما را بنویسد و به شما تحویل دهد
- **صداقت**: در غیر این صورت ممکن است نتایج را طوری که مایلید اتفاق بیافتند، دستکاری کنید
- **وضوح و روشنی**: برنامه‌نویسی باعث می‌شود بتوانید فکر خود را منظم کرده و آن را با بقیه به اشتراک بگذارید

چرا آماردان‌های خوب باید برنامه‌نویسی بلد باشند؟

- **استقلال**: در غیر این صورت باید منتظر باشید تا شخص دیگری برنامه‌های مورد نیاز شما را بنویسد و به شما تحویل دهد
- **صداقت**: در غیر این صورت ممکن است نتایج را طوری که مایلید اتفاق بیافتند، دستکاری کنید
- **وضوح و روشنی**: برنامه‌نویسی باعث می‌شود بتوانید فکر خود را منظم کرده و آن را با بقیه به اشتراک بگذارید

و یادتان باشد که علم باید عمومی باشد

برنامه‌نویسی: تبدیل ورودی به خروجی

- برنامه‌نویسی، نوشتن توابعی برای تبدیل ورودی‌ها به خروجی‌هاست

برنامه‌نویسی: تبدیل ورودی به خروجی

- برنامه‌نویسی، نوشتن توابعی برای تبدیل ورودی‌ها به خروجی‌هاست
- برنامه‌نویسی خوب به معنی انجام تبدیل بیان‌شده به سادگی و درستی است

برنامه‌نویسی: تبدیل ورودی به خروجی

- برنامه‌نویسی، نوشتن توابعی برای تبدیل ورودی‌ها به خروجی‌هاست
- برنامه‌نویسی خوب به معنی انجام تبدیل بیان‌شده به سادگی و درستی است
- ماشین‌ها از ماشین‌ها ساخته می‌شوند؛ توابع نیز از توابع ساخته می‌شوند: مانند $f(a, b) = a^۲ + b^۲$

برنامه‌نویسی: تبدیل ورودی به خروجی

- برنامه‌نویسی، نوشتن توابعی برای تبدیل ورودی‌ها به خروجی‌هاست
- برنامه‌نویسی خوب به معنی انجام تبدیل بیان‌شده به سادگی و درستی است
- ماشین‌ها از ماشین‌ها ساخته می‌شوند؛ توابع نیز از توابع ساخته می‌شوند: مانند $f(a, b) = a^۲ + b^۲$
- مسیر اجرای یک برنامه‌نویسی خوب، در نظر گرفتن یک تبدیل بزرگ و شکستن آن به قطعات کوچکتر و سپس شکستن دوباره قطعات کوچکتر است مادامی که توابع ساخته‌شده وظیفه خود را به درستی و با سادگی انجام دهند

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

مقادیر دودویی مستقیم: بولی ($TRUE/FALSE$ در R).

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

مقادیر دودویی مستقیم: بولی ($TRUE/FALSE$ در R).

اعداد صحیح

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

مقادیر دودویی مستقیم: بولی ($TRUE/FALSE$ در R).

اعداد صحیح

کاراکترها

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

مقادیر دودویی مستقیم: بولی ($TRUE/FALSE$ در R).

اعداد صحیح

کاراکترها

مقادیر گم‌شده یا بدتعریف شده: NA، NaN

شناخت داده‌ها قبل از توابع

انواع مختلف داده‌ها

تمامی داده‌ها با ساختار دودویی، توسط بیت‌ها ($TRUE/FALSE$ یا YES/NO یا ۱/۰) نمایش داده می‌شوند.

مقادیر دودویی مستقیم: بولی ($TRUE/FALSE$ در R).

اعداد صحیح

کاراکترها

مقادیر گم‌شده یا بدتعریف شده: NA، NaN

اعداد با ممیز شناور: مانند اعداد کسری. در نظر گرفتن تعداد بیت‌های بیشتر برای قسمت کسری، دقت بیشتر را نتیجه می‌دهد. برای اعداد با ممیز شناور، این تعداد دو برابر حالت استاندارد، مثلا $numeric$، است.

```
> 0.45 == 3*0.15
[1] FALSE
```

```
> 0.45 == 3*0.15
[1] FALSE
```

اغلب (نه همیشه) این اختلاف قابل صرفنظر کردن است. گرد کردن خطا در محاسبات حجیم، منجر به تجمع و بزرگی خطا میشود.

```
> 0.45 == 3*0.15
[1] FALSE
```

اغلب (نه همیشه) این اختلاف قابل صرفنظر کردن است. گرد کردن خطا در محاسبات حجیم، منجر به تجمع و بزرگی خطا میشود.

به ویژه این مساله زمانی که نتایج باید نزدیک به صفر باشند، می‌تواند خیلی مشکل‌زا باشد.
زیرا ممکن است علامت به اشتباه عوض شود

```
> 0.45-3*0.15
[1] 5.551115e-17
```

عملگرها: جمع، ضرب و ...

عملگرها: جمع، ضرب و ...

مقایسه‌ها، عملگرهای دودویی هستند

```
> 7 > 5
[1] TRUE
> 7 < 5
[1] FALSE
> 7 >= 7
[1] TRUE
> 7 <= 5
[1] FALSE
> 7 == 5
[1] FALSE
> 7 != 5
[1] TRUE
```

دقت متناهی با اعداد با ممیز شناور و مقایسه‌های دقیق، گاهی نتایجی عجیبی پدید می‌آید. دستور $all.equal()$ معمولا چنین مشکلاتی را مرتفع می‌کند.

```
(0.5-0.3) == (0.3-0.1)
[1] FALSE
> all.equal(0.5-0.3,0.3-0.1)
[1] TRUE
```

دقت متناهی با اعداد با ممیز شناور و مقایسه‌های دقیق، گاهی نتایجی عجیبی پدید می‌آید. دستور $all.equal()$ معمولا چنین مشکلاتی را مرتفع می‌کند.

```
(0.5-0.3) == (0.3-0.1)
[1] FALSE
> all.equal(0.5-0.3,0.3-0.1)
[1] TRUE
```

عملگرهای بولی برای and و or:

```
> (5 > 7) & (6*7 == 42)
[1] FALSE
> (5 > 7) | (6*7 == 42)
[1] TRUE
```

تابع $typeof$ نوع شی را نشان می‌دهد

توابع $is.foo$ یک مقدار بولی در مورد این که آرگومان تابع از نوع foo هست یا خیر نشان می‌دهند:

تابع $typeof$ نوع شی را نشان می‌دهد
توابع $is.foo$ یک مقدار بولی در مورد این که آرگومان تابع از نوع foo هست یا خیر نشان می‌دهند:

```
> typeof(7)
[1] "double"
> is.numeric(7)
[1] TRUE
> is.integer(7)
[1] FALSE
> is.character(7)
[1] FALSE
> is.character("7")
[1] TRUE
> is.character("seven")
[1] TRUE
> is.na("seven")
[1] FALSE
```

تابع $as.foo$:

```
> as.character(5/6)
[1] "0.833333333333333"
> as.numeric(as.character(5/6))
[1] 0.8333333
> 6*as.character(5/6)
Error in 6 * as.character(5/6) : non-numeric argument to binary ope
> 6*as.numeric(as.character(5/6))
[1] 5
> 5/6 == as.numeric(as.character(5/6))
[1] FALSE
```

چرا آخرین دستور $FALSE$ است؟

نام دادن به داده‌ها

می‌توان داده‌ها را نام‌گذاری کرد. این نام‌گذاری متغیرها را در اختیار ما قرار می‌دهد. بعخ نام‌ها قبلا در R ساخته شده‌اند:

```
> pi
[1] 3.141593
```

نام دادن به داده‌ها

می‌توان داده‌ها را نام‌گذاری کرد. این نام‌گذاری متغیرها را در اختیار ما قرار می‌دهد. بعخ نام‌ها قبلا در R ساخته شده‌اند:

```
> pi
[1] 3.141593
```

عملگر گمارش عبارتست از $<-$ یا $=$

```
> approx.pi <- 22/7
> approx.pi
[1] 3.142857
> diameter.in.cubits = 10
> approx.pi*diameter.in.cubits
[1] 31.42857
```

استفاده از نام‌ها و متغیرها، کدهای برنامه‌نویسی را پدید می‌آورند:

- خواندن راحت برای دیگران

نام دادن به داده‌ها

می‌توان داده‌ها را نام‌گذاری کرد. این نام‌گذاری متغیرها را در اختیار ما قرار می‌دهد. بعخ نام‌ها قبلا در R ساخته شده‌اند:

```
> pi
[1] 3.141593
```

عملگر گمارش عبارتست از $-$ یا $<$ یا $=$

```
> approx.pi <- 22/7
> approx.pi
[1] 3.142857
> diameter.in.cubits = 10
> approx.pi*diameter.in.cubits
[1] 31.42857
```

استفاده از نام‌ها و متغیرها، کدهای برنامه‌نویسی را پدید می‌آورند:

- خواندن راحت برای دیگران
- اصلاح راحت

نام دادن به داده‌ها

می‌توان داده‌ها را نام‌گذاری کرد. این نام‌گذاری متغیرها را در اختیار ما قرار می‌دهد. بعد نام‌ها قبلا در R ساخته شده‌اند:

```
> pi
[1] 3.141593
```

عملگر گمارش عبارتست از $-$ یا $=$

```
> approx.pi <- 22/7
> approx.pi
[1] 3.142857
> diameter.in.cubits = 10
> approx.pi*diameter.in.cubits
[1] 31.42857
```

استفاده از نام‌ها و متغیرها، کدهای برنامه‌نویسی را پدید می‌آورند:

- خواندن راحت برای دیگران
- اصلاح راحت
- سادگی طراحی

نام دادن به داده‌ها

می‌توان داده‌ها را نام‌گذاری کرد. این نام‌گذاری متغیرها را در اختیار ما قرار می‌دهد. بعخ نام‌ها قبلا در R ساخته شده‌اند:

```
> pi
[1] 3.141593
```

عملگر گمارش عبارتست از $-$ یا $=$

```
> approx.pi <- 22/7
> approx.pi
[1] 3.142857
> diameter.in.cubits = 10
> approx.pi*diameter.in.cubits
[1] 31.42857
```

استفاده از نام‌ها و متغیرها، کدهای برنامه‌نویسی را پدید می‌آورند:

- خواندن راحت برای دیگران
- اصلاح راحت
- سادگی طراحی
- وجود کمتر خطا

مثال: تخصیص منابع

کارخانه‌ای با استفاده از فولاد و توسط کارگرانش ماشین و کامیون تولید می‌کند:

- یک ماشین ۴۰ ساعت کار و ۱ تن فولاد نیاز دارد

مثال: تخصیص منابع

کارخانه‌ای با استفاده از فولاد و توسط کارگرانش ماشین و کامیون تولید می‌کند:

- یک ماشین ۴۰ ساعت کار و ۱ تن فولاد نیاز دارد
- یک کامیون ۶۰ ساعت کار و ۳ تن فولاد نیاز دارد

مثال: تخصیص منابع

کارخانه‌ای با استفاده از فولاد و توسط کارگرانش ماشین و کامیون تولید می‌کند:

- یک ماشین ۴۰ ساعت کار و ۱ تن فولاد نیاز دارد
- یک کامیون ۶۰ ساعت کار و ۳ تن فولاد نیاز دارد
- منابع عبارتند از: ۱۶۰۰ ساعت کار و ۷۰ تن فولاد در هر هفته

آیا می‌توان ۲۰ کامیون و ۸ ماشین تولید کرد؟

```
> 60*20 + 40*8 <= 1600
[1] TRUE
> 3*20 + 1*8 <= 70
[1] TRUE
```

مثال: تخصیص منابع

کارخانه‌ای با استفاده از فولاد و توسط کارگرانش ماشین و کامیون تولید می‌کند:

- یک ماشین ۴۰ ساعت کار و ۱ تن فولاد نیاز دارد
- یک کامیون ۶۰ ساعت کار و ۳ تن فولاد نیاز دارد
- منابع عبارتند از: ۱۶۰۰ ساعت کار و ۷۰ تن فولاد در هر هفته

آیا می‌توان ۲۰ کامیون و ۸ ماشین تولید کرد؟

```
> 60*20 + 40*8 <= 1600
[1] TRUE
> 3*20 + 1*8 <= 70
[1] TRUE
```

۲۰ کامیون و ۹ ماشین چطور؟

```
> 60*20 + 40*9 <= 1600
[1] TRUE
> 3*20 + 1*9 <= 70
[1] TRUE
```

۲۰ کامیون و ۱۰ ماشین چطور؟

۲۰ کامیون و ۱۰ ماشین چطور؟
برای پاسخ به آن می‌توان مشابه دو مورد قبل نوشت، اما:

۲۰ کامیون و ۱۰ ماشین چطور؟
برای پاسخ به آن می‌توان مشابه دو مورد قبل نوشت، اما:

- خسته‌کننده و تکراری است

۲۰ کامیون و ۱۰ ماشین چطور؟
برای پاسخ به آن می‌توان مشابه دو مورد قبل نوشت، اما:

- خسته‌کننده و تکراری است
- مراجعه به آن در آینده، ممکن است این سوال را پیش بیاورد که این اعداد چه بودند؟

۲۰ کامیون و ۱۰ ماشین چطور؟
برای پاسخ به آن می‌توان مشابه دو مورد قبل نوشت، اما:

- خسته‌کننده و تکراری است
- مراجعه به آن در آینده، ممکن است این سوال را پیش بیاورد که این اعداد چه بودند؟
- یافتن و مرتفع کردن خطاها کار مشکلی است

```
> hours.car <- 40; hours.truck <- 60
> steel.car <- 1; steel.truck <- 3
> available.hours <- 1600; available.steel <- 70
> output.trucks <- 20; output.cars <- 10
> hours.car*output.cars + hours.truck*output.trucks <= available.hours
[1] TRUE
> steel.car*output.cars + steel.truck*output.trucks <= available.steel
[1] TRUE
```

۲۰ کامیون و ۱۰ ماشین چطور؟
برای پاسخ به آن می‌توان مشابه دو مورد قبل نوشت، اما:

- خسته‌کننده و تکراری است
- مراجعه به آن در آینده، ممکن است این سوال را پیش بیاورد که این اعداد چه بودند؟
- یافتن و مرتفع کردن خطاها کار مشکلی است

```
> hours.car <- 40; hours.truck <- 60
> steel.car <- 1; steel.truck <- 3
> available.hours <- 1600; available.steel <- 70
> output.trucks <- 20; output.cars <- 10
> hours.car*output.cars + hours.truck*output.trucks <= available.hours
[1] TRUE
> steel.car*output.cars + steel.truck*output.trucks <= available.steel
[1] TRUE
```

با این شکل نوشتن، اکنون اگر لازم باشد چیزی تغییر کند فقط کافی است متغیرهای متناظر تغییر کنند و دو خط آخر را دوباره اجرا کنیم: یک مرحله به سمت ایجاز

اولین ساختار داده: بردارها

گروه‌بندی کردن مقادیر داده‌های مرتبط به هم را در یک شی، ساختار داده گویند.

یک بردار، دنباله‌ای از مقادیر هم‌نوع است:

```
> x <- c(7, 8, 10, 45)
> x
[1] 7 8 10 45
> is.vector(x)
[1] TRUE
```

اولین ساختار داده: بردارها

گروه‌بندی کردن مقادیر داده‌های مرتبط به هم را در یک شی، ساختار داده گویند.

یک بردار، دنباله‌ای از مقادیر هم‌نوع است:

```
> x <- c(7, 8, 10, 45)
> x
[1] 7 8 10 45
> is.vector(x)
[1] TRUE
```

تابع $c()$ مقادیر یک بردار را با همان ترتیبی که وارد شده است برمی‌گرداند.

$x[۱]$ اولین عنصر بردار x است، $x[۴]$ چهارمین عنصر است و $x[-۴]$ برداری است شامل همه عناصر بردار x به جز چهارمین عنصر

دستور $vector(length = ۶)$ یک بردار خالی به طول ۶ برمی‌گرداند که در بسیاری از موارد، دستور مفیدی است برای آن که بعدا توسط عناصری باید پر شوند.

```
weekly.hours <- vector(length=5)
weekly.hours[5] <- 8
```

محاسبات برداری

عملگرها بر روی بردارها به صورت جفتی عمل می‌کنند:

```
> y <- c(-7, -8, -10, -45)
> x+y
[1] 0 0 0 0
```

محاسبات برداری

عملگرها بر روی بردارها به صورت جفتی عمل می‌کنند:

```
> y <- c(-7, -8, -10, -45)
> x+y
[1] 0 0 0 0
```

دوباره‌تکراری: در عملیات جبری برداری، چنانچه برداری کوتاهتر از دیگری باشد، عناصر آن تکرار می‌شوند

```
> x + c(-7,-8)
[1] 0 0 3 37
```

بنابراین اعداد تنها، بردارهای با طول ۱ هستند که عدد مورد نظر تکرار می‌شود:

```
> x + 1
[1] 8 9 11 46
```

مقایسه‌ها نیز جفتی انجام می‌شوند:

```
[1] FALSE FALSE TRUE TRUE
```

توجه: برگرداندن بردارهای بولی

عملگرهای بولی جفتی عمل می‌کنند. اما اگر دوتایی نوشته شوند، تمام مقادیر تکی را در یک مقدار بولی تنها جمع می‌کنند:

```
> (x > 9) & (x < 20)
[1] FALSE FALSE TRUE FALSE
> (x > 9) && (x < 20)
[1] FALSE
```

برای مقایسه کل یک بردار با کل بردار دیگر، بهترین راه استفاده از $identical$ یا $all.equal$ است:

```
> x == -y
[1] TRUE TRUE TRUE TRUE
> identical(x,-y)
[1] TRUE
> identical(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
[1] FALSE
> all.equal(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
[1] TRUE
```

توابع بر روی بردارها

ورودی بسیاری از توابع، بردارها هستند

- $mean()$,$median()$,$sd()$,$var()$,$max()$,$min()$,$length()$,$sum()$ که خروجی همه آن‌ها یک عدد است

توابع بر روی بردارها

ورودی بسیاری از توابع، بردارها هستند

- $mean()$,$median()$,$sd()$,$var()$,$max()$,$min()$,$length()$,$sum()$ که خروجی همه آن‌ها یک عدد است
- $sort()$ که یک بردار جدید (مرتب‌شده) برمی‌گرداند

توابع بر روی بردارها

ورودی بسیاری از توابع، بردارها هستند

- $mean()$,$median()$,$sd()$,$var()$,$max()$,$min()$,$length()$,$sum()$ که خروجی همه آن‌ها یک عدد است
- $sort()$ که یک بردار جدید (مرتب‌شده) برمی‌گرداند
- $hist()$ که یک هیستوگرام تولید می‌کند

توابع بر روی بردارها

ورودی بسیاری از توابع، بردارها هستند

- $mean()$,$median()$,$sd()$,$var()$,$max()$,$min()$,$length()$,$sum()$ که خروجی همه آن‌ها یک عدد است
- $sort()$ که یک بردار جدید (مرتب‌شده) برمی‌گرداند
- $hist()$ که یک هیستوگرام تولید می‌کند
- $summary()$ که یک خلاصه شامل ۵ عدد را از بردارهای عددی تولید می‌کند

توابع بر روی بردارها

ورودی بسیاری از توابع، بردارها هستند

- $mean()$,$median()$,$sd()$,$var()$,$max()$,$min()$,$length()$,$sum()$ که خروجی همه آن‌ها یک عدد است
- $sort()$ که یک بردار جدید (مرتب‌شده) برمی‌گرداند
- $hist()$ که یک هیستوگرام تولید می‌کند
- $summary()$ که یک خلاصه شامل ۵ عدد را از بردارهای عددی تولید می‌کند
- $any()$ و $all()$ برای بردارهای بولی مفیدند

آدرس دادن عناصر بردارها

برداری از اندیس‌ها

```
> x[c(2,4)]
[1] 8 45
```

برداری از اندیس‌های منفی

```
> x[c(-1,-3)]
[1] 8 45
```

بردارهای بولی

```
> x[x>9]
[1] 10 45
> y[x>9]
[1] -10 -45
```

ورودی $which()$ یک بردار بولی است و یک بردار از اندیس‌هایی می‌دهد که مقادیر آن‌ها $TRUE$ هستند

```
> places <- which(x > 9)
> y[places]
[1] -10 -45
```

مولفه‌های دارای نام

می‌توان به عناصر یا مولفه‌های بردارها، نام‌هایی را اختصاص داد

```
> names(x) <- c("v1","v2","v3","Hossein")
> names(x)
[1] "v1" "v2" "v3" "Hossein"
> x[c("Hossein","v1")]
Hossein v1
45 7
```

دقت کنید این برچسب‌ها قسمتی از مقادیر بردار x نیستند. $names(x)$ خود یک بردار دیگر از (کاراکترها) است:

```
> names(y) <- names(x)
> sort(names(x))
[1] "Hossein" "v1" "v2" "v3"
> which(names(x)=="Hossein")
[1] 4
```

بازگشت به مثال تخصیص منابع

استفاده از بردارها برای گروه‌بندی چیزهایی با هم

```
> hours <- c(hours.car,hours.truck)
> steel <- c(steel.car,steel.truck)
> output <- c(output.cars,output.trucks)
> available <- c(available.hours,available.steel)
```

حالا می‌شود به صورت زیر عمل کرد:

```
> all(hours[1]*output[1]+hours[2]*output[2] <= available[1],
+ steel[1]*output[1]+steel[2]*output[2] <= available[2])
[1] TRUE
```

یا حتی

```
> all(c(sum(hours*output), sum(steel*output)) <= available)
[1] TRUE
```

اما با این شکل باید ترتیب مولفه‌ها (ماشین، کامیون، کارگر و فولاد) در هر بردار را به یاد داشته باشیم و همیشه از همان ترتیب استفاده کنیم

اما با این شکل باید ترتیب مولفه‌ها (ماشین، کامیون، کارگر و فولاد) در هر بردار را به یاد داشته باشیم و همیشه از همان ترتیب استفاده کنیم

به جای آن می‌توان از نام‌گذاری استفاده کرد

```
> names(hours) <- c("cars", "trucks")
> names(steel) <- names(hours)
> names(output) <- names(hours)
> names(available) <- c("hours","steel")
> all(hours["cars"]*output["cars"] + hours["trucks"]*
+ output["trucks"] <= available["hours"],
+ steel["cars"]*output["cars"] + steel["trucks"]*
+ output["trucks"] <= available["steel"])
[1] TRUE
```

تا اینجا کد نوشته شده بهتر شد، اما هنوز جای بهتر شدن دارد. مثل زیر:

```
> needed <- c(sum(hours*output[names(hours)]),
+ sum(steel*output[names(steel)]))
> names(needed) <- c("hours","steel")
> all(needed <= available[names(needed)])
[1] TRUE
```

این یک برنامه‌نویسی بی‌نقص نیست، اما بهتر از کدهای قبلی است.

ساختارهای برداری: شروع از آرایه‌ها

ساختارهای برداری، عبارتند از بردارها با ویژگی‌های اضافه. آرایه‌های چندبعدی، ساختارهای برداری هستند.

ساختارهای برداری: شروع از آرایه‌ها

ساختارهای برداری، عبارتند از بردارها با ویژگی‌های اضافه. آرایه‌های چندبعدی، ساختارهای برداری هستند.

```
> x.arr <- array(x,dim=c(2,2))
> x.arr
     [,1] [,2]
[1,]    7   10
[2,]    8   45
```

دقت کنید ابتدا ستون اول پر می‌شود و سپس ستون دوم. dim هم به R می‌گوید چند سطر و ستون در نظر بگیرد.

ساختارهای برداری: شروع از آرایه‌ها

ساختارهای برداری، عبارتند از بردارها با ویژگی‌های اضافه. آرایه‌های چندبعدی، ساختارهای برداری هستند.

```
> x.arr <- array(x,dim=c(2,2))
> x.arr
     [,1] [,2]
[1,]    7   10
[2,]    8   45
```

دقت کنید ابتدا ستون اول پر می‌شود و سپس ستون دوم. dim هم به R می‌گوید چند سطر و ستون در نظر بگیرد.

می‌توان آرایه‌های n بعدی هم داشت. بنابراین dim برداری با طول n خواهد بود. $n = ۲, ۳, \ldots,$

بعضی از ویژگی‌های آرایه‌ها

```
> dim(x.arr)
[1] 2 2
> is.vector(x.arr)
[1] FALSE
> is.array(x.arr)
[1] TRUE
> typeof(x.arr)
[1] "double"
> str(x.arr)
num [1:2, 1:2] 7 8 10 45
```

توجه: $typeof()$ نوع عناصر آرایه را نشان می‌دهد. $str()$ ساختار آرایه را نشان می‌دهد: در مثال بالا، آرایه عددی است، با دو بعد که هر دو با ۲ − ۱ اندیس‌گذاری شده‌اند، و در نهایت مقادیر عددی

دسترسی به و عملیات بر روی آرایه‌ها

به یک آرایه دوبعدی می‌توان با جفت‌های اندیسی یا بردار زیربنایی آرایه دسترسی داشت:

```
> x.arr[1,2]
[1] 10
> x.arr[3]
[1] 10
```

حذف یک اندیس به معنی همه آن است:

```
> x.arr[c(1:2),2]
[1] 10 45
> x.arr[,2]
[1] 10 45
```

استفاده از یک تابع برداری بر روی ساختارهای برداری، بر روی بردار زیربنایی آن‌ها کار مگر این که تابع برای کار اختصاصی با آرایه‌ها تنظیم شده باشد:

```
> which(x.arr > 9)
[1] 3 4
```

بسیاری از توابع با پیش‌فرض ساختار آرایه‌ای کار می‌کنند:

```
> y.arr <- array(y,dim=c(2,2))
> y.arr + x.arr
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

برخی دیگر به طور خاص بر روی هر سطر یا هر ستون از آرایه عمل می‌کنند:

```
> rowSums(x.arr)
[1] 17 53
```

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم
- انواع داده پایه (بولی، کاراکتر، عدد) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم
- انواع داده پایه (بولی، کاراکتر، عدد) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- ساختارهای داده پایه (بردارها و آرایه‌ها) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم
- انواع داده پایه (بولی، کاراکتر، عدد) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- ساختارهای داده پایه (بردارها و آرایه‌ها) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- یاد گرفتیم، استفاده از متغیرها، به جای ثابت‌های عددی، اولین مرحله برای خلاصه کردن کدهای برنامه‌نویسی است

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم
- انواع داده پایه (بولی، کاراکتر، عدد) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- ساختارهای داده پایه (بردارها و آرایه‌ها) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- یاد گرفتیم، استفاده از متغیرها، به جای ثابت‌های عددی، اولین مرحله برای خلاصه کردن کدهای برنامه‌نویسی است
- دانستیم داده‌ها، انواع مختلف و ساختارهای مختلفی دارند

خلاصه (موضعی)

تا اینجا:

- یاد گرفتیم چطوری برای تحلیل داده‌ها، برنامه‌نویسی کنیم
- با ترکیب توابع برای ساخت داده‌ها، برنامه‌هایی را نوشتیم
- انواع داده پایه (بولی، کاراکتر، عدد) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- ساختارهای داده پایه (بردارها و آرایه‌ها) و توابعی که بر روی آن‌ها کار می‌کنند را شناختیم
- یاد گرفتیم، استفاده از متغیرها، به جای ثابت‌های عددی، اولین مرحله برای خلاصه کردن کدهای برنامه‌نویسی است
- دانستیم داده‌ها، انواع مختلف و ساختارهای مختلفی دارند
- فهمیدیم ساختارهای داده، مقادیر مرتبط را گروه‌بندی می‌کنند

سایر ساختارهای داده

- ماتریس‌ها، *Matrices*

سایر ساختارهای داده

- ماتریس‌ها، *Matrices*
- لیست‌ها، *Lists*

سایر ساختارهای داده

- ماتریس‌ها، *Matrices*
- لیست‌ها، *Lists*
- ساختارهای داده، *Data frames*

سایر ساختارهای داده

- ماتریس‌ها، *Matrices*
- لیست‌ها، *Lists*
- ساختارهای داده، *Data frames*
- ساختارهایی از ساختارها، *Structures of structures*

ماتریس‌ها

در R یک ماتریس، حالت خاصی از یک آرایه است.

```
> factory <- matrix(c(40,1,60,3),nrow=2)
> factory
     [,1] [,2]
[1,]   40   60
[2,]    1    3
> is.array(factory)
[1] TRUE
> is.matrix(factory)
[1] TRUE
```

البته می‌توان از $ncol$ و $byrow = TRUE$ برای پر کردن سطری استفاده کرد.

ماتریس‌ها

در R یک ماتریس، حالت خاصی از یک آرایه است.

```
> factory <- matrix(c(40,1,60,3),nrow=2)
> factory
     [,1] [,2]
[1,]   40   60
[2,]    1    3
> is.array(factory)
[1] TRUE
> is.matrix(factory)
[1] TRUE
```

البته می‌توان از $ncol$ و $byrow = TRUE$ برای پر کردن سطری استفاده کرد.

عملگرهای جبری و مقایسه‌ای به صورت عنصر به عنصر عمل می‌کنند. مثلا در $factory/3$، تمام عناصر ماتریس $factory$ بر ۳ تقسیم می‌شوند.

ضرب ماتریسی، عملگر خاص خود را دارد: (برای ضرب ماتریس با بردار نیز از همین استفاده می‌شود).

```
> six.sevens <- matrix(rep(7,6),ncol=3)
> six.sevens
     [,1] [,2] [,3]
[1,]    7    7    7
[2,]    7    7    7
> factory %*% six.sevens # [2x2] * [2x3]
     [,1] [,2] [,3]
[1,]  700  700  700
[2,]   28   28   28
> six.sevens %*% factory # [2x3] * [2x2]
Error in six.sevens %*% factory : non-conformable arguments
> output <- c(10,20)
> factory %*% output
     [,1]
[1,] 1600
[2,]   70
> output %*% factory
     [,1] [,2]
[1,]  420  660
```

سایر عملگرهای ماتریسی:

```
> t(factory) # Matrix transpose
     [,1] [,2]
[1,]   40    1
[2,]   60    3
> det(factory) # Matrix determinant
[1] 60
> diag(factory) # Extracting or replacing the diagonal
[1] 40 3
> diag(factory) <- c(35,4) # Change it
> factory # See that it changed
     [,1] [,2]
[1,]   35   60
[2,]    1    4
> diag(factory) <- c(40,3) # Set it back for later
```

```
ag(c(3,4)) # Creating a diagonal matrix
     [,1] [,2]
[1,]    3    0
[2,]    0    4
> diag(2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> solve(factory) # Inverting a matrix
        [,1]    [,2]
[1,]  0.0500 -0.7500
[2,] -0.0125  0.4375
> factory %*% solve(factory)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

چرا برای معکوس کردن ماتریس‌ها از دستوری مثل $solve()$ استفاده می‌شود؟

چرا برای معکوس کردن ماتریس‌ها از دستوری مثل $solve()$ استفاده می‌شود؟

جواب‌های معادلات خطی $Ax = b$، بر حسب بردار x، به صورت زیر انجام می‌شود:

```
> available <- c(1600,70)
> solve(factory,available)
[1] 10 20
> factory %*% solve(factory,available)
     [,1]
[1,] 1600
[2,]   70
```

نام‌گذاری در ماتریس‌ها

در ماتریس‌ها، می‌توان سطرها، ستون‌ها، یا هر دو را با کمک توابع $rownames()$ و $colnames()$ نام‌گذاری کرد.

نام‌گذاری در ماتریس‌ها

در ماتریس‌ها، می‌توان سطرها، ستون‌ها، یا هر دو را با کمک توابع $rownames()$ و $colnames()$ نام‌گذاری کرد.

نام‌گذاری، به ما کمک می‌کند تا بهتر بفهمیم با چه چیزهایی کار می‌کنیم:

```
> rownames(factory) <- c("labor","steel")
> colnames(factory) <- c("cars","trucks")
> factory
      cars trucks
labor   40     60
steel    1      3
> output <- c(20,10)
> names(output) <- c("trucks","cars")
> available <- c(1600,70)
> names(available) <- c("labor","steel")
> factory %*% output # # But we've got cars and trucks mixed up
      [,1]
labor 1400
steel   50
> factory %*% output[colnames(factory)]
      [,1]
labor 1600
steel   70
> all(factory %*% output[colnames(factory)] <= available[rownames(factory)])
[1] TRUE
```

انجام کاری مشابه بر روی هر سطر یا ستون

$rowMeans()$، $colMeans()$، $rowSums()$، ...

انجام کاری مشابه بر روی هر سطر یا ستون

$rowMeans()$، $colMeans()$، $rowSums()$،...

برای ماتریس‌ها، تابع $summary()$ بر روی هر ستون به طور جداگانه اجرا می‌شود.

انجام کاری مشابه بر روی هر سطر یا ستون

$rowMeans()$، $colMeans()$، $rowSums()$،...

برای ماتریس‌ها، تابع $summary()$ بر روی هر ستون به طور جداگانه اجرا می‌شود.

یک تابع بسیار مفید، تابع $apply()$ است که دارای ۳ آرگومان هست: ماتریس، ۱ برای سطرها و ۲ برای ستون‌ها، و نام تابعی که باید اجرا شود:

```
> rowMeans(factory)
labor steel
   50     2
> apply(factory,1,mean)
labor steel
   50     2
```

دنباله‌ای از مقادیر، نه لزوماً هم‌نوع، را لیست گویند.

```
> my.distribution <- list("exponential",7,FALSE)
> my.distribution
[[1]]
[1] "exponential"

[[2]]
[1] 7

[[3]]
[1] FALSE
```

دنباله‌ای از مقادیر، نه لزوماً هم‌نوع، را لیست گویند.

```
> my.distribution <- list("exponential",7,FALSE)
> my.distribution
[[1]]
[1] "exponential"

[[2]]
[1] 7

[[3]]
[1] FALSE
```

اکثر چیزهایی که می‌توان برای بردارها به کار برد، برای لیست‌ها نیز می‌توان استفاده کرد.

دسترسی به عناصر لیست

برای دسترسی می‌توان مانند بردارها از [] یا از [[]] با یک اندیس تکی استفاده کرد:

```
> is.character(my.distribution)
[1] FALSE
> is.character(my.distribution[[1]])
[1] TRUE
> my.distribution[2]^ 2
Error in my.distribution[2]^2 : non-numeric argument
to binary operator
> my.distribution[[2]]^2
[1] 49
```

دسترسی به عناصر لیست

برای دسترسی می‌توان مانند بردارها از [] یا از [[]] با یک اندیس تکی استفاده کرد:

```
> is.character(my.distribution)
[1] FALSE
> is.character(my.distribution[[1]])
[1] TRUE
> my.distribution[2]^ 2
Error in my.distribution[2]^2 : non-numeric argument
to binary operator
> my.distribution[[2]]^2
[1] 49
```

اگر برای بردارها از [[]] برای آدرس‌دهی استفاده کنیم، چه اتفاقی می‌افتد؟

افزودن عناصری به لیست با تابع ()c امکان‌پذیر است (برای بردارهای نیز به همین صو
است).

```
> my.distribution <- c(my.distribution,7)
> my.distribution
[[1]]
[1] "exponential"
[[2]]
[1] 7
[[3]]
[1] FALSE
[[4]]
[1] 7
> length(my.distribution)
[1] 4
> length(my.distribution) <- 3
> my.distribution
[[1]]
[1] "exponential"
[[2]]
[1] 7
[[3]]
[1] FALSE
```

نام‌گذاری در لیست‌ها

بعضی یا همه عناصر یک لیست را می‌توان نام‌گذاری کرد:

```
> names(my.distribution) <- c("family","mean","is.symmetric")
> my.distribution
$family
[1] "exponential"
$mean
[1] 7
$is.symmetric
[1] FALSE
```

در این صورت به این عناصر می‌توان با نام آن‌ها همراه با علامت $ دسترسی داشت (البته $ نام و ساختار عناصر را حذف می‌کند).

```
> my.distribution$family
[1] "exponential"
> my.distribution[["family"]]
[1] "exponential"
> my.distribution["family"]
$family
[1] "exponential"
```

از نام‌گذاری در برنامه‌نویسی وقتی که یک لیست ایجاد می‌شود، زیاد استفاده می‌کنند

```
> another.distribution <- list(family="gaussian",mean=7,
+ sd=1,is.symmetric=TRUE)
> another.distribution
$family
[1] "gaussian"
$mean
[1] 7
$sd
[1] 1
$is.symmetric
[1] TRUE
```

```
my.distribution$was.estimated <- FALSE
my.distribution[["last.updated"]] <- "2011-08-30"
> my.distribution
$family
[1] "exponential"
$mean
[1] 7
$is.symmetric
[1] FALSE
$was.estimated
[1] FALSE
$last.updated
[1] "2011-08-30"
```

```
my.distribution$was.estimated <- FALSE
my.distribution[["last.updated"]] <- "2011-08-30"
> my.distribution
$family
[1] "exponential"
$mean
[1] 7
$is.symmetric
[1] FALSE
$was.estimated
[1] FALSE
$last.updated
[1] "2011-08-30"
```

حذف یک ورودی در لیست با مقداردهی آن با مقدار $NULL$ انجام می‌شود. مثلا با نوشتن

```
my.distribution$was.estimated<-NULL
```

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

ساختارهای داده

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

در بسیاری از موارد توابع موجود در R با ساختار داده‌ها سر و کار دارند.

ساختارهای داده

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

در بسیاری از موارد توابع موجود در R با ساختار داده‌ها سر و کار دارند.

ساختار داده یک ماتریس نیست، (چرا؟)

ساختارهای داده

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

در بسیاری از موارد توابع موجود در R با ساختار داده‌ها سر و کار دارند.

ساختار داده یک ماتریس نیست، (چرا؟)

ترکیبی است از یک ماتریس و یک لیست. می‌توان به ستون‌های آن مانند یک ماتریس یا قسمت‌های نام‌گذاری‌شده یک لیست دسترسی داشت.

ساختارهای داده

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

در بسیاری از موارد توابع موجود در R با ساختار داده‌ها سر و کار دارند.

ساختار داده یک ماتریس نیست، (چرا؟)

ترکیبی است از یک ماتریس و یک لیست. می‌توان به ستون‌های آن مانند یک ماتریس یا قسمت‌های نام‌گذاری‌شده یک لیست دسترسی داشت.

بسیاری از توابع ماتریس‌ها برای ساختارهای داده هم قابل به کار گیری است ($rowSums()$، $apply()$، $summary()$ و...).

ساختارهای داده

ساختار داده‌ها = جداول داده‌های کلاسیک با n سطر برای اشیا و p ستون برای متغیرها

در بسیاری از موارد توابع موجود در R با ساختار داده‌ها سر و کار دارند.

ساختار داده یک ماتریس نیست، (چرا؟)

ترکیبی است از یک ماتریس و یک لیست. می‌توان به ستون‌های آن مانند یک ماتریس یا قسمت‌های نام‌گذاری‌شده یک لیست دسترسی داشت.

بسیاری از توابع ماتریس‌ها برای ساختارهای داده هم قابل به کار گیری است ($rowSums()$، $apply()$، $summary()$ و...).

ضرب ماتریسی را نمی‌توان برای یک ساختار داده به کار برد، حتی اگر همه ورودی آن اعداد باشند.

```
> a.matrix <- matrix(c(35,8,10,4),nrow=2)
> colnames(a.matrix) <- c("v1","v2")
> a.matrix
     v1 v2
[1,] 35 10
[2,]  8  4
> a.matrix$v1 # The $ access operator doesn't work on a matrix
Error in a.matrix$v1 : $ operator is invalid for atomic vectors
> a.data.frame <- data.frame(a.matrix,logicals=c(TRUE,FALSE))
> a.data.frame
  v1 v2 logicals
1 35 10     TRUE
2  8  4    FALSE
> a.data.frame$v1 # But $ does work on a data frame
[1] 35 8
> a.data.frame[,"v1"]
[1] 35 8
> a.data.frame[1,]
  v1 v2 logicals
1 35 10     TRUE
> colMeans(a.data.frame)
v1 v2 logicals
21.5 7.0 0.5
```

با $rbind()$ و $cbind()$ می‌توان سطرها یا ستون‌هایی را به یک آرایه یا ساختار داده اضافه کرد. البته باید مراقب نوع تبدیل‌های القاشده باشید:

```
> rbind(a.data.frame,list(v1=-3,v2=-5,logicals=TRUE))
  v1 v2 logicals
1 35 10     TRUE
2  8  4    FALSE
3 -3 -5     TRUE
> rbind(a.data.frame,c(3,4,6))
  v1 v2 logicals
1 35 10        1
2  8  4        0
3  3  4        6
```

ساختارهایی از ساختارها

لیست‌هایی از لیست‌ها، لیست‌هایی از بردارها، لیست‌هایی از لیست‌هایی از لیست‌های بردارها، و ....

ساختارهایی از ساختارها

لیست‌هایی از لیست‌ها، لیست‌هایی از بردارها، لیست‌هایی از لیست‌هایی از لیست‌های بردارها، و ....

این قابلیت ایجاد ساختارها، به ما این امکان را می‌دهد تا ساختارهای داده پیچیده مورد نیاز در برخی از موارد را بتوانیم بسازیم.

ساختارهایی از ساختارها

لیست‌هایی از لیست‌ها، لیست‌هایی از بردارها، لیست‌هایی از لیست‌هایی از لیست‌های بردارها، و ....

این قابلیت ایجاد ساختارها، به ما این امکان را می‌دهد تا ساختارهای داده پیچیده مورد نیاز در برخی از موارد را بتوانیم بسازیم.

بسیاری از اشیاء پیچیده، لیست‌هایی از ساختارهای داده هستند.

مثال

تابع $eigen()$ مقادیر و بردارهای ویژه یک ماتریس را محاسبه می‌کند.

مثال

تابع $eigen()$ مقادیر و بردارهای ویژه یک ماتریس را محاسبه می‌کند.

مقادیر خروجی این تابع، لیستی از یک بردار (مقادیر ویژه) و یک ماتریس (از بردارهای ویژه) است.

```
> eigen(factory)
$values
[1] 41.556171  1.443829
$vectors
           [,1]       [,2]
[1,] 0.99966383 -0.8412758
[2,] 0.02592747  0.5406062
> class(eigen(factory))
[1] "list"
> str(eigen(factory))
List of 2
 $ values : num [1:2] 41.56 1.44
 $ vectors: num [1:2, 1:2] 0.9997 0.0259 -0.8413 0.5406
```

با اشیاء پیچیده، می‌توانید به قسمت‌هایی از قسمت‌هایی (از قسمت‌ها...) دسترسی داشته باشید:

```
> factory %*% eigen(factory)$vectors[,2]
             [,1]
labor -1.2146583
steel  0.7805429
> eigen(factory)$values[2] * eigen(factory)$vectors[,2]
[1] -1.2146583 0.7805429
> eigen(factory)$values[2]
[1] 1.443829
> eigen(factory)[[1]][[2]] # NOT [[1, 2]]
[1] 1.443829
```

خلاصه (موضعی)

- در بسیاری از مواقع با ساختارهای ماتریسی سر و کار داریم

خلاصه (موضعی)

- در بسیاری از مواقع با ساختارهای ماتریسی سر و کار داریم
- لیست‌ها این امکان را می‌دهند تا انواع متفاوتی از داده‌ها را با هم ترکیب کنیم

خلاصه (موضعی)

- در بسیاری از مواقع با ساختارهای ماتریسی سر و کار داریم
- لیست‌ها این امکان را می‌دهند تا انواع متفاوتی از داده‌ها را با هم ترکیب کنیم
- ساختارهای داده، ترکیبی از ماتریس‌ها و لیست‌ها هستند که برای جداول داده معمول به کار می‌روند

خلاصه (موضعی)

- در بسیاری از مواقع با ساختارهای ماتریسی سر و کار داریم
- لیست‌ها این امکان را می‌دهند تا انواع متفاوتی از داده‌ها را با هم ترکیب کنیم
- ساختارهای داده، ترکیبی از ماتریس‌ها و لیست‌ها هستند که برای جداول داده معمول به کار می‌روند
- استفاده از نام‌گذاری مولفه‌ها، باعث می‌شود داده‌ها معنی‌دار شده و دسترسی به آن‌ها با کنترل بیشتری صورت گیرد

خلاصه (موضعی)

- در بسیاری از مواقع با ساختارهای ماتریسی سر و کار داریم
- لیست‌ها این امکان را می‌دهند تا انواع متفاوتی از داده‌ها را با هم ترکیب کنیم
- ساختارهای داده، ترکیبی از ماتریس‌ها و لیست‌ها هستند که برای جداول داده معمول به کار می‌روند
- استفاده از نام‌گذاری مولفه‌ها، باعث می‌شود داده‌ها معنی‌دار شده و دسترسی به آن‌ها با کنترل بیشتری صورت گیرد
- استفاده مکرر از لیست‌های ساختارهای داده، منجر به تولید ساختارهای پیچیده می‌شود

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه دوم

حسین باغیشنی

دانشگاه شاهرود

عملگرهای شرطی

بعضی موارد باید به کامپیوتر گفته شود که تصمیم بگیرد کدام عمل را انجام دهد:

$$|x| = \begin{cases} x & x \geq ۰ \\ -x & x < ۰ \end{cases}$$

یا

$$\psi(x) = \begin{cases} x^۲ & |x| \leq ۱ \\ ۲|x| - ۱ & |x| > ۱ \end{cases}$$

عملگرهای شرطی

بعضی موارد باید به کامپیوتر گفته شود که تصمیم بگیرد کدام عمل را انجام دهد:

$$|x| = \begin{cases} x & x \geq ۰ \\ -x & x < ۰ \end{cases}$$

یا

$$\psi(x) = \begin{cases} x^۲ & |x| \leq ۱ \\ ۲|x| - ۱ & |x| > ۱ \end{cases}$$

تمرین: تابع $\psi(x)$ را در R رسم کنید

ساده‌ترین عملگر شرطی: if

```
if (x >= 0) {
    x
} else {
   -x
}
```

شرط موجود در if نیازمند برگرداندن مقدار $TRUE$ یا $FALSE$ است.

ساده‌ترین عملگر شرطی: if

```
if (x >= 0) {
    x
} else {
   -x
}
```

شرط موجود در if نیازمند برگرداندن مقدار $TRUE$ یا $FALSE$ است.

شرط $else$ اختیاری است.

تمام مقادیر عددی معتبر به جز ۰، $TRUE$ محسوب می‌شوند و ۰، $FALSE$ محسوب می‌شود. اکثر مقادیر غیرعددی مسدود می‌شود (خطا خواهد داد):

```
> if(1) {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if(-1) {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if(0) {"Truth!"} else {"Falsehood!"}
[1] "Falsehood!"
> if("TRUE") {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if("TRUTH") {"Truth!"} else {"Falsehood!"}
Error in if ("TRUTH") { : argument is not interpretable as logical
> if("c") {"Truth!"} else {"Falsehood!"}
Error in if ("c") { : argument is not interpretable as logical
> if(NULL) {"Truth!"} else {"Falsehood!"}
Error in if (NULL) { : argument is of length zero
> if(NA) {"Truth!"} else {"Falsehood!"}
Error in if (NA) { : missing value where TRUE/FALSE needed
> if(NaN) {"Truth!"} else {"Falsehood!"}
Error in if (NaN) { : argument is not interpretable as logical
```

عملگرهای منطقی: تکی یا دوگانه؟

عملگرهای منطقی & و | مانند عملگرهای حسابی هستند، یعنی بر روی بردارها عنصر عنصر عمل می‌کنند:

```
> c(TRUE,TRUE) & c(TRUE,FALSE)
[1] TRUE FALSE
```

برای کنترل حلقه‌های برنامه، معمولاً برنامه نیاز به مقادیر بولی تکی دارد. به این معنی که نیاز داریم به کامپیوتر بگوییم چیزی که نیاز نداریم را محاسبه نکن.

عملگرهای منطقی: تکی یا دوگانه؟

عملگرهای منطقی & و | مانند عملگرهای حسابی هستند، یعنی بر روی بردارها عنصر عنصر عمل می‌کنند:

```
> c(TRUE,TRUE) & c(TRUE,FALSE)
[1] TRUE FALSE
```

برای کنترل حلقه‌های برنامه، معمولاً برنامه نیاز به مقادیر بولی تکی دارد. به این معنی که نیاز داریم به کامپیوتر بگوییم چیزی که نیاز نداریم را محاسبه نکن.

راه حل استفاده از && و || است: از چپ به راست حرکت کن و وقتی به جواب رسیدی متوقف شو.

```
> (0>0) & ("c"+1)
Error in "c" + 1 : non-numeric argument to binary operator
> (0>0) && ("c"+1)
[1] FALSE
```

با مثال بالا، تصور کنید برای مولفه دوم محاسبات پیچیده‌ای نیاز باشد. R از آن صرفنظر می‌کند زیرا نیازی به محاسبه آن نیست!

اگر بر روی بردارها عمل شود، عملگرهای بولی دوتایی اعضای اول هر بردار را برای مقایسه در نظر می‌گیرند:

```
> c(FALSE,FALSE) | c(TRUE,FALSE)
[1] TRUE FALSE
> c(FALSE,FALSE) || c(TRUE,FALSE)
[1] TRUE
> c(FALSE,FALSE) || c(FALSE,TRUE)
[1] FALSE
```

به طور کلی: از && و || برای کنترل برنامه‌ها استفاده کنید و سعی کنید ورودی آن‌ها آرگومان‌های برداری نباشند

عملگرهای شرطی آشیانه‌ای

عملگرهای شرطی می‌توانند به صورت آشیانه‌ای به کار روند:

```
if (x^2 < 1) {
   x^2
} else {
   if (x >= 0) {
   x
   } else {
   -x
   }
}
```

به جای عمل خسته‌کننده آشیانه‌ای کردن عملگرهای $if/else$، می‌توان از $switch$ استفاده کرد (چگونه؟)

تکرار: انجام چیزهایی مشابه برای چندین بار

تکرار عملی یکسان (یا خیلی مشابه) به چندین بار مشخص:

```
n <- 10
table.of.logarithms <- vector(length=n)
table.of.logarithms
for (i in 1:n) {
   table.of.logarithms[i] <- log(i)
}
table.of.logarithms
```

حلقه for یک شمارنده را (در اینجا i) به کمک مقادیر یک بردار (در اینجا n : ۱) افزایش می‌دهد و بدنه حلقه را مادامی که کل بردار اجرا شود، تکرار می‌کند

تکرار: انجام چیزهایی مشابه برای چندین بار

تکرار عملی یکسان (یا خیلی مشابه) به چندین بار مشخص:

```
n <- 10
table.of.logarithms <- vector(length=n)
table.of.logarithms
for (i in 1:n) {
   table.of.logarithms[i] <- log(i)
}
table.of.logarithms
```

حلقه for یک شمارنده را (در اینجا i) به کمک مقادیر یک بردار (در اینجا n : ۱) افزایش می‌دهد و بدنه حلقه را مادامی که کل بردار اجرا شود، تکرار می‌کند

توجه: روشی بهتر و کارآمدتر برای انجام این وظیفه وجود دارد.

ترکیب *for* و *if*

```
x <- c(-5,7,-8,0)
y <- vector(length=length(x))
for (i in 1:length(x)) {
   if (x[i] >= 0) {
      y[i] <- x[i]
   } else {
      y[i] <- -x[i]
   }
}
y # now c(5,7,8,0)
```

توجه: روشی بهتر و کارآمدتر برای انجام این وظیفه وجود دارد.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

حلقه مادامی که ورودی شرط $FALSE$ شود، اجرا می‌شود.

- اگر شرط همیشه $TRUE$ باشد، حلقه تا ابد تکرار می‌شود.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

حلقه مادامی که ورودی شرط $FALSE$ شود، اجرا می‌شود.

- اگر شرط همیشه $TRUE$ باشد، حلقه تا ابد تکرار می‌شود.
- هیچ وقت شروع نمی‌شود، مگر این که شرط با $TRUE$ آغاز شود.

تمرین: چطور یک حلقه *for* را با *while* جایگزین می‌کنید؟

تکرارهای غیرشرطی

```
repeat {
   print("Help! I am hba, trapped in an endless loop!")
}
```

یا مفیدتر از آن به صورت زیر است:

```
repeat {
   if (watched) {
     next()
   }
   print("Help! I am Dr. Morris Culpepper, trapped in an endless loo
   if (rescued) {
     break()
   }
}
```

همیشه حداقل یک بار وارد حلقه می‌شویم، حتی اگر $rescued$ برابر $TRUE$ باشد.

تکرارهای غیرشرطی

```
repeat {
   print("Help! I am hba, trapped in an endless loop!")
}
```

یا مفیدتر از آن به صورت زیر است:

```
repeat {
   if (watched) {
     next()
   }
   print("Help! I am Dr. Morris Culpepper, trapped in an endless loo
   if (rescued) {
     break()
   }
}
```

همیشه حداقل یک بار وارد حلقه می‌شویم، حتی اگر $rescued$ برابر $TRUE$ باشد.
$break()$ ما را از حلقه خارج می‌کند و $next()$ از مابقی بدنه (در اینجا if) صرفنظر می‌کند و به حلقه برمی‌گردد

تخصیص منابع: دوباره

یادآوری:

- تولید اتومبیل و کامیون توسط کارخانه با فولاد و ساعت کار کارگران
- منابع موجود شامل ۱۶۰۰ ساعت کار و ۷۰ تن فولاد با تولید ۱۰ اتومبیل و ۲۰ کامیون به طور کامل به کار گرفته می‌شوند
- مساله به طور دقیق با برنامه‌ریزی خطی حل می‌شود
- فرض کنید چیزی از جبر خطی نمی‌دانیم و مهم نیست چقدر از منابع، مادامی که یک حد اولیه موجود باشد، مصرف شود
- یافتن جواب با در نظر گرفتن یک طرح اولیه شروع شده و با بسط آن مادامی که قیدها رخ دهند، به دست می‌آید

```
factory <- matrix(c(40,1,60,3),nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
available <- c(1600,70); names(available) <- rownames(factory)
slack <- c(8,1); names(slack) <- rownames(factory)
output <-c(30,20); names(output) <- colnames(factory)
```

چطور عمل می‌کند؟

```
passes <- 0 # How many times have we been around the loop?
repeat {
passes <- passes + 1
needed <- factory %*% output # What do we need for that output level?
# If we're not using too much, and are within the slack, we're done
if (all(needed <= available) &&
all((available - needed) <= slack)) {
break()
}
# If we're using too much of everything, cut back by 10%
if (all(needed > available)) {
output <- output * 0.9
next()
}
# If we're using too little of everything, increase by 10%
if (all(needed < available)) {
output <- output * 1.1
next()
}
# If we're using too much of some resources but not others, randomly
# tweak the plan by up to 10%
output <- output * (1+runif(length(output),min=-0.1,max=0.1))
}
```

خروجی بعد از شروع با ۳۰ اتومبیل و ۲۰ کامیون:

```
> round(output,1)
cars trucks
10.4 19.7
> round(needed,1)
[,1]
labor 1596.1
steel 69.4
> passes
[1] 3452
```

یعنی کد نوشته‌شده، طرح اولیه را ۳۴۵۲ بار تصحیح می‌کند و نتیجه را با حالت واقعی ۱۰ اتومبیل و ۲۰ کامیون مقایسه کنید.

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند
- کدهای ساده‌تری خواهند داشت

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند
- کدهای ساده‌تری خواهند داشت
- سریعتر هستند (گاهی کمی سریع و گاهی خیلی خیلی سریعتر)

اکثر این روش‌ها در مورد محاسبات با بردارسازی می‌باشد

مثال

اکثر زبان‌های برنامه‌نویسی چطوری دو بردار a و b را با هم جمع می‌کنند:

```
c <- vector(length(a))
for (i in 1:length(a)) {
   c[i] <- a[i] + b[i]
}
```

و R چطوری دو بردار را با هم جمع می‌کند:

```
c <- a+b
```

مزایا:

- وضوح: علامت + در مورد چیزی است که در حال انجام آن هستیم
- خلاصه‌سازی: این عملگر آنچه را که کامپیوتر انجام می‌دهد، پنهان می‌کند
- کوتاه کردن: یک خط به جای چهار خط
- سرعت

معایب:

- باید یاد بگیرید در مورد کل اشیاء فکر کنید نه قسمت‌هایی از آن‌ها

محاسبات برداری‌شده

بسیاری از توابع به طور خودکار برای بردارسازی تنظیم شده‌اند:

```
abs(x) # Absolute value of each element in x
log(x) # Logarithm of each element in x
```

شرطی‌سازی با $ifelse()$:

```
ifelse(x<0,-x,x) # Pretty much the same as abs(x)
ifelse(x^2>1,abs(x),x^2)
```

تکرارسازی

- $rep(x, n)$: x را n بار تکرار می‌کند

تکرارسازی

- $rep(x, n)$: x را n بار تکرار می‌کند
- $seq()$: یک دنباله تولید می‌کند

آرایه‌ها با ساختارهای تکرارشده

همه ترکیبات مقادیری از بردارها با $expand.grid()$ قابل انجام است:

```
> expand.grid(v1=c("lions","tigers"),v2=c(0.1,1.1))
      v1  v2
1  lions 0.1
2 tigers 0.1
3  lions 1.1
4 tigers 1.1
```

یک ساختار داده تولید می‌کند به طوری که می‌توانید انواع متفاوتی از داده‌ها را با هم ترکیب کنید

آرایه‌ها با ساختارهای تکرارشده

همه ترکیبات مقادیری از بردارها با $expand.grid()$ قابل انجام است:

```
> expand.grid(v1=c("lions","tigers"),v2=c(0.1,1.1))
     v1  v2
1  lions 0.1
2 tigers 0.1
3  lions 1.1
4 tigers 1.1
```

یک ساختار داده تولید می‌کند به طوری که می‌توانید انواع متفاوتی از داده‌ها را با هم ترکیب کنید

بیشتر از یک بردار ورودی مناسب است

ترکیب ورودی‌های یک تابع: *outer*

```
:er(c(1,3,5),c(2,3,7),'*')
     [,1] [,2] [,3]
[1,]    2    3    7
[2,]    6    9   21
[3,]   10   15   35
```

برای عملگرها نیاز به علامت " هست.
البته برای دستور بالا، می‌توان از عملگر مخصوص خود استفاده کرد

```
c(1,3,5) %o% c(2,3,7)
```

با *outer* هر تابع برداری‌شده دو مقداره، قابل کاربرد است:

```
> outer(c(1024,1000),c(2,10),log)
          [,1]   [,2]
[1,] 10.000000 3.0103
[2,]  9.965784 3.0000
```

$replicate()$: یک چیز دقیقا یکسان را چندین بار انجام می‌دهد

```
# Take a sample of size 1000 from the standard exponential
rexp(1000,rate=1)
# Take the mean of such a sample
mean(rexp(1000,rate=1))
# Draw 1000 such samples, and take the mean of each one
replicate(1000,mean(rexp(1000),rate=1))
# Plot the histogram of sample means
hist(replicate(1000,mean(rexp(1000,rate=1))))

# Equivalent to that last, but dumb
sample.means <- vector(length=1000)
for (i in 1:length(sample.means)) {
sample.means[i] <- mean(rexp(1000,rate=1))
}
hist(sample.means)
```

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه سوم

حسین باغیشنی

دانشگاه شاهرود

چرا توابع؟

ساختارهای داده، مقادیر مرتبط با هم را در یک شیء گرد هم می‌آورند

چرا توابع؟

ساختارهای داده، مقادیر مرتبط با هم را در یک شیء گرد هم می‌آورند

توابع، دستورهای مرتبط با هم را در یک شیء به هم پیوند می‌دهند.

```
my.clever.function <- function(an.argument,another.argument) {
    # clever calculations
    return(important.result)
}
```

```
# "Robust" loss function, for outlier-resistant regression
# Inputs: vector of numbers (x)
# Outputs: vector with x^2 for small entries, 2|x|-1 for large ones
psi.1 <- function(x) {
    psi <- ifelse(x^2 > 1, 2*abs(x)-1, x^2)
    return(psi)
}

> z <- c(-0.5,-5,0.9,9)
> psi.1(z)
[1] 0.25 9.00 0.81 17.00
```

به قسمت‌های مختلف تابع نوشته‌شده دقت کنید.

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

دستور $return()$ نشان‌دهنده خروجی تابع است

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

دستور $return()$ نشان‌دهنده خروجی تابع است

توجه: همیشه یک خط توضیح در مورد هدف تابع نوشته‌شده، فهرست کردن آرگومان‌های تابع، و خروجی‌های آن خیلی مفید خواهد بود

آرگومان‌های نام‌گذاری‌شده و پیش‌فرض

```
# "Robust" loss function, for outlier-resistant regression
# Inputs: vector of numbers (x), scale for crossover (c)
# Outputs: vector with x^2 for small entries, 2c|x|-c^2 for large ones
psi.2 <- function(x,c=1) {
psi <- ifelse(x^2 > c^2, 2*c*abs(x)-c^2, x^2)
return(psi)
}
> identical(psi.1(z), psi.2(z,c=1))
[1] TRUE
```

چنانچه نام یک آرگومان تابع در استفاده از آن تابع نباشد، مقادیر پیش‌فرض در نظر گرفته می‌شوند.

```
> identical(psi.2(z,c=1), psi.2(z))
[1] TRUE
```

آرگومان‌های نام‌دارشده، چنانچه به طور واضح برچسب خورده باشد، به هر ترتیب دلخواهی قابل استفاده است

```
> identical(psi.2(x=z,c=2), psi.2(c=2,x=z))
[1] TRUE
```

بررسی آرگومان‌ها

یک مشکل: مشاهده نتایج عجیب زمانی که آرگومان‌های مورد استفاده مورد انتظار نیستند

```
> psi.2(x=z,c=c(1,1,1,10))
[1] 0.25 9.00 0.81 81.00
> psi.2(x=z,c=-1)
[1] 0.25 -11.00 0.81 -19.00
```

پاسخ: در کد نوشته شده چند خط برای چک کردن این گونه آرگومان‌ها بنویسید

```
psi.3 <- function(x,c=1) {
# Scale should be a single positive number
stopifnot(length(c) == 1,c>0)
psi <- ifelse(x^2 > c^2, 2*c*abs(x)-c^2, x^2)
return(psi)
}
```

آرگومان‌های $stopifnot()$ عبارت‌هایی هستند که باید ارزیابی همه آن‌ها $TRUE$ باشد. به محض رخداد اولین $FALSE$ اجرا، همراه با پیغام خطا، متوقف می‌شود.

یک تابع چه می‌تواند ببیند و انجام دهد؟

نام آرگومان‌ها ویژه درون تابع هستند و در یک محیط بزرگتر باطل می‌شوند. در واقع تغییراتی که در درون تابع انجام می‌شوند، به خارج آن سرایت نمی‌کنند.

```
> x <- 7
> y <- c("A","C","G","T","U")
> adder <- function(y) { x<- x+y; return(x) }
> adder(1)
[1] 8
> x
[1] 7
> y
[1] "A" "C" "G" "T" "U"
```

جستجو در محیط R

هر نامی که در تابع تعریف نشده باشد، مقدار آن در محیط، جستجو می‌شود.

جستجو در محیط R

هر نامی که در تابع تعریف نشده باشد، مقدار آن در محیط، جستجو می‌شود.

چیزی که مهم است مقدار این گونه نام‌ها در هنگام فراخوانی تابع است نه هنگام تعریف کردن آن

```
> circle.area <- function(r) { return(pi*r^2) }
> circle.area(c(1,2,3))
[1] 3.141593 12.566371 28.274334
> truepi <- pi
> pi <- 3
> circle.area(c(1,2,3))
[1] 3 12 27
> pi <- truepi # Restore sanity
> circle.area(c(1,2,3))
[1] 3.141593 12.566371 28.274334
```

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود
- استثناء هم وجود دارد: شناخته‌شده‌های عمومی مثل π

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود
- استثناء هم وجود دارد: شناخته‌شده‌های عمومی مثل π
- به طور مشابه، خروجی تابع باید فقط از طریق مقادیر برگشت‌شده قابل دسترس باشند

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

اخیراً وجود یک رابطه غیرخطی بین تعداد جمعیت یک شهر، N، و تولید ناخالص شهری، Y، به صورت زیر، پیشنهاد شده است:

$$Y = y_0 N^a + \epsilon_0$$

به این مدل، یک مدل قانون توانی می‌گویند.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

اخیراً وجود یک رابطه غیرخطی بین تعداد جمعیت یک شهر، N، و تولید ناخالص شهری، Y ، به صورت زیر، پیشنهاد شده است:

$$Y = y_0 N^a + \epsilon.$$

به این مدل، یک مدل قانون توانی می‌گویند.

می‌توان پارامترها را با می‌نیمم کردن MSE به دست آورد

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - y_0 N_i^a)^2,$$

که در آن y_0 معلوم و برابر ۶۶۱۱ دلار می‌باشد.

ادامه مثال

می‌توان پارامتر a را با تقریب زدن مشتق MSE و به روش تکراری به دست آورد (؟)

$$
\begin{aligned}
MSE(a) &= \frac{1}{n}\sum_{i=1}^{n}(Y_i - y_0 N_i^a)^2 \\
MSE'(a) &\approx \frac{MSE(a+h) - MSE(a)}{h} \\
a_{t+1} - a_t &\propto -MSE'(a)
\end{aligned}
$$

```
maximum.iterations <- 100
deriv.step <- 1/1000
step.scale <- 1e-12
stopping.deriv <- 1/100
iteration <- 0
deriv <- Inf
a <- 0.15
while ((iteration < maximum.iterations) && (deriv > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
list(a=a,iterations=iteration,converged=(iteration < maximum.iterations))
```

مشکلات کد نوشته‌شده

- غیرقابل انعطاف: برای تغییر حدس اولیه برای a باید آن را ویرایش کرد و سپس کد را کپی‌ـ پیست کرد و دوباره آن را اجرا کرد

مشکلات کد نوشته‌شده

- غیرقابل انعطاف: برای تغییر حدس اولیه برای a باید آن را ویرایش کرد و سپس کد را کپی‌ـ پیست کرد و دوباره آن را اجرا کرد
- پاک کردن خطا: چنانچه مجموعه داده تغییر کند، باید دوباره ویرایش و اجرای دوباره انجام شود و امیدوار باشیم همه ویرایش‌ها درست انجام شده‌اند

با تبدیل این کد به یک تابع و بهتر کردن آن، این مشکلات مرتفع خواهند شد.

اولین تلاش

```
estimate.scaling.exponent.1 <- function(a) {
maximum.iterations <- 100
deriv.step <- 1/1000
step.scale <- 1e-12
stopping.deriv <- 1/100
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

مشکل: همه اعدادی است که عجیب هم به نظر می‌رسند

پاسخ: به عنوان آرگومان تابع با مقادیر پیش‌فرض تعریف شوند

```
estimate.scaling.exponent.2 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - y0*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - y0*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

تمرین: با مقادیر متفاوتی از $deriv.step$ برنامه را اجرا کنید

```
estimate.scaling.exponent.2 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - y0*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - y0*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

تمرین: با مقادیر متفاوتی از $deriv.step$ برنامه را اجرا کنید

مشکل: چرا محاسبات مشابه برای MSE دو بار انجام می‌شود؟

پاسخ: آن را به صورت یک تابع معرفی کن

```
estimate.scaling.exponent.3 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
mse <- function(a) { mean((gmp$pcgmp - y0*gmp$pop^a)^2) }
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

دقت کنید تابع mse درون تابع $estimate.scaling.exponent$ قرار گرفته است و به محیط R تعلق ندارد. یعنی با فراخوانی آن در محیط بزرگتر R پیغام خطا خواهید دید. اما مثلا مقدار y_0 را می‌توانید ببینید.

مشکل: این تابع فقط برای مجموعه داده gmp نوشته شده است. اگر بخواهیم با مجموعه داده دیگری نتایج را مقایسه کنیم، باید تابع دیگری بنویسیم؟

پاسخ: افزودن آرگومان‌های بیشتر با مقادیر پیش‌فرض

```
estimate.scaling.exponent.4 <- function(a, y0=6611, response=gmp$pcgmp,
predictor = gmp$pop, maximum.iterations=100, deriv.step = 1/100,
step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
mse <- function(a) { mean((response - y0*predictor^a)^2) }
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

می‌توان حلقه $while()$ را به حلقه $for()$ تبدیل کرد

```
estimate.scaling.exponent.5 <- function(a, y0=6611, response=gmp$pcgmp,
predictor = gmp$pop, maximum.iterations=100, deriv.step = 1/100,
step.scale = 1e-12, stopping.deriv = 1/100) {
mse <- function(a) { mean((response - y0*predictor^a)^2) }
for (iteration in 1:maximum.iterations) {
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
if (abs(deriv) <= stopping.deriv) { break() }
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد
- می‌توانید مسایل مشابه را حل کنید

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد
- می‌توانید مسایل مشابه را حل کنید
- می‌توان مساله را به قسمت‌های کوچکتر تجزیه کرد و برای هر قسمت یک تابع نوشت

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

برای این منظور، باید مدل را به عنوان یک شیء تعریف کنیم و در توابع مرتبط با هم باید یک سازگاری در رابط‌ها وجود داشته باشد:

- توابع نوشته شده برای یک نوع مشابه از اشیاء، باید آرگومان‌های مشابه داشته باشند و یک ساختار مشابه را در نظر بگیرند

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

برای این منظور، باید مدل را به عنوان یک شیء تعریف کنیم و در توابع مرتبط با هم باید یک سازگاری در رابط‌ها وجود داشته باشد:

- توابع نوشته شده برای یک نوع مشابه از اشیاء، باید آرگومان‌های مشابه داشته باشند و یک ساختار مشابه را در نظر بگیرند
- توابعی که برای انجام یک کار مشابه نوشته می‌شوند باید آرگومان‌های مشابه داشته باشند و مقادیر خروجی مشابهی گزارش دهند

در نهایت همه توابع مرتبط با هم را در یک فایل تکی قرار دهید و برای توضیح هر کدام چند خط توضیح بنویسید

ادامه مثال مدل رگرسیون

پیش‌گویی مقادیر جدید با استفاده از مدل برازش‌شده

```
# Predict response values from a power-law scaling model
# Inputs: fitted power-law model (object), vector of values at which to make
# predictions at (newdata)
# Outputs: vector of predicted response values
predict.plm <- function(object, newdata) {
# Check that object has the right components
stopifnot("a" %in% names(object), "y0" %in% names(object))
a <- object$a
y0 <- object$y0
# Sanity check the inputs
stopifnot(is.numeric(a),length(a)==1)
stopifnot(is.numeric(y0),length(y0)==1)
stopifnot(is.numeric(newdata))
return(y0*newdata^a) # Actual calculation and return
}
```

```
# Plot fitted curve from power law model over specified range
# Inputs: list containing parameters (plm), start and end of range (from, to)
# Outputs: TRUE, silently, if successful
# Side-effect: Makes the plot
plot.plm.1 <- function(plm,from,to) {
# Take sanity-checking of parameters as read
y0 <- plm$y0 # Extract parameters
a <- plm$a
f <- function(x) { return(y0*x^a) }
curve(f(x),from=from,to=to)
# Return with no visible value on the terminal
invisible(TRUE)
}
```

وقتی یک تابع، تابع دیگری را فراخوانی می‌کند، از ... به عنوان یک فرا آرگومان استفاده می‌شود تا ورودی‌های نامشخص را به تابع فراخوانی‌شده واگذار کند

```
plot.plm.2 <- function(plm,...) {
y0 <- plm$y0
a <- plm$a
f <- function(x) { return(y0*x^a) }
# from and to are possible arguments to curve()
curve(f(x),...)
invisible(TRUE)
}
```

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر
- اصلاح و بهبود دادن راحت‌تر

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر
- اصلاح و بهبود دادن راحت‌تر
- طراحی ساده‌تر

یک قانون سرانگشتی: تابعی طولانی‌تر از یک صفحه، احتمالا بیش از حد طولانی است

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

معایب آن‌ها: ارزیابی هر بار آن‌ها در هر بار فراخوانی تابع بزرگتر است و این که در محیط بزرگتر نمی‌توان به آن‌ها دسترسی داشت

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

معایب آن‌ها: ارزیابی هر بار آن‌ها در هر بار فراخوانی تابع بزرگتر است و این که در محیط بزرگتر نمی‌توان به آن‌ها دسترسی داشت

جانشین: نوشتن جداگانه آن‌ها و تعبیه همگی در یک فایل و یکی کردن آن‌هاست.

مثال

```
plot.plm.3 <- function(plm,from,to,n=101,...) {
x <- seq(from=from,to=to,length.out=n)
y <- predict.plm(object=plm,newdata=x)
plot(x,y,...)
invisible(TRUE)
}
```

مثال تخصیص منابع

```
planner <- function(output,factory,available,slack,tweak=0.1) {
needed <- plan.needs(output,factory)
if (all(needed <= available) && all(available-needed <= slack)) {
return(list(output=output,needed=needed))
}
else {
output <- adjust.plan(output,needed,available,tweak)
return(planner(output,factory,available,slack))
}
}
plan.needs <- function(output,factory) { factory %*% output }
adjust.plan <- function(output,needed,available,tweak) {
if (all(needed >= available)) { return(output*(1-tweak)) }
if (all(needed < available)) { return((1+tweak)) }
return(output*runif(n=length(output),min=1-tweak,max=1+tweak))
}
```

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه چهارم: تولید تحقق‌هایی از متغیرهای تصادفی

حسین باغیشنی

دانشگاه شاهرود

تولید اعداد تصادفی

برای تولید تحقق‌هایی از هر متغیر تصادفی، باید ابتدا اعدادی تصادفی تولید شوند

تولید اعداد تصادفی

برای تولید تحقق‌هایی از هر متغیر تصادفی، باید ابتدا اعدادی تصادفی تولید شوند

به طور کلی، با ابزار موجود، نمی‌توان اعداد کاملا تصادفی تولید کرد.

تولید اعداد تصادفی

برای تولید تحقق‌هایی از هر متغیر تصادفی، باید ابتدا اعدادی تصادفی تولید شوند

به طور کلی، با ابزار موجود، نمی‌توان اعداد کاملا تصادفی تولید کرد.

روش‌های مختلفی وجود دارند که به کمک آن‌ها (البته با امیدواری) اعدادی نزدیک به کاملا تصادفی تولید می‌شوند.

تولید اعداد تصادفی

برای تولید تحقق‌هایی از هر متغیر تصادفی، باید ابتدا اعدادی تصادفی تولید شوند

به طور کلی، با ابزار موجود، نمی‌توان اعداد کاملا تصادفی تولید کرد.

روش‌های مختلفی وجود دارند که به کمک آن‌ها (البته با امیدواری) اعدادی نزدیک به کاملا تصادفی تولید می‌شوند.

به چنین اعدادی، شبه‌تصادفی می‌گویند.

دنیای خیلی کوچک تولید اعداد شبه‌تصادفی

در دنیای تصادفی واقعی: نتیجه پرتاب n بار یک سکه، یک حالت از ۲^n حالت ممکن با احتمال ۲^{-n} است

دنیای خیلی کوچک تولید اعداد شبه‌تصادفی

در دنیای تصادفی واقعی: نتیجه پرتاب n بار یک سکه، یک حالت از ۲^n حالت ممکن با احتمال ۲^{-n} است

در دنیای واقعی (با استفاده از رایانه‌ها برای پرتاب n بار سکه): تعداد حالات ممکن محدود به $۲^{۳۲}$ است. زیرا

دنیای خیلی کوچک تولید اعداد شبه‌تصادفی

در دنیای تصادفی واقعی: نتیجه پرتاب n بار یک سکه، یک حالت از ۲^n حالت ممکن با احتمال ۲^{-n} است

در دنیای واقعی (با استفاده از رایانه‌ها برای پرتاب n بار سکه): تعداد حالات ممکن محدود به $۲^{۳۲}$ است. زیرا

اکثر تولیدکننده‌های اعداد شبه‌تصادفی بر پایه هسته‌های ($seed$) ۳۲ بیتی هستند.

دنیای خیلی کوچک تولید اعداد شبه‌تصادفی

در دنیای تصادفی واقعی: نتیجه پرتاب n بار یک سکه، یک حالت از ۲^n حالت ممکن با احتمال ۲^{-n} است

در دنیای واقعی (با استفاده از رایانه‌ها برای پرتاب n بار سکه): تعداد حالات ممکن محدود به $۲^{۳۲}$ است. زیرا

اکثر تولیدکننده‌های اعداد شبه‌تصادفی بر پایه هسته‌های ($seed$) ۳۲ بیتی هستند.

با مقایسه ۲^n و $۲^{۳۲}$ برای nهای بزرگ، می‌توان عملکرد ناپخته تولیدکننده‌های اعداد تصادفی را درک کرد.

دنیای خیلی کوچک تولید اعداد شبه‌تصادفی

در دنیای تصادفی واقعی: نتیجه پرتاب n بار یک سکه، یک حالت از ۲^n حالت ممکن با احتمال ۲^{-n} است

در دنیای واقعی (با استفاده از رایانه‌ها برای پرتاب n بار سکه): تعداد حالات ممکن محدود به $۲^{۳۲}$ است. زیرا

اکثر تولیدکننده‌های اعداد شبه‌تصادفی بر پایه هسته‌های $(seed)$ ۳۲ بیتی هستند.

با مقایسه ۲^n و $۲^{۳۲}$ برای nهای بزرگ، می‌توان عملکرد ناپخته تولیدکننده‌های اعداد تصادفی را درک کرد.

این شرایط در عمل خیلی هم بد نیست. زیرا معمولا علاقه‌مند به اطلاع از مقادیر دقیق بردار n بعدی نیستیم، بلکه به خلاصه‌ای از آن‌ها با بعدی خیلی کوچکتر، $d << ۲^{۳۲}$ نیاز داریم.

اعداد شبه‌تصادفی

در این درس، فرض می‌کنیم اعداد شبه‌تصادفی در دسترس هستند.

اعداد شبه‌تصادفی

در این درس، فرض می‌کنیم اعداد شبه‌تصادفی در دسترس هستند.

اعداد شبه‌تصادفی مورد نظر با دستور $runif()$ در R تولید می‌شوند.

نمونه‌گیری از یک جامعه متناهی

برای تولید یک نمونه از یک جامعه متناهی، می‌توان از دستور *sample* استفاده کرد

```
> # tose some coins
> sample(0:1, size=10, replace=TRUE)
 [1] 0 0 1 1 0 1 1 0 1 0
 > # choose some lottery numbers
 > sample(1:100, size=6, replace=FALSE)
[1] 53 33 12 91  3 89
> # sample from a multinomial distribution
> x <- sample(1:3, size = 100, replace = TRUE,
prob = c(.2, .3, .5))
> table(x)
x
 1  2  3
18 40 42
```

تولید از توزیع‌های احتمال معمول

در R دستوراتی مرتبط با توزیع‌های احتمالی معروف وجود دارد.

تولید از توزیع‌های احتمال معمول

در R دستوراتی مرتبط با توزیع‌های احتمالی معروف وجود دارد.

برای هر توزیع (گسسته یا پیوسته)، توابعی مرتبط با:

- تابع (چگالی) احتمال، d?

تولید از توزیع‌های احتمال معمول

در R دستوراتی مرتبط با توزیع‌های احتمالی معروف وجود دارد.

برای هر توزیع (گسسته یا پیوسته)، توابعی مرتبط با:

- تابع (چگالی) احتمال، d?
- تابع توزیع تجمعی، p?

تولید از توزیع‌های احتمال معمول

در R دستوراتی مرتبط با توزیع‌های احتمالی معروف وجود دارد.

برای هر توزیع (گسسته یا پیوسته)، توابعی مرتبط با:

- تابع (چگالی) احتمال، d?
- تابع توزیع تجمعی، p?
- تابع چندک، q?

تولید از توزیع‌های احتمال معمول

در R دستوراتی مرتبط با توزیع‌های احتمالی معروف وجود دارد.

برای هر توزیع (گسسته یا پیوسته)، توابعی مرتبط با:

- تابع (چگالی) احتمال، d?
- تابع توزیع تجمعی، p?
- تابع چندک، q?
- تولید اعداد تصادفی، r?

وجود دارند. به عنوان مثال برای توزیع دوجمله‌ای

```
dbinom(x, size, prob, log=FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

برای هر توزیع، یک نام اختصاری در نظر گرفته شده است که با این حروف ترکیب می‌شود.

برای هر توزیع، یک نام اختصاری در نظر گرفته شده است که با این حروف ترکیب می‌شود.

برای اطلاع از نام‌های اختصاری مرتبط با هر توزیع، به منبع **مقدمه‌ای بر** R که قبلا معرفی کرده‌ایم، مراجعه کنید.

برای هر توزیع، یک نام اختصاری در نظر گرفته شده است که با این حروف ترکیب می‌شود.

برای اطلاع از نام‌های اختصاری مرتبط با هر توزیع، به منبع **مقدمه‌ای بر** R که قبلا معرفی کرده‌ایم، مراجعه کنید.

دقت کنید برای توزیع‌هایی مثل گاما، که می‌توان دو شکل متفاوت پارامتری برای آن در نظر گرفت، با مشخص کردن پارامتر $rate$ یا $scale$ نوع شکل پارامتری مورد نظر خود را به R توضیح می‌دهید.

```
rgamma(n, shape, rate = 1, scale = 1/rate)
```

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

برای تولید متغیرهای تصادفی از توزیع‌های معمول و غیرمعمول، روش‌های مختلفی وجود دارند، که در این جا به چند روش از آن‌ها می‌پردازیم.

- روش تبدیل معکوس

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

برای تولید متغیرهای تصادفی از توزیع‌های معمول و غیرمعمول، روش‌های مختلفی وجود دارند، که در این جا به چند روش از آن‌ها می‌پردازیم.

- روش تبدیل معکوس
- روش پذیرش ـ رد

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

برای تولید متغیرهای تصادفی از توزیع‌های معمول و غیرمعمول، روش‌های مختلفی وجود دارند، که در این جا به چند روش از آن‌ها می‌پردازیم.

- روش تبدیل معکوس
- روش پذیرش ـ رد
- روش‌های تبدیل

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

برای تولید متغیرهای تصادفی از توزیع‌های معمول و غیرمعمول، روش‌های مختلفی وجود دارند، که در این جا به چند روش از آن‌ها می‌پردازیم.

- روش تبدیل معکوس
- روش پذیرش ـ رد
- روش‌های تبدیل
- تولید از توزیع‌های آمیخته

روش‌های تولید متغیرهای تصادفی

با وجود آن که برای تولید متغیرهای تصادفی در R به طور گسترده توابعی وجود دارند، اما در موارد زیادی با توزیع‌هایی مواجه می‌شویم که فرم تابع چگالی (احتمال) آن‌ها آشنا نیست یا اصلاً فرم بسته‌ای برای آن‌ها وجود ندارد.

برای تولید متغیرهای تصادفی از توزیع‌های معمول و غیرمعمول، روش‌های مختلفی وجود دارند، که در این جا به چند روش از آن‌ها می‌پردازیم.

- روش تبدیل معکوس
- روش پذیرش ـ رد
- روش‌های تبدیل
- تولید از توزیع‌های آمیخته
- تولید از توزیع‌های چندمتغیره

روش تبدیل معکوس

تبدیل انتگرال احتمال. اگر X یک متغیر تصادفی پیوسته با تابع توزیع $F_X(x)$ باشد، آنگاه $U = F_X(X) \sim U(۰, ۱)$.
با استفاده از این قضیه، می‌توان متغیرهای تصادفی را تولید کرد. تبدیل معکوس را می‌توان به صورت زیر تعریف کرد:

$$F_X^{-۱}(u) = \inf\{x : F_X(x) = u\}, \quad ۰ < u < ۱.$$

اگر $U \sim U(۰, ۱)$، آنگاه برای هر $x \in \Re$،

$$P(F_X^{-۱}(U) \leq x) = P(\inf\{t : F_X(t) = U\} \leq x) = P(U \leq F_X(x)) = F_X(x),$$

بنابراین $F_X^{-۱}(U)$ با X هم‌توزیع است. پس برای تولید تحققی از X باید

- تابع معکوس $F_X^{-۱}(u)$ را محاسبه کنید

روش تبدیل معکوس

تبدیل انتگرال احتمال. اگر X یک متغیر تصادفی پیوسته با تابع توزیع $F_X(x)$ باشد، آنگاه $U = F_X(X) \sim U(۰, ۱)$.
با استفاده از این قضیه، می‌توان متغیرهای تصادفی را تولید کرد. تبدیل معکوس را می‌توان به صورت زیر تعریف کرد:

$$F_X^{-۱}(u) = \inf\{x : F_X(x) = u\},\ ۰ < u < ۱.$$

اگر $U \sim U(۰, ۱)$، آنگاه برای هر $x \in \Re$،

$$P(F_X^{-۱}(U) \leq x) = P(\inf\{t : F_X(t) = U\} \leq x) = P(U \leq F_X(x)) = F_X(x),$$

بنابراین $F_X^{-۱}(U)$ با X هم‌توزیع است. پس برای تولید تحققی از X باید

- تابع معکوس $F_X^{-۱}(u)$ را محاسبه کنید
- یک مشاهده از توزیع $U(۰, ۱)$ تولید کنید

روش تبدیل معکوس

تبدیل انتگرال احتمال. اگر X یک متغیر تصادفی پیوسته با تابع توزیع $F_X(x)$ باشد، آنگاه $U = F_X(X) \sim U(۰, ۱)$.
با استفاده از این قضیه، می‌توان متغیرهای تصادفی را تولید کرد. تبدیل معکوس را می‌توان به صورت زیر تعریف کرد:

$$F_X^{-۱}(u) = \inf\{x : F_X(x) = u\},\ ۰ < u < ۱.$$

اگر $U \sim U(۰, ۱)$، آنگاه برای هر $x \in \Re$،

$$P(F_X^{-۱}(U) \leq x) = P(\inf\{t : F_X(t) = U\} \leq x) = P(U \leq F_X(x)) = F_X(x),$$

بنابراین $F_X^{-۱}(U)$ با X هم‌توزیع است. پس برای تولید تحققی از X باید

- تابع معکوس $F_X^{-۱}(u)$ را محاسبه کنید
- یک مشاهده از توزیع $U(۰, ۱)$ تولید کنید
- مشاهده متغیر به صورت $x = F_X^{-۱}(u)$ خواهد بود

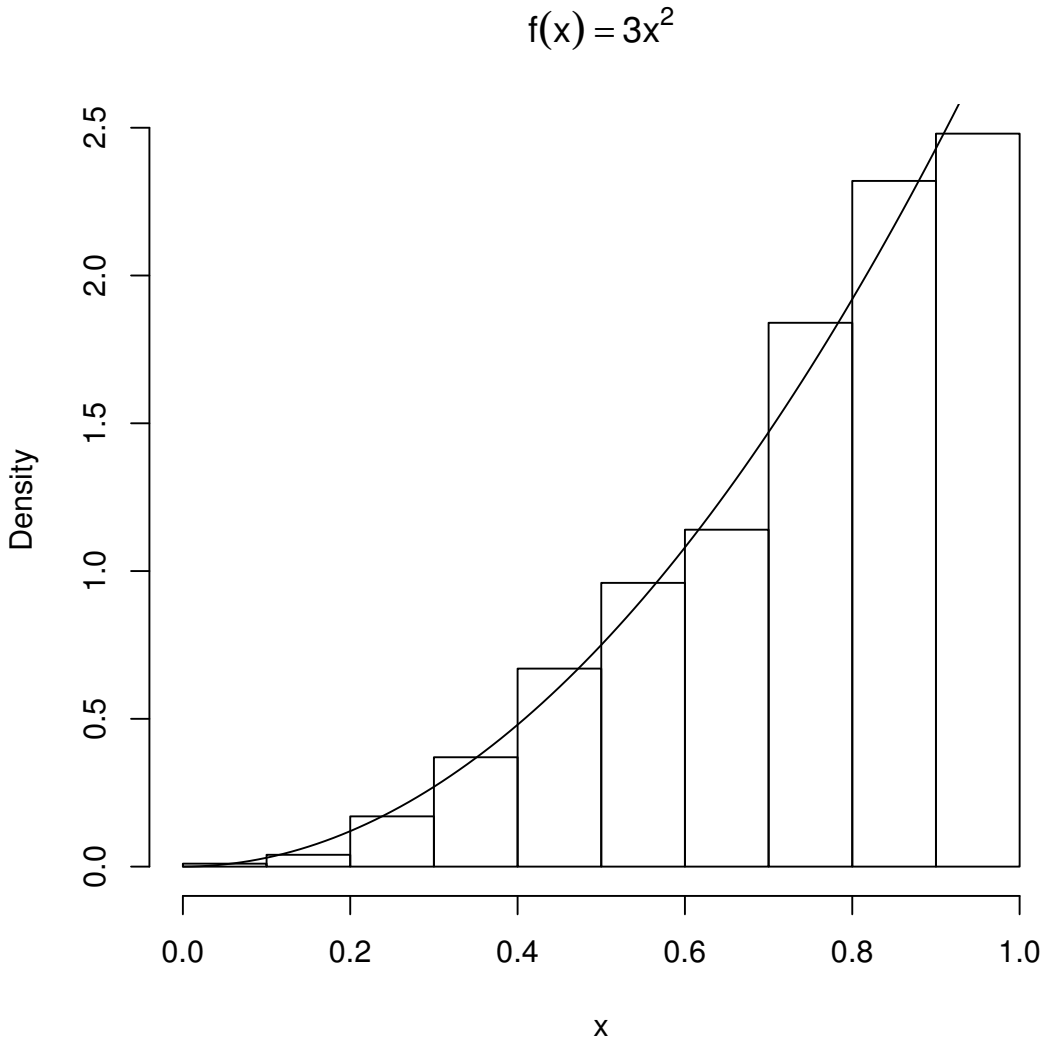
مثال: حالت پیوسته

$$f_X(x) = ۳x^۲, \quad ۰ < x < ۱.$$

$$F_X(x) = x^۳ \longrightarrow F_X^{-۱}(u) = u^{۱/۳}.$$

```
> n <- 1000
> u <- runif(n)
> x <- u^(1/3)
> # density histogram of sample
> # main = expression(f(x)==3*x^2)
> hist(x, prob = TRUE, main = bquote(f(x)==3*x^2))
> y <- seq(0, 1, .01)
> lines(y, 3*y^2)    #density curve f(x)
```

تمرین. برای توزیع نمایی با میانگین ۲، یک نمونه ۱۰۰۰ تایی تولید کرده و با دستور *rexp* مقایسه کنید


$f(x) = 3x^2$
Density
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.2
0.4
0.6
0.8
1.0
x

مثال: حالت گسسته

این روش برای توزیع‌های گسسته هم قابل استفاده است.

مثال: حالت گسسته

این روش برای توزیع‌های گسسته هم قابل استفاده است.
اگر X یک متغیر تصادفی گسسته باشد و $\cdots < x_{i-1} < x_i < \cdots$ نقاط ناپیوستگی تابع توزیع $F_X(x)$ باشند، آنگاه تبدیل معکوس زمانی که $F_X(x_{i-1}) < u \leq F_X(x_i)$ عبارتست از $F_X^{-1}(u) = x_i$. بنابراین

- از توزیع یکنواخت استاندارد، یک مشاهده تولید کنید

مثال: حالت گسسته

این روش برای توزیع‌های گسسته هم قابل استفاده است.
اگر X یک متغیر تصادفی گسسته باشد و $\cdots < x_{i-1} < x_i < \cdots$ نقاط ناپیوستگی تابع توزیع $F_X(x)$ باشند، آنگاه تبدیل معکوس زمانی که $F_X(x_{i-1}) < u \leq F_X(x_i)$ عبارتست از $F_X^{-1}(u) = x_i$. بنابراین

- از توزیع یکنواخت استاندارد، یک مشاهده تولید کنید
- هر گاه $F(x_{i-1}) < u \leq F(x_i)$، مقدار x_i را به عنوان مشاهده تولیدشده برگردانید

مثال: حالت گسسته

این روش برای توزیع‌های گسسته هم قابل استفاده است.
اگر X یک متغیر تصادفی گسسته باشد و $\cdots < x_{i-1} < x_i < \cdots$ نقاط ناپیوستگی تابع توزیع $F_X(x)$ باشند، آنگاه تبدیل معکوس زمانی که $F_X(x_{i-1}) < u \leq F_X(x_i)$ عبارتست از $x_i = F_X^{-1}(u)$. بنابراین

- از توزیع یکنواخت استاندارد، یک مشاهده تولید کنید
- هر گاه $F(x_{i-1}) < u \leq F(x_i)$، مقدار x_i را به عنوان مشاهده تولیدشده برگردانید

دقت کنید که محاسبه $F(x_{i-1}) < u \leq F(x_i)$ برای بعضی از توزیع‌ها می‌تواند خیلی مشکل باشد

توزیع برنولی

فرض کنید $X \sim Ber(p = ۰/۴)$. بنابراین $F_X(۰) = f(۰) = ۱ - p$ و $F_X(۱) = ۱$.
بنابراین اگر $u > ۰/۶$، آنگاه $F_X^{-۱}(u) = ۱$ و زمانی که $u \leq ۰/۶$، $F_X^{-۱}(u) = ۰$

```
> n <- 1000
> p <- 0.4
> u <- runif(n)
> x <- as.integer(u > 0.6)   #(u > 0.6) is a logical vector
> ####
> mean(x)
[1] 0.399
>     var(x)
[1] 0.240039
> ## rbinom(n, size = 1, prob = p)
> ## sample(c(0,1), size = n, replace = TRUE, prob = c(.6,.4))
```

در اینجا گشتاورهای نمونه و توزیع واقعی با هم مقایسه شده‌اند. میانگین توزیع همان p است و واریانس $p(۱ - p) = ۰/۲۴$.

توزیع هندسی

$$f(x) = pq^x,\ x = ۰, ۱, ۲, \cdots$$

به طوری که $p = \frac{۱}{۴}$ و $q = ۱ - p$.
در نقاط ناپیوستگی $x = ۰, ۱, ۲, \cdots$، $F(x) = ۱ - q^{x+۱}$. بنابراین باید نامساوی زیر را حل کنیم:

$$۱ - q^x < u \leq ۱ - q^{x+۱}.$$

نتیجه می‌شود:

$$x < \frac{\log(۱ - u)}{\log(q)} \leq x + ۱.$$

بنابراین پاسخ عبارتست از

$$x + ۱ = \lceil \frac{\log(۱ - u)}{\log(q)} \rceil,$$

که در آن، $\lceil t \rceil$ کوچکترین عدد صحیحی است که از t کوچکتر نیست.

```
> n <- 1000
> p <- 0.25
> u <- runif(n)
> k <- ceiling(log(1-u) / log(1-p)) - 1
> # more efficient
> # k <- floor(log(u) / log(1-p))
```

دقت کنید که U و $U - ۱$ هم‌توزیع هستند.

```
> n <- 1000
> p <- 0.25
> u <- runif(n)
> k <- ceiling(log(1-u) / log(1-p)) - 1
> # more efficient
> # k <- floor(log(u) / log(1-p))
```

دقت کنید که U و $1 - U$ هم‌توزیع هستند.

دقت کنید چون حل نامساوی ذکرشده برای توزیع هندسی ساده بود، با روش تبدیل معکوس به راحتی می‌توان از این توزیع نمونه تولید کرد.

موارد مشکل‌تر

روشی مشابه برای توزیع پواسون پیچیده‌تر است چون که برای x نمی‌توان از نامساوی ذکرشده یک فرمول سرراست پیدا کرد
روش پایه برای تولید یک متغیر پواسون با پارامتر λ، تولید و ذخیره تابع توزیع با استفاده از یک فرمول بازگشتی به شکل زیر است:

$$f(x+1) = \frac{\lambda f(x)}{x+1}, \quad F(x+1) = F(x) + f(x+1).$$

در این حالت، یک متغیر یکنواخت استاندارد تولید می‌شود و در بردار مقادیر تابع توزیع به دنبال مشاهده‌ای می‌گردیم که نامساوی $F(x-1) < u \leq F(x)$ برای آن برقرار باشد.

توزیع لگاریتمی

این مثال، توضیحات مرتبط با توزیع پواسون را تشریح می‌کند. دقت کنید که برای توزیع لگاریتمی، تابعی در R وجود ندارد.

$$f(x) = P(X = x) = \frac{a\theta^x}{x}, \; x = ۱, ۲, \cdots$$

که در آن، $۰ < \theta < ۱$ و $a = (-\log(۱ - \theta))^{-۱}$.

$$f(x + ۱) = \frac{\theta^x}{x + ۱} f(x).$$

با این روابط، برای xهای بزرگ، محاسبات از دقت کافی برخوردار نیستند. یک راه حل استفاده از

$$\exp(\log a + x \log \theta - \log x),$$

است.

```
rlogarithmic <- function(n, theta) {
        #returns a random logarithmic(theta) sample size n
        u <- runif(n)
        #set the initial length of cdf vector
        N <- ceiling(-16 / log10(theta))
        k <- 1:N
        a <- -1/log(1-theta)
        fk <- exp(log(a) + k * log(theta) - log(k))
        Fk <- cumsum(fk)
        x <- integer(n)
        for (i in 1:n) {
            x[i] <- as.integer(sum(u[i] > Fk)) #F^{-1}(u)-1
            while (x[i] == N) {
                #if x==N we need to extend the cdf
                #very unlikely because N is large
                logf <- log(a) + (N+1)*log(theta) - log(N+1)
                fk <- c(fk, exp(logf))
                Fk <- c(Fk, Fk[N] + fk[N+1])
                N <- N + 1
                x[i] <- as.integer(sum(u[i] > Fk))
            }
        }
        x + 1
    }
```

```
> n <- 1000
> theta <- 0.5
> x <- rlogarithmic(n, theta)
> # compute density of logarithmic(theta) for comparison
> k <- sort(unique(x))
> p <- -1 / log(1 - theta) * theta^k / k
> se <- sqrt(p*(1-p)/n)   # standard error
> round(rbind(table(x)/n, p, se),3)
       1     2     3     4     5     6     8     9
   0.713 0.194 0.060 0.019 0.009 0.002 0.001 0.002
p  0.721 0.180 0.060 0.023 0.009 0.004 0.001 0.000
se 0.014 0.012 0.008 0.005 0.003 0.002 0.001 0.001
```

روش پذیرش ـ رد

فرض کنید X و Y متغیرهای تصادفی با توابع (چگالی) احتمال به ترتیب f و g باشند.
همچنین ثابت c وجود داشته باشد به طوری که، برای تمام xهایی که $f(x) > 0$ داشته باشیم:

$$\frac{f(x)}{g(x)} \leq c.$$

در این حالت می‌توان از روش پذیرش ـ رد (یا روش رد)، برای تولید متغیر X استفاده کرد.

۱. یافتن یک متغیر تصادفی Y با تابع احتمال g که رابطه $f(x)/g(x) \leq c$، برای تمام xهایی که $f(x) > 0$، برقرار باشد.

روش پذیرش ـ رد

فرض کنید X و Y متغیرهای تصادفی با توابع (چگالی) احتمال به ترتیب f و g باشند.
همچنین ثابت c وجود داشته باشد به طوری که، برای تمام xهایی که $f(x) > 0$ داشته باشیم:

$$\frac{f(x)}{g(x)} \leq c.$$

در این حالت می‌توان از روش پذیرش ـ رد (یا روش رد)، برای تولید متغیر X استفاده کرد.

1. یافتن یک متغیر تصادفی Y با تابع احتمال g که رابطه $f(x)/g(x) \leq c$، برای تمام xهایی که $f(x) > 0$، برقرار باشد.
2. متغیر Y را تولید کنید

روش پذیرش ـ رد

فرض کنید X و Y متغیرهای تصادفی با توابع (چگالی) احتمال به ترتیب f و g باشند.
همچنین ثابت c وجود داشته باشد به طوری که، برای تمام xهایی که $f(x) > 0$ داشته باشیم:

$$\frac{f(x)}{g(x)} \leq c.$$

در این حالت می‌توان از روش پذیرش ـ رد (یا روش رد)، برای تولید متغیر X استفاده کرد.

1. یافتن یک متغیر تصادفی Y با تابع احتمال g که رابطه $f(x)/g(x) \leq c$، برای تمام xهایی که $f(x) > 0$، برقرار باشد.
2. متغیر Y را تولید کنید
3. یک متغیر تصادفی U از یکنواخت استاندارد تولید کنید

روش پذیرش ـ رد

فرض کنید X و Y متغیرهای تصادفی با توابع (چگالی) احتمال به ترتیب f و g باشند.
همچنین ثابت c وجود داشته باشد به طوری که، برای تمام xهایی که $f(x) > 0$ داشته باشیم:

$$\frac{f(x)}{g(x)} \leq c.$$

در این حالت می‌توان از روش پذیرش ـ رد (یا روش رد)، برای تولید متغیر X استفاده کرد.

1. یافتن یک متغیر تصادفی Y با تابع احتمال g که رابطه $f(x)/g(x) \leq c$، برای تمام xهایی که $f(x) > 0$، برقرار باشد.
2. متغیر Y را تولید کنید
3. یک متغیر تصادفی U از یکنواخت استاندارد تولید کنید
4. اگر $u < f(y)/(cg(y))$، y را بپذیر و قرار بده $x = y$. در غیر این صورت y را رد کن و از مرحله ۲ شروع به تکرار کن

در مرحله ۴،

$$P(Accept|\,Y) = P(\,U < \frac{f(\,Y)}{cg(\,Y)}|\,Y) = \frac{f(\,Y)}{cg(\,Y)}.$$

بنابراین برای هر تکرار، احتمال پذیرش کل برابر است با

$$\sum_y P(Accept|y)P(\,Y = y) = \sum_y \frac{f(y)}{cg(y)} g(y) = \frac{۱}{c},$$

و تعداد تکرارها تا پذیرش یک نمونه دارای توزیع هندسی با میانگین c است.

در مرحله ۴،

$$P(Accept|\,Y) = P(\,U < \frac{f(\,Y)}{cg(\,Y)}|\,Y) = \frac{f(\,Y)}{cg(\,Y)}.$$

بنابراین برای هر تکرار، احتمال پذیرش کل برابر است با

$$\sum_y P(Accept|y)P(\,Y = y) = \sum_y \frac{f(y)}{cg(y)} g(y) = \frac{۱}{c},$$

و تعداد تکرارها تا پذیرش یک نمونه دارای توزیع هندسی با میانگین c است.
بنابراین برای تولید هر نمونه از X، c تکرار، به طور متوسط، لازم است. در نتیجه برای کارایی روش، باید c کوچک و تولید از Y ساده باشد.

در مرحله ۴،

$$P(Accept|Y) = P(U < \frac{f(Y)}{cg(Y)}|Y) = \frac{f(Y)}{cg(Y)}.$$

بنابراین برای هر تکرار، احتمال پذیرش کل برابر است با

$$\sum_y P(Accept|y)P(Y = y) = \sum_y \frac{f(y)}{cg(y)} g(y) = \frac{1}{c},$$

و تعداد تکرارها تا پذیرش یک نمونه دارای توزیع هندسی با میانگین c است.
بنابراین برای تولید هر نمونه از X، c تکرار، به طور متوسط، لازم است. در نتیجه برای کارایی روش، باید c کوچک و تولید از Y ساده باشد.
سوال: آیا نمونه پذیرش‌شده با روش رد، با X هم‌توزیع است؟

$$P(k|Accept) = \frac{P(Accept|k)g(k)}{P(Accept)} = \frac{[f(k)/(cg(k)]g(k)}{1/c} = f(k).$$

توزیع بتا

تابع چگالی $Beta(۲,۲)$ به صورت $f(x) = ۶x(۱-x),\ ۰ < x < ۱$ است. فرض کنید $g(x)$ یکنواخت استاندارد باشد.

$$\forall ۰ < x < ۱;\ \frac{f(x)}{g(x)} \leq ۶ \longrightarrow c = ۶.$$

یک متغیر x پذیرفته می‌شود اگر

$$\frac{f(x)}{cg(x)} = \frac{۶x(۱-x)}{۶(۱)} = x(۱-x) > u.$$

به طور متوسط برای تولید یک نمونه به حجم ۱۰۰۰، نیاز به $cn = ۶۰۰۰$ تکرار (که برابر تولید ۱۲۰۰۰ عدد تصادفی است) داریم.

```
n <- 1000
    k <- 0       #counter for accepted
    j <- 0       #iterations
    y <- numeric(n)

    while (k < n) {
        u <- runif(1)
        j <- j + 1
        x <- runif(1)  #random variate from g
        if (x * (1-x) > u) {
            #we accept x
            k <- k + 1
            y[k] <- x
        }
    }


>     j
[1] 5921
```

```
>     p <- seq(.1, .9, .1)
>     Qhat <- quantile(y, p)   # quantiles of sample
>     Q <- qbeta(p, 2, 2)      # theoretical quantiles
>     se <- sqrt(p * (1-p) / (n * dbeta(Q, 2, 2)^2))
>     round(rbind(Qhat, Q, se), 3)
       10%   20%   30%   40%   50%   60%   70%   80%   90%
Qhat 0.193 0.287 0.373 0.450 0.516 0.577 0.629 0.701 0.788
Q    0.196 0.287 0.363 0.433 0.500 0.567 0.637 0.713 0.804
se   0.010 0.010 0.010 0.011 0.011 0.011 0.010 0.010 0.010
```

برای بهتر شدن دقت، باید حجم نمونه را افزایش داد

```
> round(rbind(Qhat, Q, se), 3)
       10%   20%   30%   40%   50%   60%   70%   80%   90%
Qhat 0.198 0.292 0.370 0.439 0.504 0.573 0.641 0.719 0.804
Q    0.196 0.287 0.363 0.433 0.500 0.567 0.637 0.713 0.804
se   0.003 0.003 0.003 0.003 0.003 0.003 0.003 0.003 0.003
```

روش تبدیل

سایر روش‌های تبدیل (به غیر از تبدیل معکوس احتمال) را نیز می‌توان برای تولید متغیرهای تصادفی به کار برد

۱. $Z \sim N(۰,۱) \longrightarrow V = Z^۲ \sim \chi^۲(۱)$

روش تبدیل

سایر روش‌های تبدیل (به غیر از تبدیل معکوس احتمال) را نیز می‌توان برای تولید متغیرهای تصادفی به کار برد

۱. $Z \sim N(۰,۱) \longrightarrow V = Z^۲ \sim \chi^۲(۱)$
۲. $U \sim \chi^۲(m), V \sim \chi^۲(n) \longrightarrow F = \frac{U/m}{V/n} \sim F(m,n)$

روش تبدیل

سایر روش‌های تبدیل (به غیر از تبدیل معکوس احتمال) را نیز می‌توان برای تولید متغیرهای تصادفی به کار برد

۱. $Z \sim N(۰,۱) \longrightarrow V = Z^۲ \sim \chi^۲(۱)$
۲. $U \sim \chi^۲(m), V \sim \chi^۲(n) \longrightarrow F = \frac{U/m}{V/n} \sim F(m,n)$
۳. $U, V \sim U(۰,۱) \longrightarrow Z_۱ = \sqrt{-۲\log U}\cos(۲\pi V), Z_۲ = \sqrt{-۲\log V}\sin(۲\pi U) \sim N(۰,۱)$

روش تبدیل

سایر روش‌های تبدیل (به غیر از تبدیل معکوس احتمال) را نیز می‌توان برای تولید متغیرهای تصادفی به کار برد

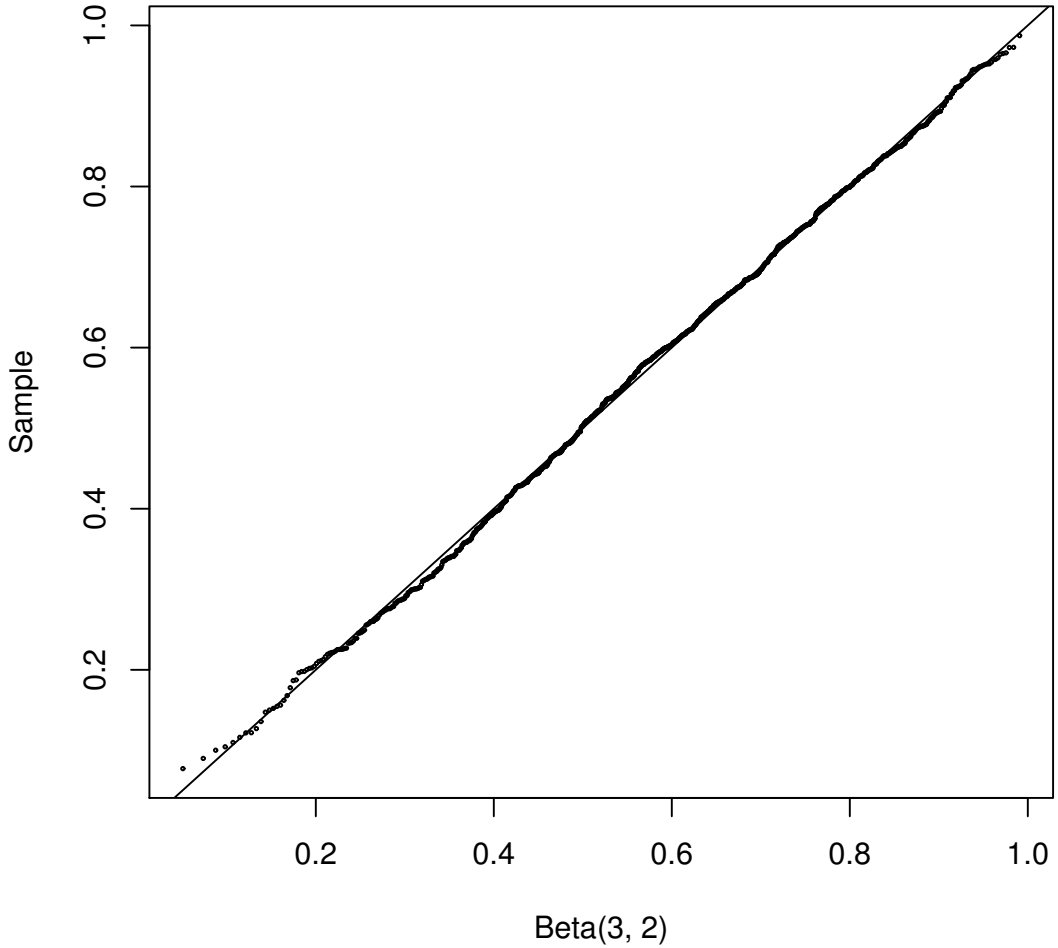
۱. $Z \sim N(۰,۱) \longrightarrow V = Z^۲ \sim \chi^۲(۱)$
۲. $U \sim \chi^۲(m), V \sim \chi^۲(n) \longrightarrow F = \frac{U/m}{V/n} \sim F(m,n)$
۳. $U, V \sim U(۰,۱) \longrightarrow Z_۱ = \sqrt{-۲\log U}\cos(۲\pi V), Z_۲ = \sqrt{-۲\log V}\sin(۲\pi U) \sim N(۰,۱)$
۴. $U, V \sim U(۰,۱) \longrightarrow X = \lfloor ۱ + \frac{\log V}{\log(۱-(۱-\theta)^U)} \rfloor \sim logarithmic(\theta)$

در بالا نماد $\lfloor t \rfloor$ قسمت صحیح t را نشان می‌دهد.

توزیع بتا

$$U \sim Gamma(r, \lambda), V \sim Gamma(s, \lambda) \longrightarrow X = \frac{U}{U+V} \sim Beta(r, s)$$

```
> n <- 1000
> a <- 3
> b <- 2
> u <- rgamma(n, shape=a, rate=1)
> v <- rgamma(n, shape=b, rate=1)
> x <- u / (u + v)
> q <- qbeta(ppoints(n), a, b)
> qqplot(q, x, cex=0.25, xlab="Beta(3, 2)", ylab="Sample")
> abline(0, 1)
```


Sample
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0
Beta(3, 2)

توزیع لگاریتمی

```
> n <- 1000
> theta <- 0.5
> u <- runif(n)  #generate logarithmic sample
> v <- runif(n)
> x <- floor(1 + log(v) / log(1 - (1 - theta)^u))
> k <- 1:max(x)  #calc. logarithmic probs.
> p <- -1 / log(1 - theta) * theta^k / k
> se <- sqrt(p*(1-p)/n)
> p.hat <- tabulate(x)/n
> print(round(rbind(p.hat, p, se), 3))
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
p.hat 0.729 0.171 0.062 0.029 0.005 0.003 0.000 0.000 0.001
p     0.721 0.180 0.060 0.023 0.009 0.004 0.002 0.001 0.000
se    0.014 0.012 0.008 0.005 0.003 0.002 0.001 0.001 0.001
```

توزیع لگاریتمی

```
rlogarithmic <- function(n, theta) {
        stopifnot(all(theta > 0 & theta < 1))
        th <- rep(theta, length=n)
        u <- runif(n)
        v <- runif(n)
        x <- floor(1 + log(v) / log(1 - (1 - th)^u))
        return(x)
    }
```

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه پنجم: تولید تحقق‌هایی از متغیرهای تصادفی

حسین باغیشنی

دانشگاه شاهرود

۱۵ اسفند ۱۳۹۳

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

به ثابت c ، ثابت نرمال‌ساز هم می‌گویند.

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

به ثابت c، ثابت نرمال‌ساز هم می‌گویند.

در اجرای روش پذیرش ـ رد، تنها دانستن نسبت f/g کافی است و الگوریتم به ثابت نرمال‌ساز نیازی ندارد.

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

به ثابت c، ثابت نرمال‌ساز هم می‌گویند.

در اجرای روش پذیرش ـ رد، تنها دانستن نسبت f/g کافی است و الگوریتم به ثابت نرمال‌ساز نیازی ندارد.

نیازی نیست کران $f \leq cg$ خیلی تند و تیز(!) باشد. با جایگزین کردن c با یک مقدار بزرگتر، الگوریتم باز هم به درستی اجرا می‌شود. هرچند کارایی آن کاهش می‌یابد.

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

به ثابت c، ثابت نرمال‌ساز هم می‌گویند.

در اجرای روش پذیرش ـ رد، تنها دانستن نسبت f/g کافی است و الگوریتم به ثابت نرمال‌ساز نیازی ندارد.

نیازی نیست کران $f \leq cg$ خیلی تند و تیز(!) باشد. با جایگزین کردن c با یک مقدار بزرگتر، الگوریتم باز هم به درستی اجرا می‌شود. هرچند کارایی آن کاهش می‌یابد.

احتمال پذیرش برابر $1/c$ است. بنابراین تا جای ممکن c باید کوچک باشد تا کارایی بیشتری داشته باشد.

چند نکته دیگر در مورد روش رد

روش پذیرش ـ رد در هر بعدی قابل به کارگیری است، البته با این شرط که g تابع (چگالی) احتمال بر روی فضای یکسان f باشد.

به ثابت c، ثابت نرمال‌ساز هم می‌گویند.

در اجرای روش پذیرش ـ رد، تنها دانستن نسبت f/g کافی است و الگوریتم به ثابت نرمال‌ساز نیازی ندارد.

نیازی نیست کران $f \leq cg$ خیلی تند و تیز(!) باشد. با جایگزین کردن c با یک مقدار بزرگتر، الگوریتم باز هم به درستی اجرا می‌شود. هرچند کارایی آن کاهش می‌یابد.

احتمال پذیرش برابر $1/c$ است. بنابراین تا جای ممکن c باید کوچک باشد تا کارایی بیشتری داشته باشد.

یک ایراد روش رد و پذیرش، آن است که متغیرهای غیرمفید را، زمانی که مقدار پیشنهادی از g رد می‌شود، تولید می‌کند. برای مرتفع کردن این مشکل، بعدا روش نمونه‌گیری با اهمیت را معرفی می‌کنیم.

روش رد برای توزیع بتا

قصد داریم از توزیع $Beta(۲/۷, ۶/۳)$ با روش رد و پذیرش، نمونه تولید کنیم. ابتدا دقت کنید که کران بالا، c، مقدار ماکسیمم تابع چگالی احتمال بتاست که با دستور زیر قابل دستیابی است:

```
c=optimize(f=function(x){dbeta(x,2.7,6.3)},
interval=c(0,1),maximum=T)$objective
```

تابع g را یکنواخت استاندارد در نظر بگیرید. بنابراین مقدار پیشنهادی y پذیرفته می‌شود هرگاه $c \times U < f(y)$.

روش رد برای توزیع بتا

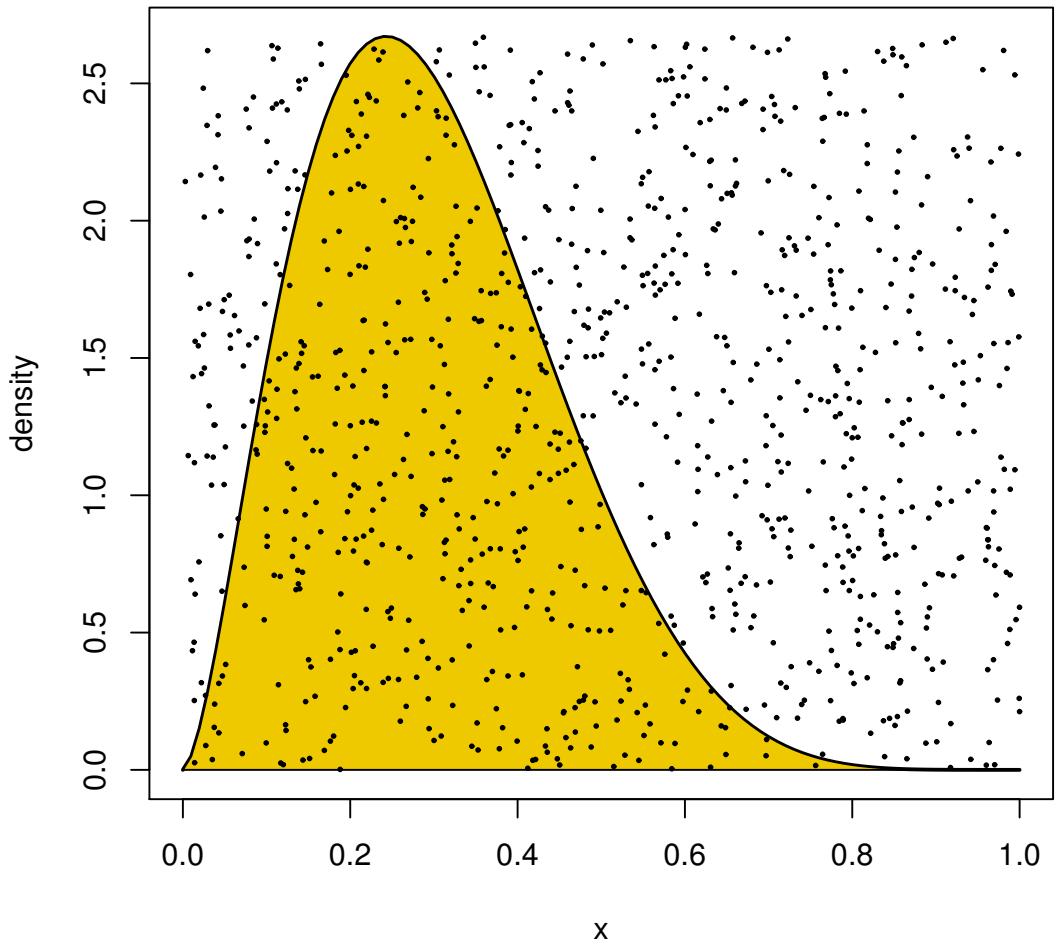
قصد داریم از توزیع $Beta(۲/۷, ۶/۳)$ با روش رد و پذیرش، نمونه تولید کنیم. ابتدا دقت کنید که کران بالا، c، مقدار ماکسیمم تابع چگالی احتمال بتاست که با دستور زیر قابل دستیابی است:

```
c=optimize(f=function(x){dbeta(x,2.7,6.3)},
interval=c(0,1),maximum=T)$objective
```

تابع g را یکنواخت استاندارد در نظر بگیرید. بنابراین مقدار پیشنهادی y پذیرفته می‌شود هرگاه $c \times U < f(y)$.
دقت کنید که تولید $U \sim U(۰, ۱)$ و سپس ضرب آن در c، معادل است با تولید $U \sim U(۰, c)$.

```
ys=runif(1000)
us=c*runif(1000)
val=seq(0,1.,.01)
valf=dbeta(val,shape1=2.7,shape2=6.3)
plot(val,valf,type="l",xlab="x",ylab="density",lwd=2)
polygon(c(val,rev(val)),c(valf,0*valf),col="gold2")
points(ys,us,cex=.4,pch=20)
```

روش رد برای توزیع بتا



مقایسه کارایی روش‌ها

برای تولید تحقق‌هایی از توزیع کای‌دو با ۶ درجه آزادی، دو روش در نظر می‌گیریم:

```
test1=function(){
    U=runif(3*10^4)
    U=matrix(data=U,nrow=3) # matrix for sums
    X=-log(U) # uniform to exponential
    X=2*apply(X,2,sum)
return(X)
}
##
test2=function(){
    X=rchisq(10^4,df=6)
return(X)
}
##
> system.time(test1());system.time(test2())
   user  system elapsed
   0.11    0.00    0.08
   user  system elapsed
   0.00    0.00    0.01
```

تولید متغیر تصادفی کای‌دو غیرمرکزی

در توزیع کای‌دو غیرمرکزی، فرم مشخصی برای تابع چگالی آن وجود ندارد.

تولید متغیر تصادفی کای‌دو غیرمرکزی

در توزیع کای‌دو غیرمرکزی، فرم مشخصی برای تابع چگالی آن وجود ندارد.

برای تولید از این توزیع، چندین روش وجود دارد. یک روش کارا استفاده از روش تبدیل می‌باشد:

$$Z \sim \chi^{۲}_{p-۱},\ Y \sim N(\sqrt{\lambda}, ۱) \longrightarrow Z + Y^{۲} \sim \chi^{۲}_{p}(\lambda).$$

تولید متغیر تصادفی کای‌دو غیرمرکزی

در توزیع کای‌دو غیرمرکزی، فرم مشخصی برای تابع چگالی آن وجود ندارد.

برای تولید از این توزیع، چندین روش وجود دارد. یک روش کارا استفاده از روش تبدیل می‌باشد:

$$Z \sim \chi^2_{p-1},\ Y \sim N(\sqrt{\lambda}, 1) \longrightarrow Z + Y^2 \sim \chi^2_p(\lambda).$$

```
rnchisq = function(n,p,lambda){
z = rchisq(n,df=p-1)
y = rnorm(n,sqrt(lambda),1)
x = z+y^2
return(x)
}
> xx = rnchisq(1000,3,3)
> mean(xx) # theoretical value is (p+lambda)=6
[1] 6.114251
> var(xx) # theoretical value is 2(p+2lambda)=18
[1] 18.25124
```

یک تمرین

تولید یک متغیر تصادفی $N(۰,۱)$ به وسیله روش پذیرش‌ـرد با توزیع نمایی دوگانه به عنوان منتخب با چگالی $g(x|\alpha) = \frac{\alpha}{۲}\exp\{-\alpha|x|\}$ را در نظر بگیرید.

۱. نشان دهید

$$\frac{f(x)}{g(x|\alpha)} \leq \sqrt{\frac{۲}{\pi}}\alpha^{-۱}e^{\frac{\alpha^۲}{۲}},$$

و نشان دهید می‌نیمم این کران بر حسب α در $\alpha = ۱$ به دست می‌آید

۲. نشان دهید احتمال پذیرش برابر $\sqrt{\frac{\pi}{۲e}} = ۰/۷۶$ است و برای تولید یک متغیر تصادفی نرمال به طور متوسط $\frac{۱}{۰/۷۶} = ۱/۳$ تولید متغیرهای یکنواخت لازم است

توزیع‌های آمیخته

متغیر تصادفی X دارای توزیع آمیخته گسسته است، اگر توزیع آن، برای دنباله‌ای از متغیرهای تصادفی $X_۱, X_۲, \cdots$، به صورت مجموع وزنی $F_X(x) = \sum \theta_i F_{X_i}(x)$ باشد به طوری که $\sum \theta_i = ۱$.

توزیع‌های آمیخته

متغیر تصادفی X دارای توزیع آمیخته گسسته است، اگر توزیع آن، برای دنباله‌ای از متغیرهای تصادفی $X_1, X_2, \cdots$، به صورت مجموع وزنی $F_X(x) = \sum \theta_i F_{X_i}(x)$ باشد به طوری که $\sum \theta_i = 1$.

به ثابت‌های θ_i، وزن‌های آمیختگی گویند.

توزیع‌های آمیخته

متغیر تصادفی X دارای توزیع آمیخته گسسته است، اگر توزیع آن، برای دنباله‌ای از متغیرهای تصادفی $X_1, X_2, \cdots$، به صورت مجموع وزنی $F_X(x) = \sum \theta_i F_{X_i}(x)$ باشد به طوری که $\sum \theta_i = 1$.

به ثابت‌های θ_i، وزن‌های آمیختگی گویند.

متغیر تصادفی X دارای توزیع آمیخته پیوسته است، اگر برای یک خانواده $X|Y = y$ که با مقادیر حقیقی y اندیس‌گذاری شده است، توزیع آن به صورت $F_X(x) = \int_{-\infty}^{\infty} F_{X|Y=y}(x) f_Y(y) dy$ باشد، به طوری که در آن f_y تابع وزن‌دهنده است و $\int_{-\infty}^{\infty} f_Y(y) dy = 1$.

توزیع‌های آمیخته

متغیر تصادفی X دارای توزیع آمیخته گسسته است، اگر توزیع آن، برای دنباله‌ای از متغیرهای تصادفی $X_1, X_2, \cdots$، به صورت مجموع وزنی $F_X(x) = \sum \theta_i F_{X_i}(x)$ باشد به طوری که $\sum \theta_i = 1$.

به ثابت‌های θ_i، وزن‌های آمیختگی گویند.

متغیر تصادفی X دارای توزیع آمیخته پیوسته است، اگر برای یک خانواده $X|Y = y$ که با مقادیر حقیقی y اندیس‌گذاری شده است، توزیع آن به صورت $F_X(x) = \int_{-\infty}^{\infty} F_{X|Y=y}(x) f_Y(y) dy$ باشد، به طوری که در آن f_y تابع وزن‌دهنده است و $\int_{-\infty}^{\infty} f_Y(y) dy = 1$.

دقت کنید توزیع‌های آمیخته با توزیع‌های حاصل از پیچش با هم فرق دارند. به مثال زیر توجه کنید.

آمیخته یا پیچش؟

فرض کنید $X_1 \sim N(0,1)$ و $X_2 \sim N(3,1)$ و مستقل از هم باشند. $S = X_1 + X_2$ پیچش دو متغیر را نشان می‌دهد. توزیع S نرمال با میانگین $\mu_1 + \mu_2 = 3$ و واریانس $\sigma_1^2 + \sigma_2^2 = 2$ است. برای تولید این پیچش داریم:

1. x_1 را از توزیع $N(0,1)$ تولید کن
2. x_2 از $N(3,1)$ تولید کن
3. قرار بده $s = x_1 + x_2$

آمیخته یا پیچش؟

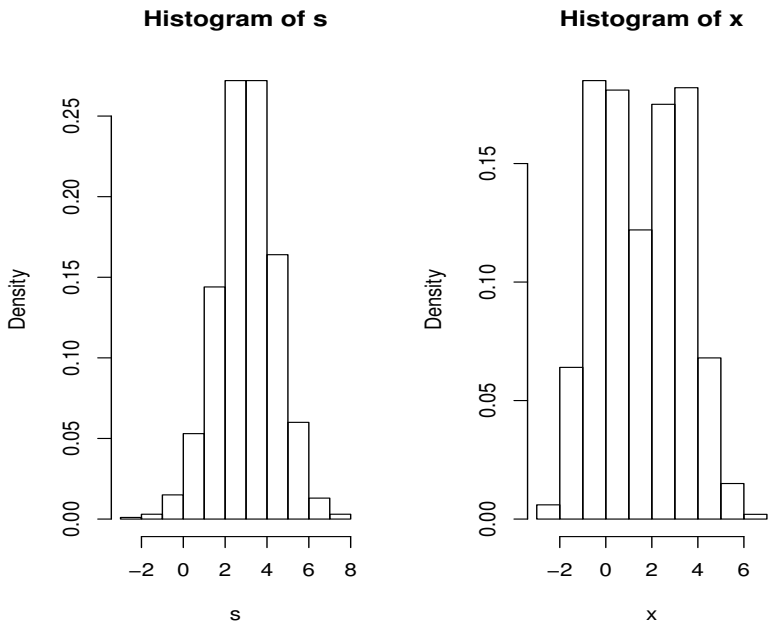
فرض کنید $X_1 \sim N(0,1)$ و $X_2 \sim N(3,1)$ و مستقل از هم باشند. $S = X_1 + X_2$ پیچش دو متغیر را نشان می‌دهد. توزیع S نرمال با میانگین $\mu_1 + \mu_2 = 3$ و واریانس $\sigma_1^2 + \sigma_2^2 = 2$ است. برای تولید این پیچش داریم:

1. x_1 را از توزیع $N(0,1)$ تولید کن
2. x_2 از $N(3,1)$ تولید کن
3. قرار بده $s = x_1 + x_2$

برای ساخت یک آمیخته ۵۰٪، یعنی $F_X(x) = 0.5F_{X_1}(x) + 0.5F_{X_2}(x)$، داریم:

1. عدد صحیح k را از مجموعه $\{1,2\}$ تولید کن به طوری که $P(1) = P(2) = 0.5$
2. اگر $k = 1$، x را از $N(0,1)$ تولید کن و اگر $k = 2$، x را از $N(3,1)$ تولید کن

```
n <- 1000
x1 <- rnorm(n, 0, 1)
x2 <- rnorm(n, 3, 1)
s <- x1 + x2                # the convolution
u <- runif(n)
k <- as.integer(u > 0.5)  # vector of 0's and 1's
x <- k * x1 + (1-k) * x2  # the mixture
par(mfcol=c(1,2))           # two graphs per page
hist(s, prob=TRUE)
hist(x, prob=TRUE)
par(mfcol=c(1,1))           # restore display
```



آمیخته‌ای از چند توزیع

مشابه بالا می‌توان برای بیشتر از دو توزیع هم، شکل آمیخته‌ای را نوشت. مثلاً فرض کنید آمیخته‌ای از ۵ توزیع گاما مورد نظر باشد. بنابراین $F_X = \sum_{i=۱}^{۵} \theta_i F_{X_i}$، به طوری که $X_i \sim Gamma(s = ۳, \lambda_i = ۱/i)$ و مستقل از هم هستند. همچنین وزن‌های آمیختگی برای $i = ۱, \ldots, ۵$ عبارتند از $\theta_i = i/۱۵$.

آمیخته‌ای از چند توزیع

مشابه بالا می‌توان برای بیشتر از دو توزیع هم، شکل آمیخته‌ای را نوشت. مثلاً فرض کنید آمیخته‌ای از ۵ توزیع گاما مورد نظر باشد. بنابراین $F_X = \sum_{i=۱}^{۵} \theta_i F_{X_i}$، به طوری که $X_i \sim Gamma(s = ۳, \lambda_i = ۱/i)$ و مستقل از هم هستند. همچنین وزن‌های آمیختگی برای $i = ۱, \ldots, ۵$ عبارتند از $\theta_i = i/۱۵$.

برای تولید یک متغیر تصادفی با این توزیع، به صورت زیر عمل می‌کنیم:

۱. عدد صحیح $k \in \{۱, ۲, ۳, ۴, ۵\}$ را با احتمال‌های $P(k) = \theta_k$، برای $k = ۱, \ldots, ۵$ تولید می‌کنیم
۲. متغیر $Gamma(s, \lambda_k)$ را به عنوان مقدار تولید شده برمی‌گردانیم

برای تولید n نمونه باید دو مرحله بالا را n بار تکرار کنیم. برای این کار باید از یک حلقه for استفاده شود که می‌تواند برای حجم نمونه بزرگ ناکارا باشد.

الگوریتم بالا را می‌توان به صورت زیر با بردارسازی کاراتر کرد:

۱. نمونه تصادفی $k = (k_1, \ldots, k_n)$ را از اعداد صحیح تولید کن به طوری که $P(k) = \theta_k$. بنابراین $k[i]$ مشخص می‌کند که برای تولید عضو iام نمونه از کدام توزیع گاما استفاده شده است.

۲. بردار $rate$ را با طول n تنظیم کن که حاوی مقادیر λ_k است.

۳. با دستور $rgamma$ با پارامتر شکل ۳ و بردار $rate$ نمونه‌ای به حجم n تولید کن

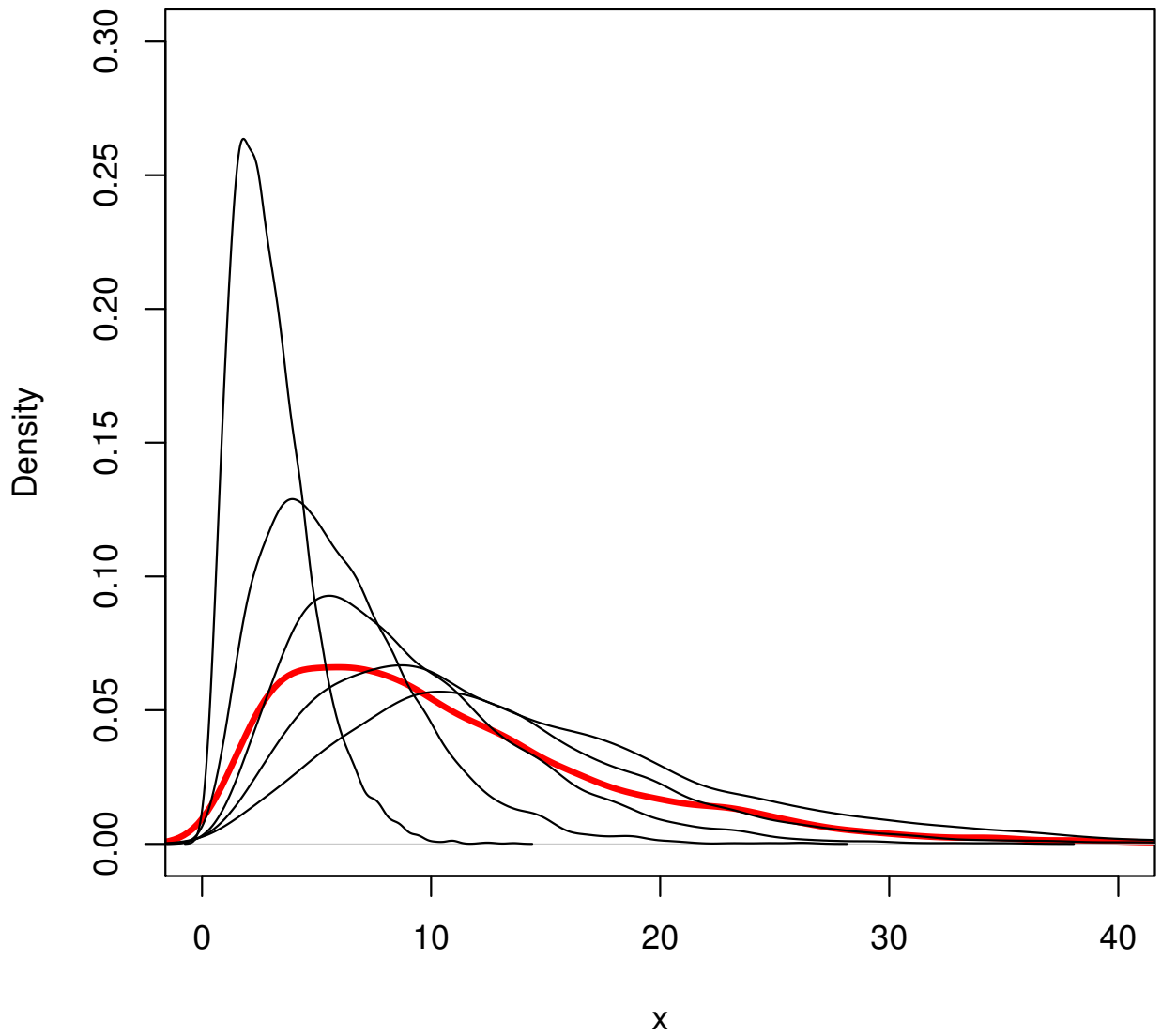
الگوریتم بالا را می‌توان به صورت زیر با بردارسازی کاراتر کرد:

۱. نمونه تصادفی $k = (k_1, \ldots, k_n)$ را از اعداد صحیح تولید کن به طوری که $P(k) = \theta_k$. بنابراین $k[i]$ مشخص می‌کند که برای تولید عضو iام نمونه از کدام توزیع گاما استفاده شده است.

۲. بردار $rate$ را با طول n تنظیم کن که حاوی مقادیر λ_k است.

۳. با دستور $rgamma$ با پارامتر شکل ۳ و بردار $rate$ نمونه‌ای به حجم n تولید کن

```
n <- 5000
k <- sample(1:5, size=n, replace=TRUE, prob=(1:5)/15)
rate <- 1/k
x <- rgamma(n, shape=3, rate=rate)
# plot the densities of the mixture alongside the components
plot(density(x), xlim=c(0,40), ylim=c(0,.3),
lwd=3, xlab="x", col="red", main="")
for (i in 1:5)
    lines(density(rgamma(n, 3, 1/i)))
```

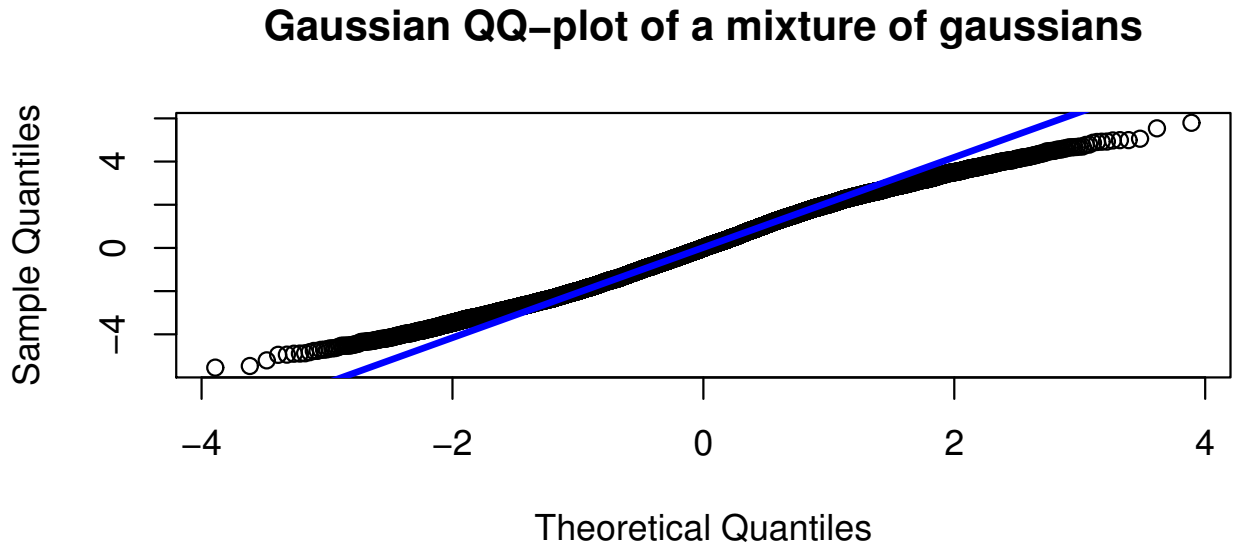

Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
10
20
30
40
x

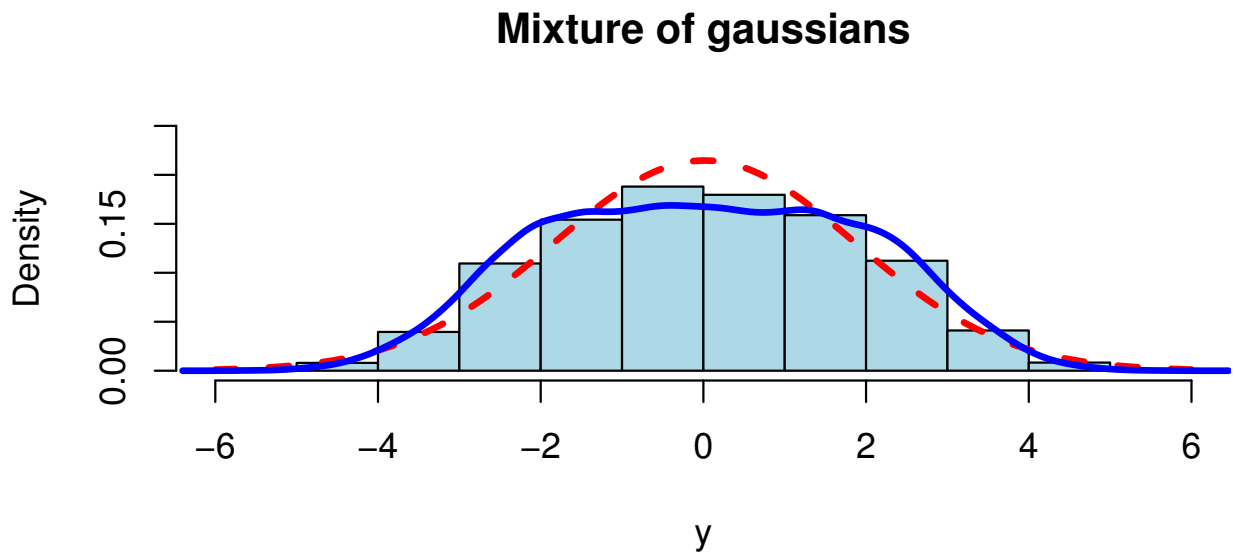
توزیع‌های آمیخته نرمال (متناهی)

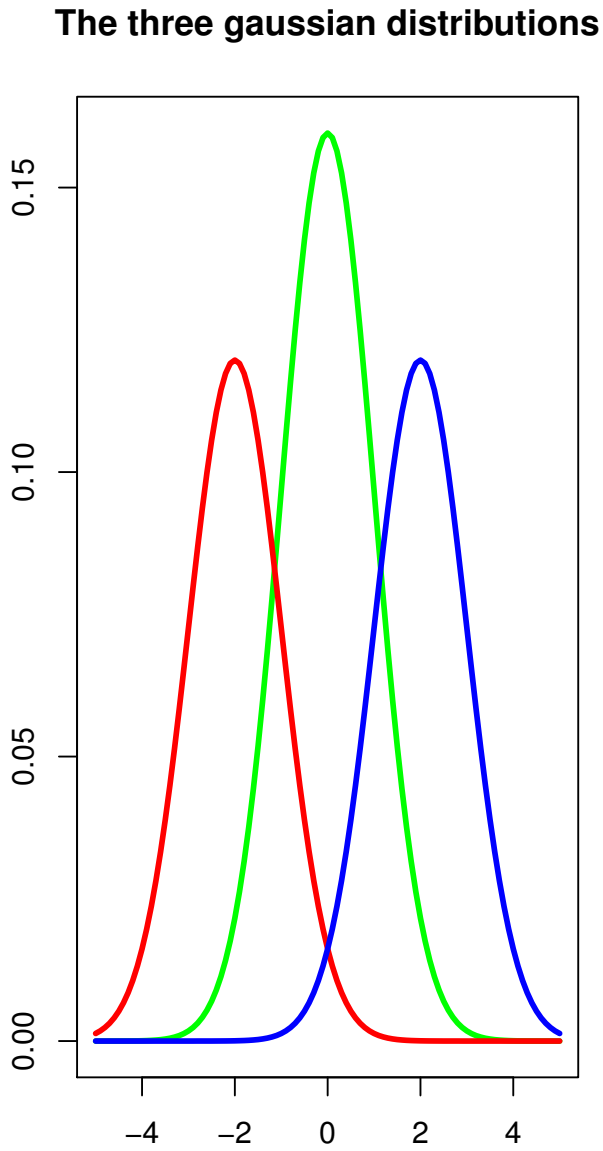
بسیاری از توزیع‌ها را شامل توزیع‌های چوله، با دم‌های پهن و غیره، می‌توان با توزیع‌های آمیخته نرمال تولید کرد:

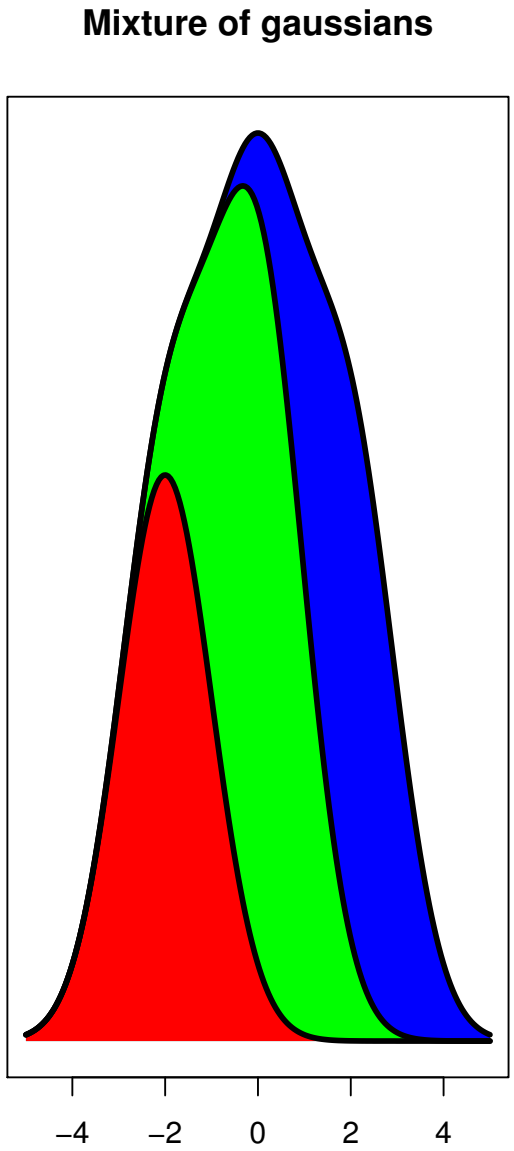
$$F_X(x) = ۰/۳N(-۲, ۱) + ۰/۴N(۰, ۱) + ۰/۳N(۲, ۱)$$

```
n <- 10000
m <- c(-2,0,2)    # Means
p <- c(.3,.4,.3)  # Probabilities
s <- c(1, 1, 1)   # Standard deviations
x <- cbind( rnorm(n, m[1], s[1]),
            rnorm(n, m[2], s[2]),
            rnorm(n, m[3], s[3]) )
a <- sample(1:3, size=n, prob=p, replace=TRUE)
y <- x[ 1:n + n*(a-1) ]
par(mfrow=c(2,1))
qqnorm(y,
       main="Gaussian QQ-plot of a mixture of gaussians")
qqline(y, col="blue", lwd=3)
hist(y, col="light blue", probability=TRUE,
     ylim=c(0,.25),
     main="Mixture of gaussians")
curve(dnorm(x, mean=mean(y), sd=sd(y)),
      add=TRUE, col="red", lwd=3, lty=2)
lines(density(x), col="blue", lwd=3)
```


Gaussian QQ–plot of a mixture of gaussians
Sample Quantiles
Theoretical Quantiles
–4
0
4
–4
–2
0
2
4


Mixture of gaussians
Density
y
0.00
0.15
–6
–4
–2
0
2
4
6


The three gaussian distributions
0.15
0.10
0.05
0.00
−4
−2
0
2
4


Mixture of gaussians
−4
−2
0
2
4

رسم نمودار چگالی در آمیخته‌ها

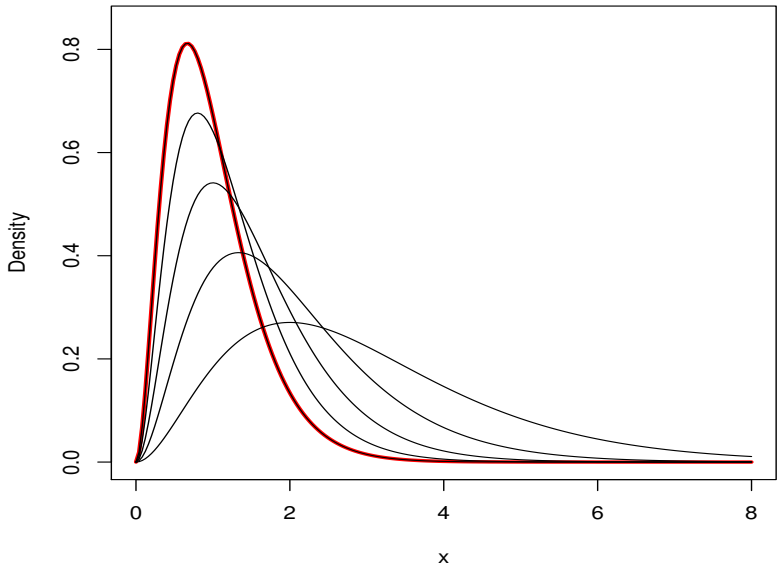
برای رسم تابع چگالی احتمال به عنوان مثال آمیخته گاما، باید ابتدا شکل $f(x) = \sum_{i=1}^{5} \theta_i f_i(x)$ را به عنوان یک تابع در R تعریف کنیم:

```
f <- function(x, lambda, theta) {
        # density of the mixture at the point x
        sum(dgamma(x, 3, lambda) * theta)
    }
```

این تابع مقدار چگالی آمیخته را به ازای هر x محاسبه می‌کند. دقت کنید در تابع $dgamma$، اگر x یک مقدار حقیقی باشد (نه بردار)، نتیجه برداری با طولی برابر طول $lambda$ خواهد بود، یعنی $(f_1(x), \ldots, f_5(x))$. همچنین برای همین تابع ضرب در θ.

```
x <- seq(0, 8, length=200)
dim(x) <- length(x)  # need for apply
# compute density of the mixture f(x) along x
y <- apply(x, 1, f, lambda=lambda, theta=p)
```

```
#plot the density of the mixture
plot(x, y, type="l", ylim=c(0,.85), lwd=3, col="red",
     ylab="Density")
for (j in 1:5) {
        # add the j-th gamma density to the plot
        y = apply(x, 1, dgamma, shape=3, rate=lambda[j])
        lines(x, y)
    }
```



تولید آمیخته پیوسته: آمیخته پواسون_ گاما

توزیع دوجمله‌ای منفی، یک آمیخته از توزیع‌های پواسون $Poisson(\Lambda)$ است که در آن Λ دارای توزیع گاما است.

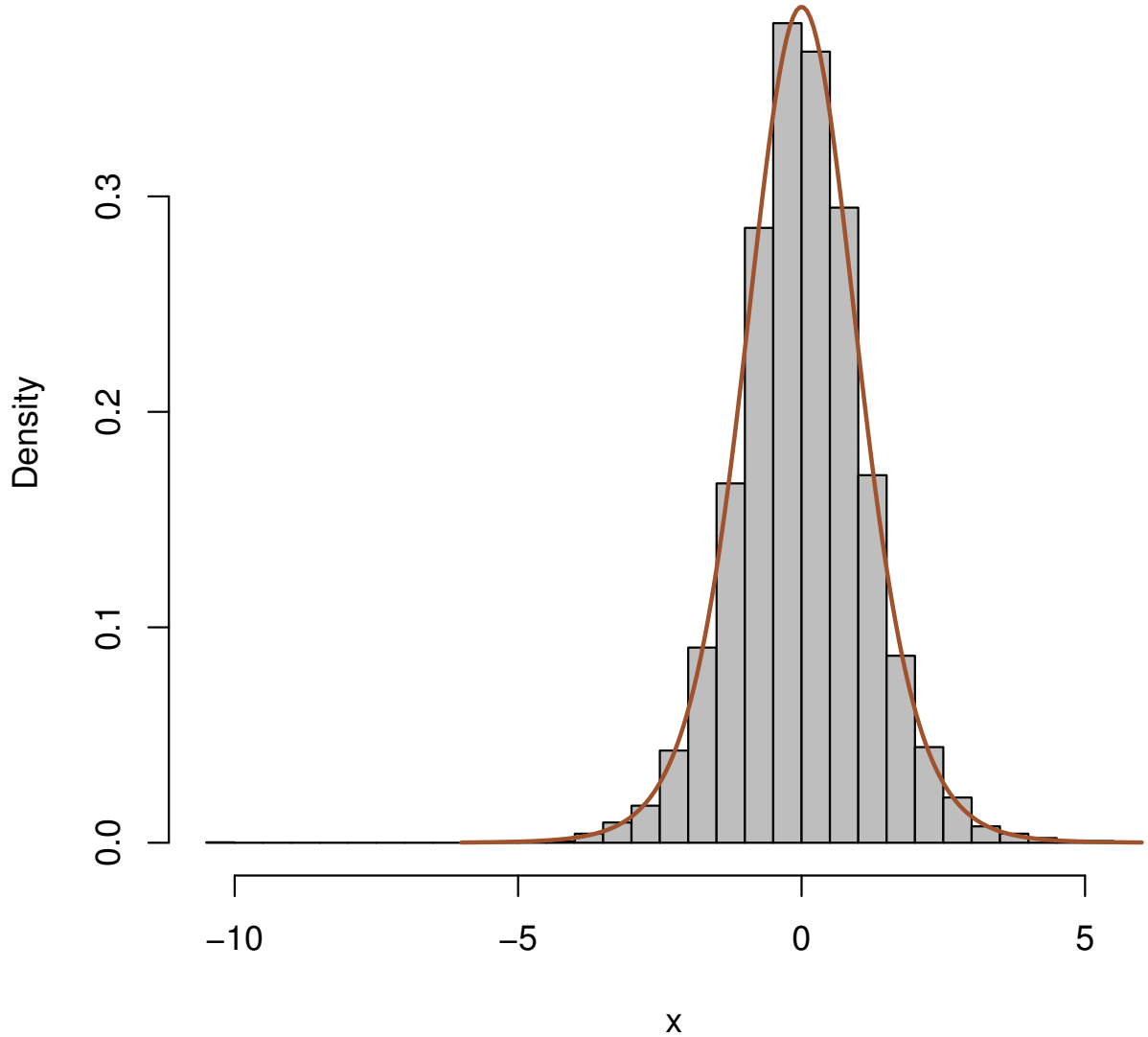
$$X|\Lambda = \lambda \sim Piosson(\lambda);\ \Lambda \sim Gamma(r, \beta),$$

$$X \sim NB(r, p = \frac{\beta}{۱ + \beta}).$$

```
# generate a Poisson-Gamma mixture
n <- 1000
r <- 4
beta <- 3
lambda <- rgamma(n, r, beta) #lambda is random
# now supply the sample of lambda's as the Poisson mean
x <- rpois(n, lambda)        #the mixture
# compare with negative binomial
mix <- tabulate(x+1) / n
negbin <- round(dnbinom(0:max(x), r, beta/(1+beta)), 3)
se <- sqrt(negbin * (1 - negbin) / n)
> round(rbind(mix, negbin, se), 3)
        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
mix    0.316 0.328 0.196 0.087 0.048 0.014 0.008 0.003
negbin 0.316 0.316 0.198 0.099 0.043 0.017 0.006 0.002
se     0.015 0.015 0.013 0.009 0.006 0.004 0.002 0.001
```

$$X|y \sim N(۰, \nu/y) \ \& \ Y \sim \chi^{۲}_{\nu} \longrightarrow X \sim T_{\nu}.$$

```
Nsim=10^4
nu=9
y=rchisq(Nsim,df=nu)
x=rnorm(Nsim,0,sqrt(nu/y))
hist(x,main="",freq=F,col="grey",breaks=40)
z = seq(-6,6,length.out=200)
lines(z,dt(z,nu),lwd=2,col="sienna")
> mean(x) # the mean of a t variable is 0
[1] 0.007723547
> var(x) # the variance of a t-variable is nu/nu-2=1.2857
[1] 1.294373
```


Density
0.0
0.1
0.2
0.3
−10
−5
0
5
x

توزیع‌های چندمتغیره: نرمال چندمتغیره

بردار $\mathbf{X} = (X_1, \ldots, X_d)$ دارای توزیع نرمال d متغیره است و با نماد $N_d(\mu, \Sigma)$ نمایش می‌دهند، هرگاه تابع چگالی احتمال آن به صورت زیر باشد:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\},\ x \in \Re^d$$

به طوری که $\mu = (\mu_1, \ldots, \mu_d)^T$ بردار میانگین و $\Sigma = (\sigma_{ij})$ ماتریس $d \times d$ متقارن و معین مثبت با درایه‌های $\sigma_{ij} = Cov(X_i, X_j)$ است.

توزیع‌های چندمتغیره: نرمال چندمتغیره

بردار $\mathbf{X} = (X_1, \ldots, X_d)$ دارای توزیع نرمال d متغیره است و با نماد $N_d(\mu, \Sigma)$ نمایش می‌دهند، هرگاه تابع چگالی احتمال آن به صورت زیر باشد:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\},\ x \in \Re^d$$

به طوری که $\mu = (\mu_1, \ldots, \mu_d)^T$ بردار میانگین و $\Sigma = (\sigma_{ij})$ ماتریس $d \times d$ متقارن و معین مثبت با درایه‌های $\sigma_{ij} = Cov(X_i, X_j)$ است.

ماتریس Σ^{-1} که معکوس ماتریس کوواریانس است، معروف به ماتریس دقت می‌باشد.

توزیع‌های چندمتغیره: نرمال چندمتغیره

بردار $\mathbf{X} = (X_1, \ldots, X_d)$ دارای توزیع نرمال d متغیره است و با نماد $N_d(\mu, \Sigma)$ نمایش می‌دهند، هرگاه تابع چگالی احتمال آن به صورت زیر باشد:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\},\ x \in \Re^d$$

به طوری که $\mu = (\mu_1, \ldots, \mu_d)^T$ بردار میانگین و $\Sigma = (\sigma_{ij})$ ماتریس $d \times d$ متقارن و معین مثبت با درایه‌های $\sigma_{ij} = Cov(X_i, X_j)$ است.

ماتریس Σ^{-1} که معکوس ماتریس کوواریانس است، معروف به ماتریس دقت می‌باشد.

یک متغیر نرمال چندمتغیره را می‌توان در دو مرحله تولید کرد:

1. ابتدا تولید یک نمونه تصادفی ساده $\mathbf{Z} = (Z_1, \ldots, Z_d)$ از توزیع نرمال استاندارد
2. تبدیل بردار $\mathbf{Z}$ به طوری که دارای بردار میانگین و ماتریس کوواریانس مورد نظر باشد

توزیع‌های چندمتغیره: نرمال چندمتغیره

بردار $\mathbf{X} = (X_1, \ldots, X_d)$ دارای توزیع نرمال d متغیره است و با نماد $N_d(\mu, \Sigma)$ نمایش می‌دهند، هرگاه تابع چگالی احتمال آن به صورت زیر باشد:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\},\ x \in \Re^d$$

به طوری که $\mu = (\mu_1, \ldots, \mu_d)^T$ بردار میانگین و $\Sigma = (\sigma_{ij})$ ماتریس $d \times d$ متقارن و معین مثبت با درایه‌های $\sigma_{ij} = Cov(X_i, X_j)$ است.

ماتریس Σ^{-1} که معکوس ماتریس کوواریانس است، معروف به ماتریس دقت می‌باشد.

یک متغیر نرمال چندمتغیره را می‌توان در دو مرحله تولید کرد:

1. ابتدا تولید یک نمونه تصادفی ساده $\mathbf{Z} = (Z_1, \ldots, Z_d)$ از توزیع نرمال استاندارد
2. تبدیل بردار $\mathbf{Z}$ به طوری که دارای بردار میانگین و ماتریس کوواریانس مورد نظر باشد

تبدیل مورد نظر نیازمند تجزیه ماتریس کوواریانس است

$$
\begin{aligned}
Z &\sim N_d(\mu, \Sigma) \longrightarrow CZ + b \sim N_d(C\mu + b, C\Sigma C^T) \\
Z &\sim N_d(0, I_d) \longrightarrow CZ + b \sim N_d(b, CC^T).
\end{aligned}
$$

حال فرض کنید بتوانیم ماتریس کوواریانس را برای یک ماتریس C به صورت $\Sigma = CC^T$ تجزیه کنیم. پس:

$$CZ + \mu \sim N_d(\mu, \Sigma).$$

$$
\begin{aligned}
Z &\sim N_d(\mu, \Sigma) \longrightarrow CZ + b \sim N_d(C\mu + b, C\Sigma C^T) \\
Z &\sim N_d(0, I_d) \longrightarrow CZ + b \sim N_d(b, CC^T).
\end{aligned}
$$

حال فرض کنید بتوانیم ماتریس کوواریانس را برای یک ماتریس C به صورت $\Sigma = CC^T$ تجزیه کنیم. پس:

$$CZ + \mu \sim N_d(\mu, \Sigma).$$

تجزیه مورد نیاز برای Σ را می‌توان به وسیله روش‌های تجزیه طیفی، تجزیه چولسکی ($Choleski$)، یا تجزیه مقدار ویژه (SVD) انجام داد.

$$
\begin{aligned}
Z &\sim N_d(\mu, \Sigma) \longrightarrow CZ + b \sim N_d(C\mu + b, C\Sigma C^T) \\
Z &\sim N_d(0, I_d) \longrightarrow CZ + b \sim N_d(b, CC^T).
\end{aligned}
$$

حال فرض کنید بتوانیم ماتریس کوواریانس را برای یک ماتریس C به صورت $\Sigma = CC^T$ تجزیه کنیم. پس:

$$CZ + \mu \sim N_d(\mu, \Sigma).$$

تجزیه مورد نیاز برای Σ را می‌توان به وسیله روش‌های تجزیه طیفی، تجزیه چولسکی ($Choleski$)، یا تجزیه مقدار ویژه (SVD) انجام داد.

توابع متناظر آن‌ها در R عبارتند از $eigen$، $chol$، و svd.

$$
\begin{aligned}
Z &\sim N_d(\mu, \Sigma) \longrightarrow CZ + b \sim N_d(C\mu + b, C\Sigma C^T) \\
Z &\sim N_d(0, I_d) \longrightarrow CZ + b \sim N_d(b, CC^T).
\end{aligned}
$$

حال فرض کنید بتوانیم ماتریس کوواریانس را برای یک ماتریس C به صورت $\Sigma = CC^T$ تجزیه کنیم. پس:

$$CZ + \mu \sim N_d(\mu, \Sigma).$$

تجزیه مورد نیاز برای Σ را می‌توان به وسیله روش‌های تجزیه طیفی، تجزیه چولسکی $(Choleski)$، یا تجزیه مقدار ویژه (SVD) انجام داد.

توابع متناظر آن‌ها در R عبارتند از $eigen$، $chol$، و svd.

در حالت ماتریسی، فرض کنید $Z = (Z_{ij})$ یک ماتریس $n \times d$ باشد که Z_{ij} ها نمونه تصادفی ساده از نرمال استاندارد هستند. در این حالت

$$X = ZQ + J\mu^T,$$

که در آن $Q^T Q = \Sigma$ و J یک بردار از ۱ ها می‌باشد.

تولید نمونه‌هایی از نرمال چندمتغیره

برای تولید یک نمونه n تایی از $N_d(\mu, \Sigma)$ باید:

1. یک ماتریس $n \times d$، Z، شامل nd متغیر نرمال استاندارد تولید کنیم
2. تجزیه $\Sigma = Q^T Q$ را انجام دهیم
3. تبدیل $X = ZQ + J\mu^T$ را اجرا کنیم
4. ماتریس X با بعد $n \times d$ را که سطرهای آن نمونه‌های یک نرمال چندمتغیره است را برگردانیم

تبدیل $X = ZQ + J\mu^T$ را در R می‌توان به صورت زیر اجرا کرد:

```
Z = matrix(rnorm(n*d), nrow = n, ncol = d)
X = Z%*%Q+matrix(mu, n, d, byrow=TRUE)
```

تجزیه ماتریس کوواریانس به روش طیفی

ریشه دوم ماتریس کوواریانس عبارتست از $\Sigma^{1/2} = P\Lambda^{1/2}P^{-1}$ که در آن Λ ماتریس قطری با مقادیر ویژه ماتریس Σ است و P ماتریسی است که ستون‌های آن بردارهای ویژه Σ متناظر با مقادیر ویژه آن است.

تجزیه ماتریس کوواریانس به روش طیفی

ریشه دوم ماتریس کوواریانس عبارتست از $\Sigma^{1/2} = P\Lambda^{1/2}P^{-1}$ که در آن Λ ماتریس قطری با مقادیر ویژه ماتریس Σ است و P ماتریسی است که ستون‌های آن بردارهای ویژه Σ متناظر با مقادیر ویژه آن است.

در این تجزیه داریم $P^{-1} = P^T$ و بنابراین $\Sigma^{1/2} = P\Lambda^{1/2}P^T$

تجزیه ماتریس کوواریانس به روش طیفی

ریشه دوم ماتریس کوواریانس عبارتست از $\Sigma^{1/2} = P\Lambda^{1/2}P^{-1}$ که در آن Λ ماتریس قطری با مقادیر ویژه ماتریس Σ است و P ماتریسی است که ستون‌های آن بردارهای ویژه Σ متناظر با مقادیر ویژه آن است.

در این تجزیه داریم $P^{-1} = P^T$ و بنابراین $\Sigma^{1/2} = P\Lambda^{1/2}P^T$

بنابراین در این تجزیه، $Q = \Sigma^{1/2}$.

تجزیه ماتریس کوواریانس به روش طیفی

ریشه دوم ماتریس کوواریانس عبارتست از $\Sigma^{1/2} = P\Lambda^{1/2}P^{-1}$ که در آن Λ ماتریس قطری با مقادیر ویژه ماتریس Σ است و P ماتریسی است که ستون‌های آن بردارهای ویژه Σ متناظر با مقادیر ویژه آن است.

در این تجزیه داریم $P^{-1} = P^T$ و بنابراین $\Sigma^{1/2} = P\Lambda^{1/2}P^T$

بنابراین در این تجزیه، $Q = \Sigma^{1/2}$.

به عنوان مثال تولید نمونه از توزیع نرمال دومتغیره با بردار میانگین صفر و ماتریس کوواریانس

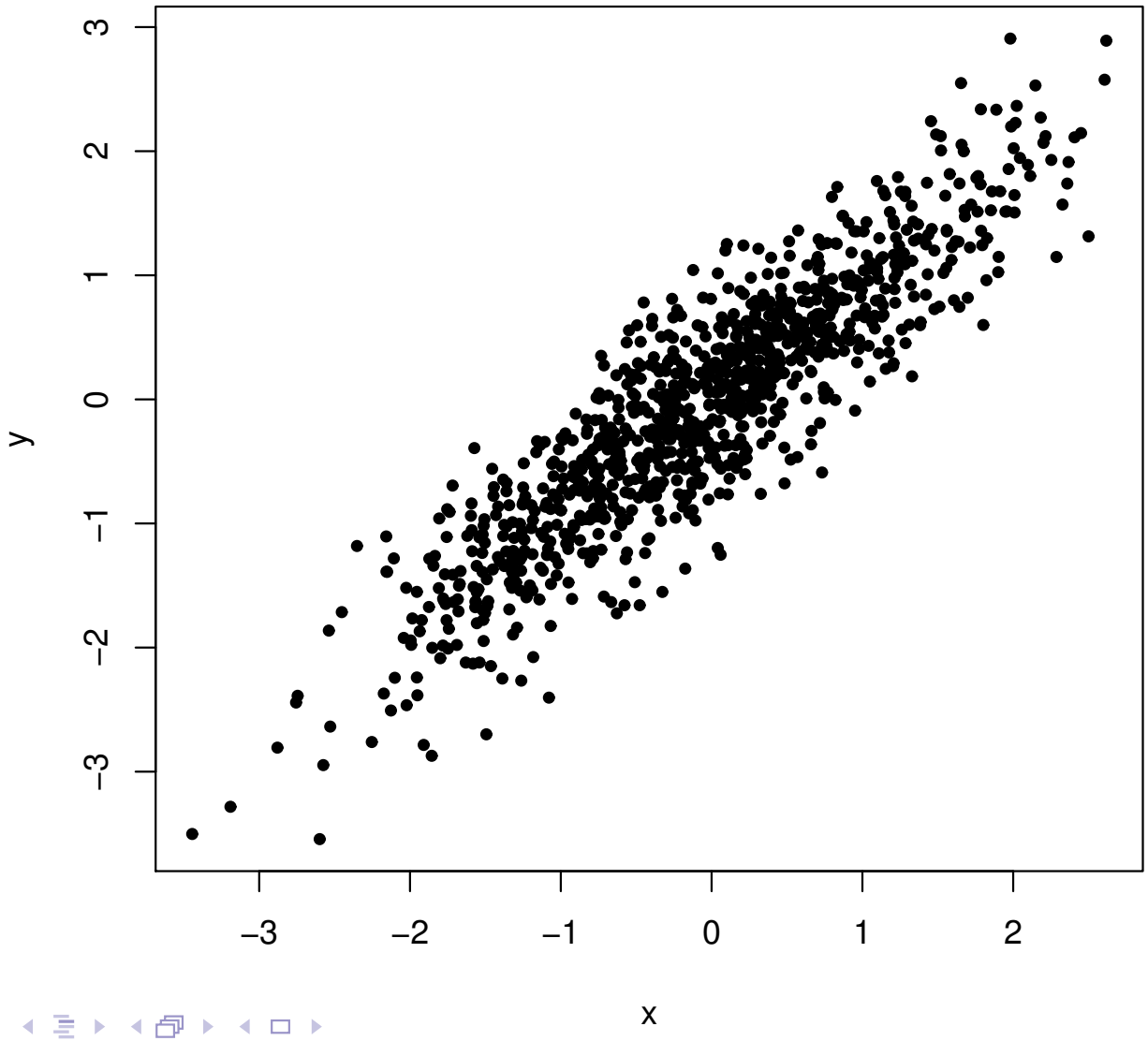
$$\Sigma = \begin{bmatrix} ۱/۰ & ۰/۹ \\ ۰/۹ & ۱/۰ \end{bmatrix}$$

را در نظر بگیرید

```
# mean and covariance parameters
mu <- c(0, 0)
Sigma <- matrix(c(1, .9, .9, 1), nrow = 2, ncol = 2)
```

```
rmvn.eigen = function(n, mu, Sigma) {
        # generate n random vectors from MVN(mu, Sigma)
        # dimension is inferred from mu and Sigma
        d = length(mu)
        ev = eigen(Sigma, symmetric = TRUE)
        lambda = ev$values
        V = ev$vectors
        R = V %*% diag(sqrt(lambda)) %*% t(V)
        Z = matrix(rnorm(n*d), nrow = n, ncol = d)
        X = Z %*% R + matrix(mu, n, d, byrow = TRUE)
        X
    }
##
X = rmvn.eigen(1000, mu, Sigma)
>     print(colMeans(X))
[1] -0.06803006 -0.06935001
>     print(cor(X))
           [,1]      [,2]
[1,] 1.0000000 0.8969038
[2,] 0.8969038 1.0000000
```

```
plot(X, xlab = "x", ylab = "y", pch = 20)
```



تجزیه ماتریس به روش SVD

تجزیه مقدار ویژه ($singular\ value$)، ایده بردارهای ویژه را به ماتریس‌های مستطیلی تعمیم می‌دهد. این تجزیه برای ماتریس X به صورت زیر است:

$$X = UDV^T,$$

که در آن D برداری شامل مقادیر ویژه X، U ماتریسی با ستون‌های شامل بردارهای ویژه چپ X و V ماتریسی با ستون‌های شامل بردارهای ویژه راست ماتریس X است.

تجزیه ماتریس به روش SVD

تجزیه مقدار ویژه ($singular\ value$)، ایده بردارهای ویژه را به ماتریس‌های مستطیلی تعمیم می‌دهد. این تجزیه برای ماتریس X به صورت زیر است:

$$X = UDV^T,$$

که در آن D برداری شامل مقادیر ویژه X، U ماتریسی با ستون‌های شامل بردارهای ویژه چپ X و V ماتریسی با ستون‌های شامل بردارهای ویژه راست ماتریس X است.

ماتریس مورد علاقه ما در اینجا Σ است که برای آن $UV^T = I$. تجزیه SVD یک ماتریس متقارن معین مثبت، رابطه $U = V = P$ را نتیجه می‌دهد. بنابراین $\Sigma^{1/2} = UD^{1/2}V^T$.

تجزیه ماتریس به روش SVD

تجزیه مقدار ویژه ($singular\ value$)، ایده بردارهای ویژه را به ماتریس‌های مستطیلی تعمیم می‌دهد. این تجزیه برای ماتریس X به صورت زیر است:

$$X = UDV^T,$$

که در آن D برداری شامل مقادیر ویژه X، U ماتریسی با ستون‌های شامل بردارهای ویژه چپ X و V ماتریسی با ستون‌های شامل بردارهای ویژه راست ماتریس X است.

ماتریس مورد علاقه ما در اینجا Σ است که برای آن $UV^T = I$. تجزیه SVD یک ماتریس متقارن معین مثبت، رابطه $U = V = P$ را نتیجه می‌دهد. بنابراین $\Sigma^{1/2} = UD^{1/2}V^T$.

برای این کاربرد، روش تجزیه SVD با تجزیه طیفی یکی است، اما از کارایی کمتری نسبت به تجزیه طیفی برخوردار است. زیرا تجزیه SVD از این اطلاع که ماتریس کوواریانس متقارن مربعی است، استفاده نمی‌کند.

```
rmvn.svd = function(n, mu, Sigma) {
        # generate n random vectors from MVN(mu, Sigma)
        # dimension is inferred from mu and Sigma
        d = length(mu)
        S = svd(Sigma)
        R = S$u %*% diag(sqrt(S$d)) %*% t(S$v)
        Z = matrix(rnorm(n*d), nrow=n, ncol=d)
        X = Z %*% R + matrix(mu, n, d, byrow=TRUE)
        X
    }
```

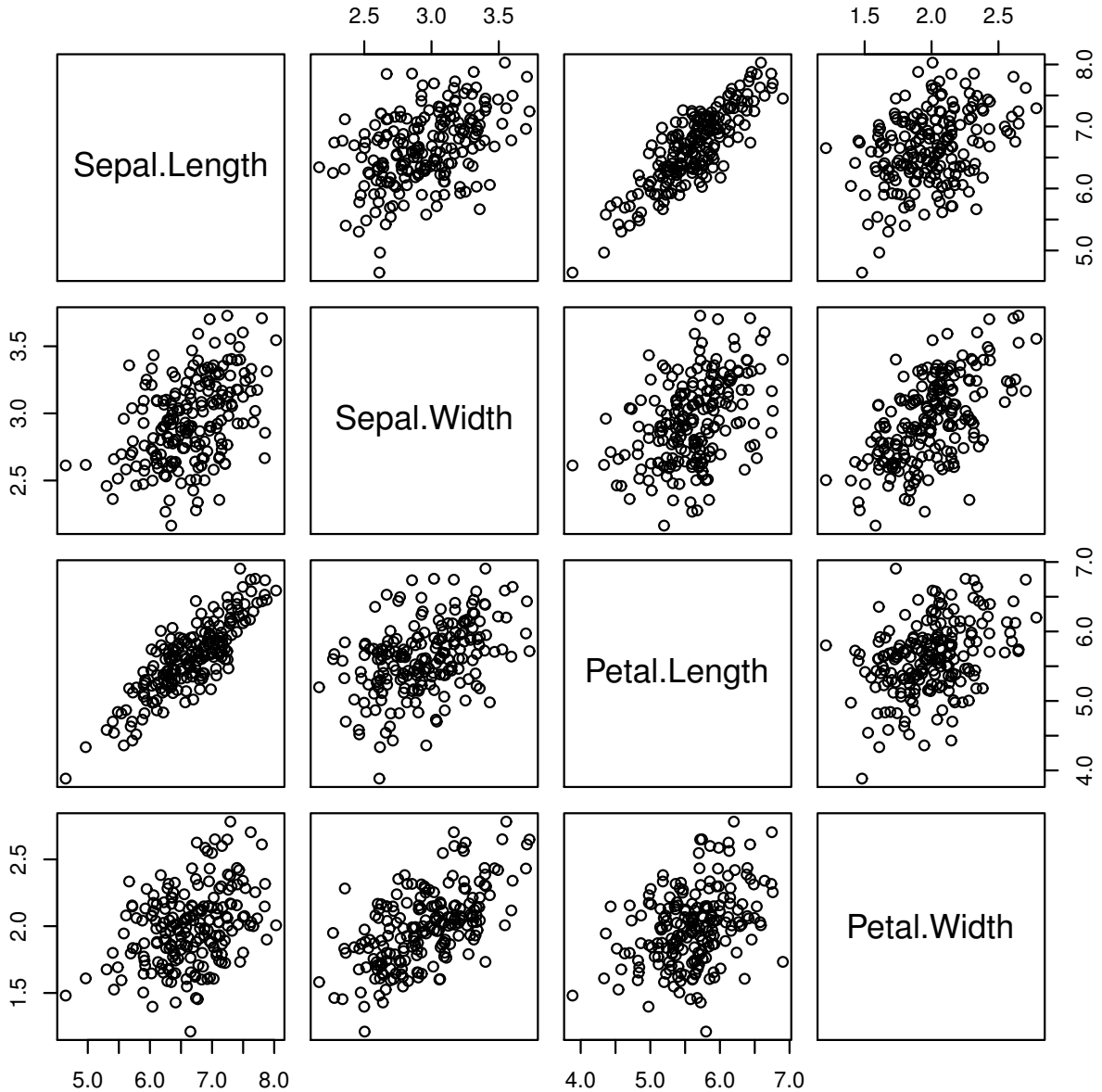
تجزیه به روش چولسکی

تجزیه چولسکی یک ماتریس متقارن معین مثبت به صورت $X = Q^T Q$ است که در آن، Q یک ماتریس بالا مثلثی است. تابع $chol(X)$ در R این تجزیه را انجام می‌دهد و خروجی آن یک ماتریس بالا مثلثی مانند S است به طوری که $S^T S = \Sigma$.

```
rmvn.Choleski = function(n, mu, Sigma) {
        # generate n random vectors from MVN(mu, Sigma)
        # dimension is inferred from mu and Sigma
        d = length(mu)
        Q = chol(Sigma) # Choleski factorization of Sigma
        Z = matrix(rnorm(n*d), nrow=n, ncol=d)
        X = Z %*% Q + matrix(mu, n, d, byrow=TRUE)
        X
    }
```

تجزیه چولسکی برای داده‌های *iris*

```
y <- subset(x=iris, Species=="virginica")[, 1:4]
mu <- colMeans(y)
Sigma <- cov(y)
>      mu
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026
>      Sigma
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length   0.40434286  0.09376327   0.30328980  0.04909388
Sepal.Width    0.09376327  0.10400408   0.07137959  0.04762857
Petal.Length   0.30328980  0.07137959   0.30458776  0.04882449
Petal.Width    0.04909388  0.04762857   0.04882449  0.07543265
X <- rmvn.Choleski(200, mu, Sigma)
pairs(X)
```


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.5
3.0
3.5
1.5
2.0
2.5
5.0
6.0
7.0
8.0
2.5
3.0
3.5
4.0
5.0
6.0
7.0
1.5
2.0
2.5
5.0
6.0
7.0
8.0
4.0
5.0
6.0
7.0

مقایسه روش‌های تجزیه

یک سوال که در اینجا ممکن است مطرح شود، آن است که کدام یک از این روش‌ها ارجح است؟

مقایسه روش‌های تجزیه

یک سوال که در اینجا ممکن است مطرح شود، آن است که کدام یک از این روش‌ها ارجح است؟

یک معیار برای جواب دادن به این سوال مقایسه سرعت این چند روش است.

مقایسه روش‌های تجزیه

یک سوال که در اینجا ممکن است مطرح شود، آن است که کدام یک از این روش‌ها ارجح است؟

یک معیار برای جواب دادن به این سوال مقایسه سرعت این چند روش است.

این سه روش را با دو تابع $mvrnorm$ در بسته $MASS$ و $rmvnorm$ در بسته $mvtnorm$ مقایسه می‌کنیم.

مقایسه روش‌های تجزیه

یک سوال که در اینجا ممکن است مطرح شود، آن است که کدام یک از این روش‌ها ارجح است؟

یک معیار برای جواب دادن به این سوال مقایسه سرعت این چند روش است.

این سه روش را با دو تابع $mvrnorm$ در بسته $MASS$ و $rmvnorm$ در بسته $mvtnorm$ مقایسه می‌کنیم.

تابع $mvrnorm$ بر اساس تجزیه طیفی عمل می‌کند.

مقایسه روش‌های تجزیه

یک سوال که در اینجا ممکن است مطرح شود، آن است که کدام یک از این روش‌ها ارجح است؟

یک معیار برای جواب دادن به این سوال مقایسه سرعت این چند روش است.

این سه روش را با دو تابع $mvrnorm$ در بسته $MASS$ و $rmvnorm$ در بسته $mvtnorm$ مقایسه می‌کنیم.

تابع $mvrnorm$ بر اساس تجزیه طیفی عمل می‌کند.

عنوان شده است که: اگرچه تجزیه چولسکی سریعتر است، تجزیه طیفی پایدارتر است.

```
library(MASS)
library(mvtnorm)
n = 100 # sample size
d <- 30 # dimension
N <- 2000 # iterations
mu <- numeric(d)
##
> set.seed(100)
>     system.time(for (i in 1:N)
+         rmvn.eigen(n, mu, cov(matrix(rnorm(n*d), n, d))))
   user  system elapsed
   4.02    0.00    4.07
>     set.seed(100)
>     system.time(for (i in 1:N)
+         rmvn.svd(n, mu, cov(matrix(rnorm(n*d), n, d))))
   user  system elapsed
   5.05    0.00    5.12
>     set.seed(100)
>     system.time(for (i in 1:N)
+         rmvn.Choleski(n, mu, cov(matrix(rnorm(n*d), n, d))))
   user  system elapsed
   3.74    0.02    3.76
>     set.seed(100)
>     system.time(for (i in 1:N)
+         mvrnorm(n, mu, cov(matrix(rnorm(n*d), n, d))))
   user  system elapsed
   3.88    0.02    3.93
>     set.seed(100)
>     system.time(for (i in 1:N)
+         rmvnorm(n, mu, cov(matrix(rnorm(n*d), n, d))))
   user  system elapsed
   4.82    0.00    4.87
```

توزیع‌های آمیخته نرمال چندمتغیره

توزیع نرمال آمیخته با دو مولفه:

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2),$$

با تغییر p، به عنوان پارامتر آمیختگی، توزیع‌های متفاوت با شکل‌ها و خواص متفاوتی خواهیم داشت.

توزیع‌های آمیخته نرمال چندمتغیره

توزیع نرمال آمیخته با دو مولفه:

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2),$$

با تغییر p، به عنوان پارامتر آمیختگی، توزیع‌های متفاوت با شکل‌ها و خواص متفاوتی خواهیم داشت.

به عنوان مثال یک آمیخته ٪۵۰ توزیعی متقارن با دم‌های سبک است، در حالی که یک آمیخته ٪۹۰ چوله با دم‌های سنگین است.

توزیع‌های آمیخته نرمال چندمتغیره

توزیع نرمال آمیخته با دو مولفه:

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2),$$

با تغییر p، به عنوان پارامتر آمیختگی، توزیع‌های متفاوت با شکل‌ها و خواص متفاوتی خواهیم داشت.

به عنوان مثال یک آمیخته ٪۵۰ توزیعی متقارن با دم‌های سبک است، در حالی که یک آمیخته ٪۹۰ چوله با دم‌های سنگین است.

با توجه به این ویژگی توزیع‌های آمیخته نرمال، از آن‌ها در استنباط‌های تنومند زیاد استفاده می‌شود.

توزیع‌های آمیخته نرمال چندمتغیره

توزیع نرمال آمیخته با دو مولفه:

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2),$$

با تغییر p، به عنوان پارامتر آمیختگی، توزیع‌های متفاوت با شکل‌ها و خواص متفاوتی خواهیم داشت.

به عنوان مثال یک آمیخته ٪۵۰ توزیعی متقارن با دم‌های سبک است، در حالی که یک آمیخته ٪۹۰ چوله با دم‌های سنگین است.

با توجه به این ویژگی توزیع‌های آمیخته نرمال، از آن‌ها در استنباط‌های تنومند زیاد استفاده می‌شود.

از چنین توزیعی می‌توان به طریق زیر، نمونه تولید کرد:

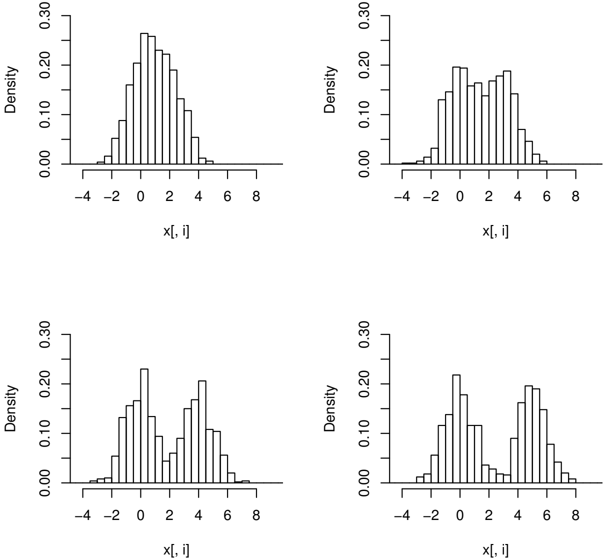
1. تولید $U \sim Uniform(0,1)$
2. اگر $U \leq p$، X را از $N_d(\mu_1, \Sigma_1)$ تولید کن، در غیر این صورت از $N_d(\mu_2, \Sigma_2)$ تولید کن

به روش زیر هم می‌توان عمل کرد:

1. N را از توزیع $Ber(p)$ تولید کن
2. اگر $N = ۱$، X را از $N_d(\mu_۱, \Sigma_۱)$ تولید کن در غیر این صورت از $N_d(\mu_۲, \Sigma_۲)$

```
library(MASS)  #for mvrnorm
# ineffecient version loc.mix.0 with loops
loc.mix.0 <- function(n, p, mu1, mu2, Sigma) {
        # generate sample from BVN location mixture
        X <- matrix(0, n, 2)
        for (i in 1:n) {
            k <- rbinom(1, size = 1, prob = p)
            if (k)
                X[i,] <- mvrnorm(1, mu = mu1, Sigma) else
                X[i,] <- mvrnorm(1, mu = mu2, Sigma)
            }
        return(X)
    }
```

```
#more efficient version
    loc.mix <- function(n, p, mu1, mu2, Sigma) {
        #generate sample from BVN location mixture
        n1 <- rbinom(1, size = n, prob = p)
        n2 <- n - n1
        x1 <- mvrnorm(n1, mu = mu1, Sigma)
        x2 <- mvrnorm(n2, mu = mu2, Sigma)
        X <- rbind(x1, x2)              #combine the samples
        return(X[sample(1:n), ])        #mix them
    }
x <- loc.mix(1000, .5, rep(0, 4), 2:5, Sigma = diag(4))
r <- range(x) * 1.2
par(mfrow = c(2, 2))
for (i in 1:4)
        hist(x[ , i], xlim = r, ylim = c(0, .3), freq = FALSE,
        main = "", breaks = seq(-5, 10, .5))
```


Density
x[, i]
0.00
0.10
0.20
0.30
−4
−2
0
2
4
6
8

توزیع ویشارت

اگر سطرهای ماتریس X $n \times d$ یک نمونه تصادفی ساده از توزیع $N_d(\cdot, \Sigma)$ باشد، آن‌گاه $M = X^T X \sim Wishart_d(\Sigma, n)$.

توزیع ویشارت

اگر سطرهای ماتریس X $n \times d$ یک نمونه تصادفی ساده از توزیع $N_d(۰, \Sigma)$ باشد، آنگاه $M = X^T X \sim Wishart_d(\Sigma, n)$.

به عنوان مثال ماتریس کوواریانس نمونه (S) دارای توزیع ویشارت است: $S \sim Wishart_d(\Sigma/n - ۱, n - ۱)$.

توزیع ویشارت

اگر سطرهای ماتریس X $n \times d$ یک نمونه تصادفی ساده از توزیع $N_d(۰, \Sigma)$ باشد، آن‌گاه $M = X^T X \sim Wishart_d(\Sigma, n)$.

به عنوان مثال ماتریس کوواریانس نمونه (S) دارای توزیع ویشارت است: $S \sim Wishart_d(\Sigma/n - ۱, n - ۱)$.

اگر $S \sim Wishart_d(\Sigma, n)$، و A یک ماتریس $q \times d$ با رتبه کامل q باشد، $ASA^T \sim Wishart_q(A\Sigma A^T, n)$.

توزیع ویشارت

اگر سطرهای ماتریس X $n \times d$ یک نمونه تصادفی ساده از توزیع $N_d(۰, \Sigma)$ باشد، آن‌گاه $M = X^T X \sim Wishart_d(\Sigma, n)$.

به عنوان مثال ماتریس کوواریانس نمونه (S) دارای توزیع ویشارت است: $S \sim Wishart_d(\Sigma/n - ۱, n - ۱)$.

اگر $S \sim Wishart_d(\Sigma, n)$، و A یک ماتریس $q \times d$ با رتبه کامل q باشد، $ASA^T \sim Wishart_q(A\Sigma A^T, n)$.

یکی از کاربردهای آن در مدل‌بندی بیزی مدل‌های رگرسیونی زمانی که برای ماتریس کوواریانس توزیع پیشین ویشارت انتخاب می‌شود.

توزیع ویشارت

اگر سطرهای ماتریس X $n \times d$ یک نمونه تصادفی ساده از توزیع $N_d(۰, \Sigma)$ باشد، آن‌گاه $M = X^T X \sim Wishart_d(\Sigma, n)$.

به عنوان مثال ماتریس کوواریانس نمونه (S) دارای توزیع ویشارت است: $S \sim Wishart_d(\Sigma/n - ۱, n - ۱)$.

اگر $S \sim Wishart_d(\Sigma, n)$، و A یک ماتریس $q \times d$ با رتبه کامل q باشد، $ASA^T \sim Wishart_q(A\Sigma A^T, n)$.

یکی از کاربردهای آن در مدل‌بندی بیزی مدل‌های رگرسیونی زمانی که برای ماتریس کوواریانس توزیع پیشین ویشارت انتخاب می‌شود.

نمونه‌گیری از توزیع ویشارت با تولید نمونه از نرمال چندمتغیره و ضرب ماتریسی آن‌ها انجام می‌شود. اما دو عیب اساسی دارد:

- برای nهای بزرگ کارا نیست
- برای nهای ناصحیح قابل کاربرد نیست

تجزیه بارتلت

یک روش کاراتر استفاده از تجزیه بارتلت می‌باشد.

تجزیه بارتلت

یک روش کاراتر استفاده از تجزیه بارتلت می‌باشد.

اگر $S \sim Wishart_d(I, n)$ دارای تجزیه چولسکی $S = LL^T$ باشد، آنگاه $p(p+۱)/۲$ عضو ناصفر L مستقل از هم و دارای توزیع‌های

- $t_{ii} \sim \sqrt{\chi^۲_{n-i+۱}}$
- $t_{ij} \sim N(۰, ۱)\ (i \neq j)$

هستند. برای تولید $Wishart_d(\Sigma, n)$ ، می‌توان تجزیه $\Sigma = AA^T$ را انتخاب کرد و مقدار ASA^T را برگرداند.

تجزیه بارتلت

یک روش کاراتر استفاده از تجزیه بارتلت می‌باشد.

اگر $S \sim Wishart_d(I, n)$ دارای تجزیه چولسکی $S = LL^T$ باشد، آنگاه $p(p+۱)/۲$ عضو ناصفر L مستقل از هم و دارای توزیع‌های

- $t_{ii} \sim \sqrt{\chi^۲_{n-i+۱}}$
- $t_{ij} \sim N(۰, ۱)\ (i \neq j)$

هستند. برای تولید $Wishart_d(\Sigma, n)$ ، می‌توان تجزیه $\Sigma = AA^T$ را انتخاب کرد و مقدار ASA^T را برگرداند.

در بسته $mixAK$، تابع $rWishart(n, df, S)$ و در بسته $bayesSurv$، تابع $rWishart(n, df, S)$ این کار را انجام می‌دهند. در این دو تابع، n تعداد ماتریس‌های ویشارت و df درجه آزادی است.

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه ششم: نمایش گرافیکی داده‌ها

حسین باغیشنی

دانشگاه شاهرود

۱۶ آبان ۱۳۹۳

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

در واقع همراه با هر روش آماری، نمایش‌های گرافیکی برای بررسی وضعیت موجود، تحلیل نیکویی مدل در نظر گرفته شده، پیدا کردن برخی مشکلات، بهتر کردن مدل قبلی و خیلی موارد دیگر، به کار می‌روند.

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

در واقع همراه با هر روش آماری، نمایش‌های گرافیکی برای بررسی وضعیت موجود، تحلیل نیکویی مدل در نظر گرفته شده، پیدا کردن برخی مشکلات، بهتر کردن مدل قبلی و خیلی موارد دیگر، به کار می‌روند.

نمایش گرافیکی، بیشتر مربوط به تحلیل اکتشافی داده‌ها ($ExploratoryDataAnalysis$) (EDA) و نمودارهای آماری است.

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

در واقع همراه با هر روش آماری، نمایش‌های گرافیکی برای بررسی وضعیت موجود، تحلیل نیکویی مدل در نظر گرفته شده، پیدا کردن برخی مشکلات، بهتر کردن مدل قبلی و خیلی موارد دیگر، به کار می‌روند.

نمایش گرافیکی، بیشتر مربوط به تحلیل اکتشافی داده‌ها ($ExploratoryDataAnalysis$) (EDA) و نمودارهای آماری است.

اصطلاح اکتشافی در مقابل تاییدی است که روش‌های تاییدی را می‌توان به آزمون فرضیه‌ها منتسب کرد.

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

در واقع همراه با هر روش آماری، نمایش‌های گرافیکی برای بررسی وضعیت موجود، تحلیل نیکویی مدل در نظر گرفته شده، پیدا کردن برخی مشکلات، بهتر کردن مدل قبلی و خیلی موارد دیگر، به کار می‌روند.

نمایش گرافیکی، بیشتر مربوط به تحلیل اکتشافی داده‌ها ($ExploratoryDataAnalysis$) (EDA) و نمودارهای آماری است.

اصطلاح اکتشافی در مقابل تاییدی است که روش‌های تاییدی را می‌توان به آزمون فرضیه‌ها منتسب کرد.

در واقع توکی، معتقد بود قبل از آزمون فرضیه، بهتر است تحلیل اکتشافی بر روی داده‌ها انجام شود تا سوالات مناسب برای پرسیدن و روش‌های مناسب برای پاسخ به آن‌ها را بیابیم.

مقدمه

بحث این جلسه خیلی گسترده است و نمی‌توان همه چیز را پوشاند.

در واقع همراه با هر روش آماری، نمایش‌های گرافیکی برای بررسی وضعیت موجود، تحلیل نیکویی مدل در نظر گرفته شده، پیدا کردن برخی مشکلات، بهتر کردن مدل قبلی و خیلی موارد دیگر، به کار می‌روند.

نمایش گرافیکی، بیشتر مربوط به تحلیل اکتشافی داده‌ها ($ExploratoryDataAnalysis$) (EDA) و نمودارهای آماری است.

اصطلاح اکتشافی در مقابل تاییدی است که روش‌های تاییدی را می‌توان به آزمون فرضیه‌ها منتسب کرد.

در واقع توکی، معتقد بود قبل از آزمون فرضیه، بهتر است تحلیل اکتشافی بر روی داده‌ها انجام شود تا سوالات مناسب برای پرسیدن و روش‌های مناسب برای پاسخ به آن‌ها را بیابیم.

در این جلسه، قسمتی از موارد مهم و شاید کمی جذاب‌تر را (از دیدگاه مدرس) مطرح می‌کنیم.

نمودارهای چندک ـ چندک

یکی از مواردی که در مورد داده‌ها مایلیم بدانیم، این است که آیا آن‌ها از توزیع نرمال پیروی می‌کنند؟ آیا از یک توزیع شناخته‌شده پیروی می‌کنند؟

نمودارهای چندک ـ چندک

یکی از مواردی که در مورد داده‌ها مایلیم بدانیم، این است که آیا آن‌ها از توزیع نرمال پیروی می‌کنند؟ آیا از یک توزیع شناخته‌شده پیروی می‌کنند؟

در بعضی از موارد، واضح است که داده‌ها مثلا از نرمال پیروی نمی‌کنند. زیرا به عنوان مثال چگالی داده‌ها دو مدی است. اما مواردی هم وجود دارند که چنین آگاهی واضح و روشن نیست.

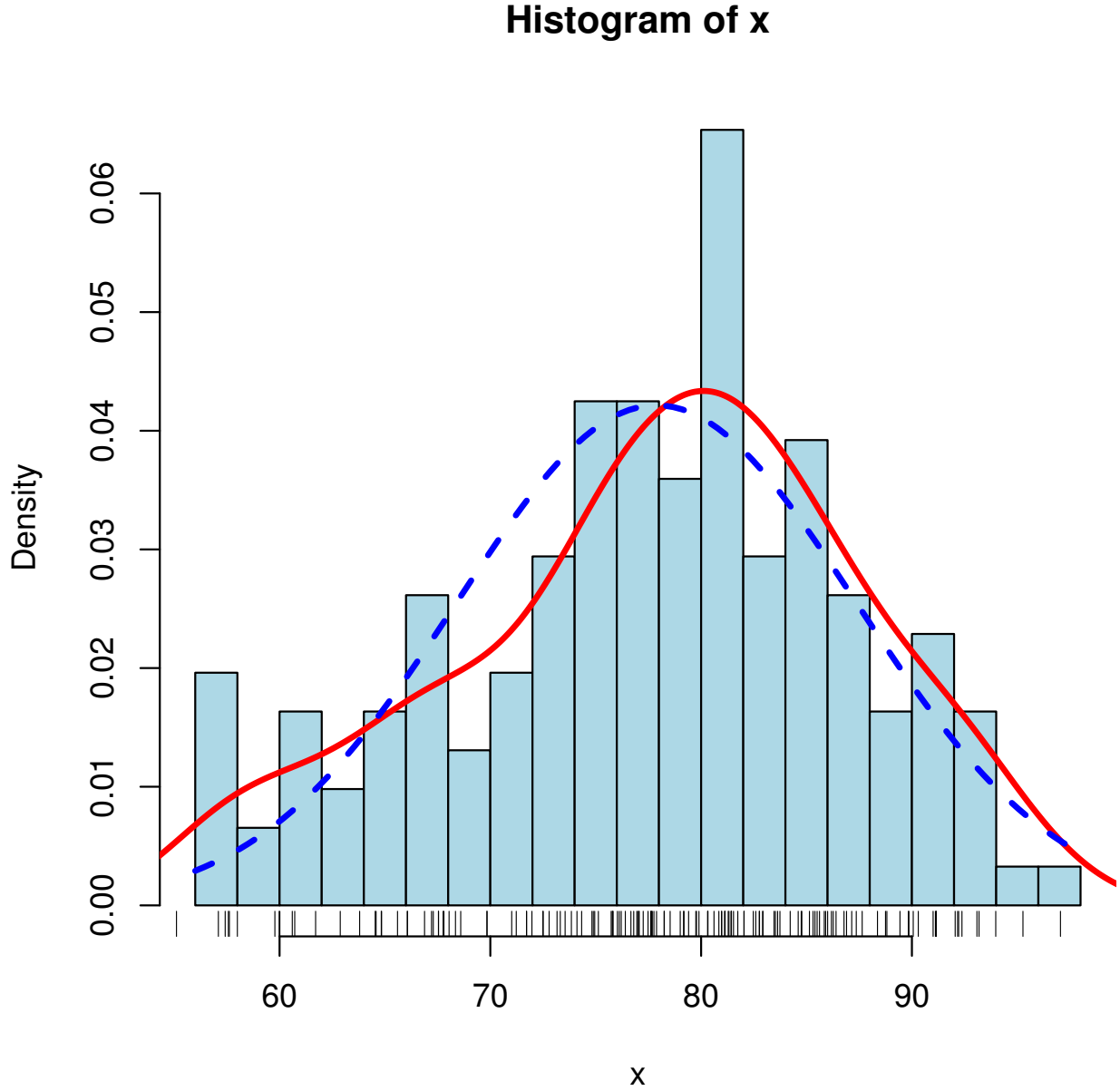
نمودارهای چندک ـ چندک

یکی از مواردی که در مورد داده‌ها مایلیم بدانیم، این است که آیا آن‌ها از توزیع نرمال پیروی می‌کنند؟ آیا از یک توزیع شناخته‌شده پیروی می‌کنند؟

در بعضی از موارد، واضح است که داده‌ها مثلا از نرمال پیروی نمی‌کنند. زیرا به عنوان مثال چگالی داده‌ها دو مدی است. اما مواردی هم وجود دارند که چنین آگاهی واضح و روشن نیست.

یک راه برای بررسی این موضوع، مقایسه چگالی برآوردشده با چگالی توزیع نرمال است.

```
data(airquality)
x <- airquality[,4]
hist(x, probability=TRUE, breaks=20, col="light blue")
rug(jitter(x, 5))
points(density(x), type='l', lwd=3, col='red')
f <- function(t) {
  dnorm(t, mean=mean(x), sd=sd(x) )
}
curve(f, add=T, col="blue", lwd=3, lty=2)
```


Histogram of x
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
60
70
80
90
x

راه حل دوم

روش دیگر که یک روش بصری است، رسم چندک‌های نمونه‌ای در مقابل چندک‌های نرمال است. به این نمودار، چندک‌ـ‌چندک گویند.

راه حل دوم

روش دیگر که یک روش بصری است، رسم چندک‌های نمونه‌ای در مقابل چندک‌های نرمال است. به این نمودار، چندک‌ـ‌چندک گویند.

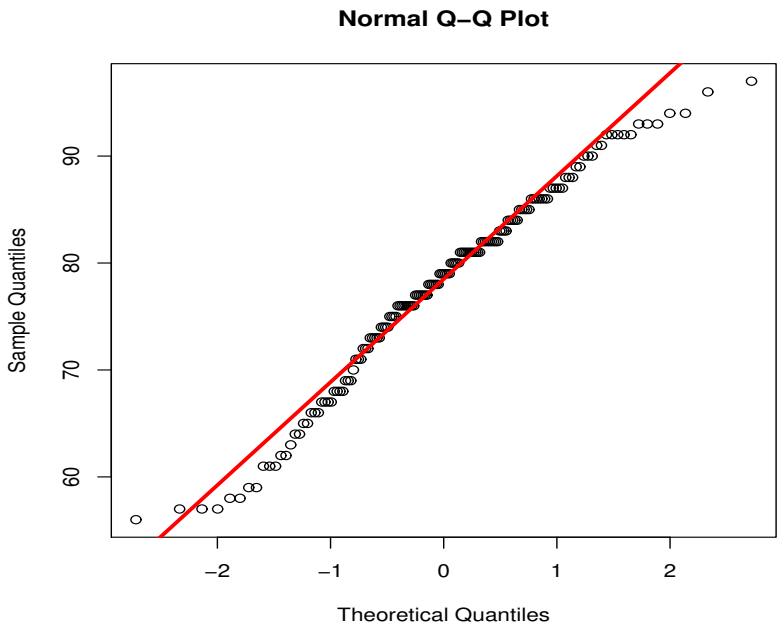
در نمودار زیر به نظر می‌رسد داده‌ها تقریبا نرمال هستند، اما می‌توان دید که داده‌ها گسسته‌اند!!

راه حل دوم

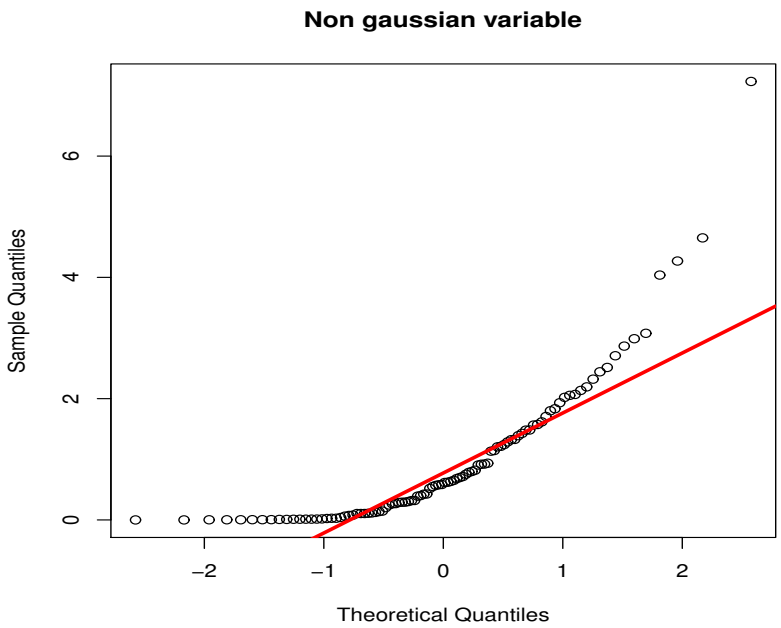
روش دیگر که یک روش بصری است، رسم چندک‌های نمونه‌ای در مقابل چندک‌های نرمال است. به این نمودار، چندک‌ـ‌چندک گویند.

در نمودار زیر به نظر می‌رسد داده‌ها تقریبا نرمال هستند، اما می‌توان دید که داده‌ها گسسته‌اند!!

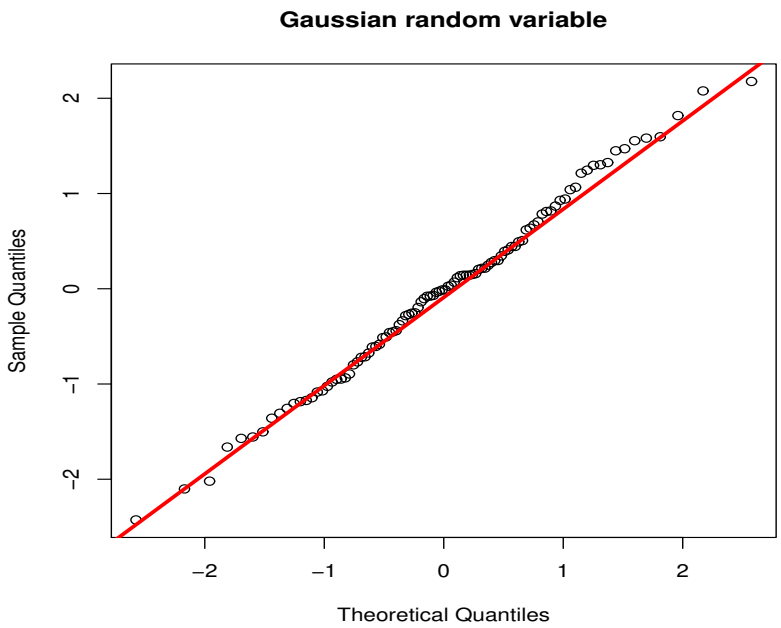
```
x <- airquality[,4]
qqnorm(x)
qqline(x, col="red", lwd=3)
```



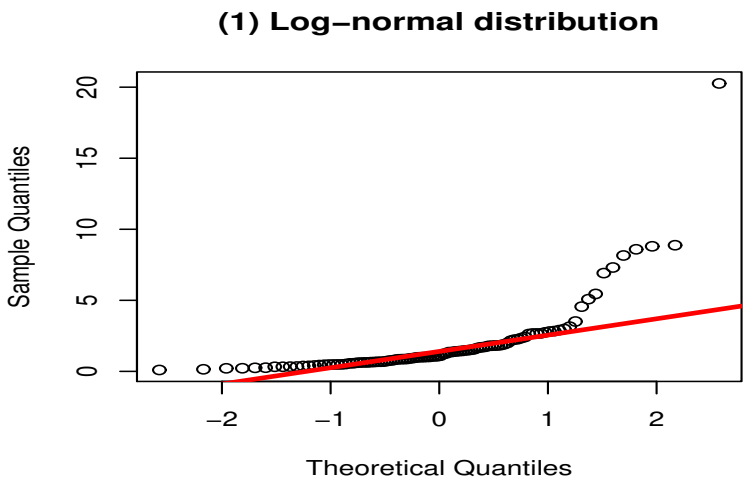
```
y <- rnorm(100)^2
qqnorm(y, main="Non gaussian variable")
qqline(y, col="red", lwd=3)
```

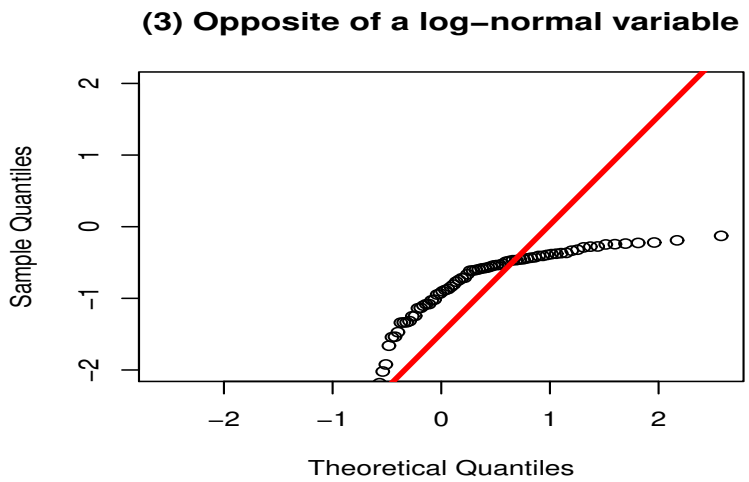


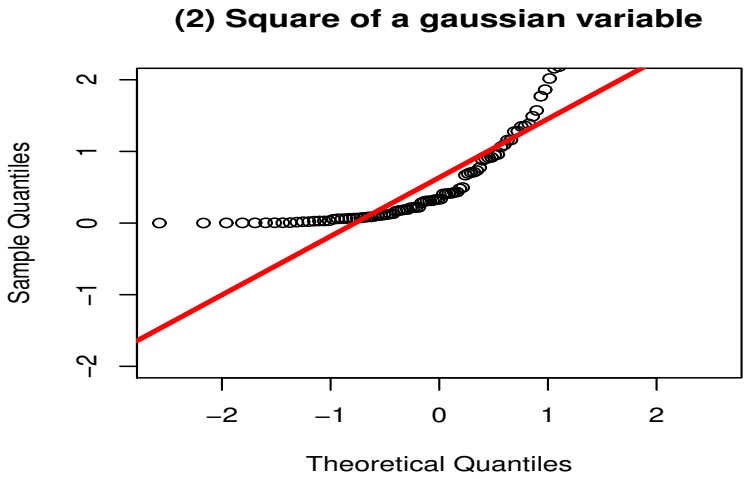

```
y <- rnorm(100)
qqnorm(y, main="Gaussian random variable")
qqline(y, col="red", lwd=3)
```

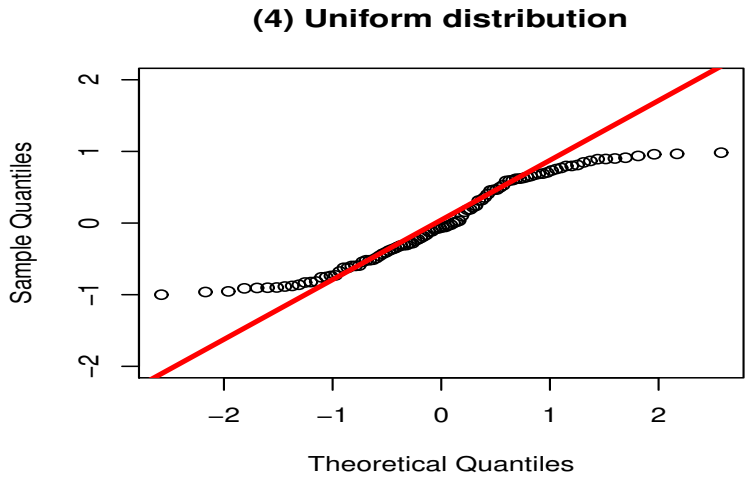


سایر مثال‌ها

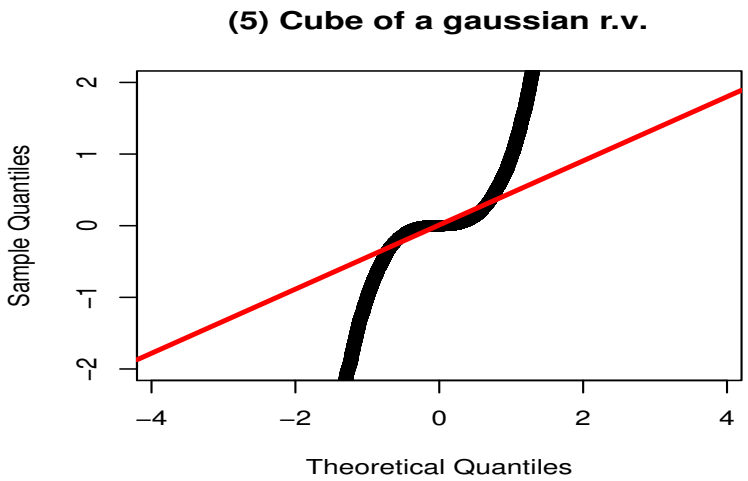

(1) Log−normal distribution
Sample Quantiles
Theoretical Quantiles
0
5
10
15
20
−2
−1
0
1
2

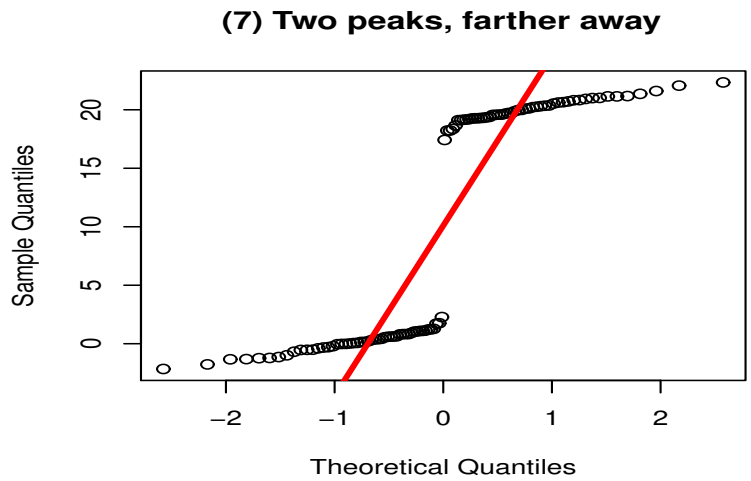

(3) Opposite of a log−normal variable
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2

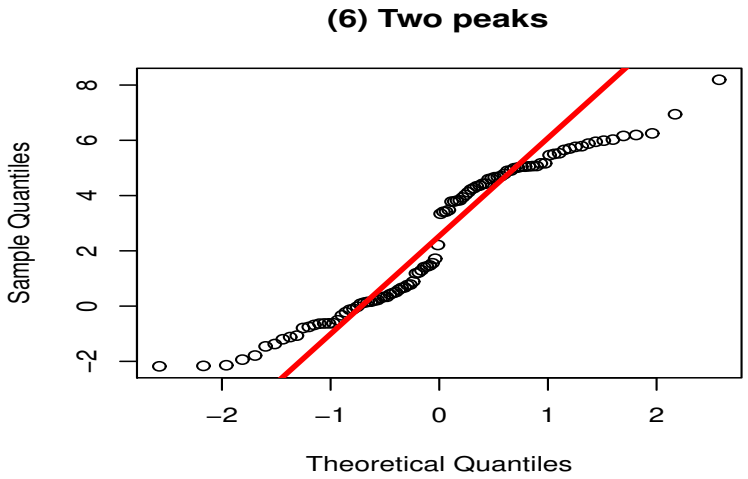

(2) Square of a gaussian variable
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2

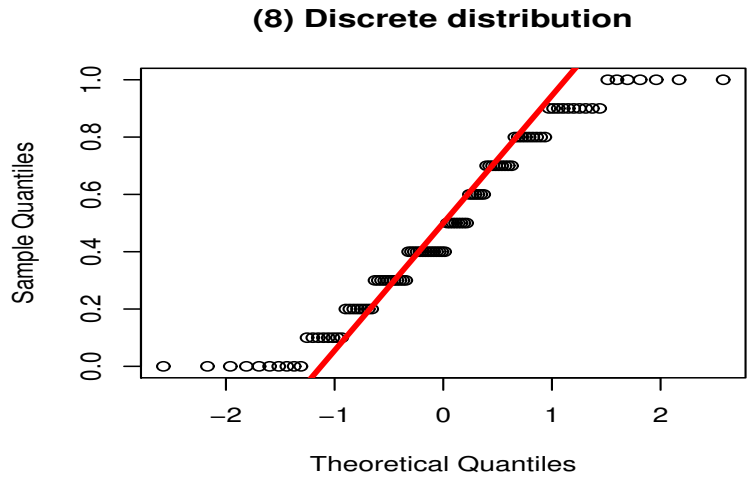

(4) Uniform distribution
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2

سایر مثال‌ها


(5) Cube of a gaussian r.v.
Sample Quantiles
Theoretical Quantiles
−4
−2
0
2
4
−2
−1
0
1
2

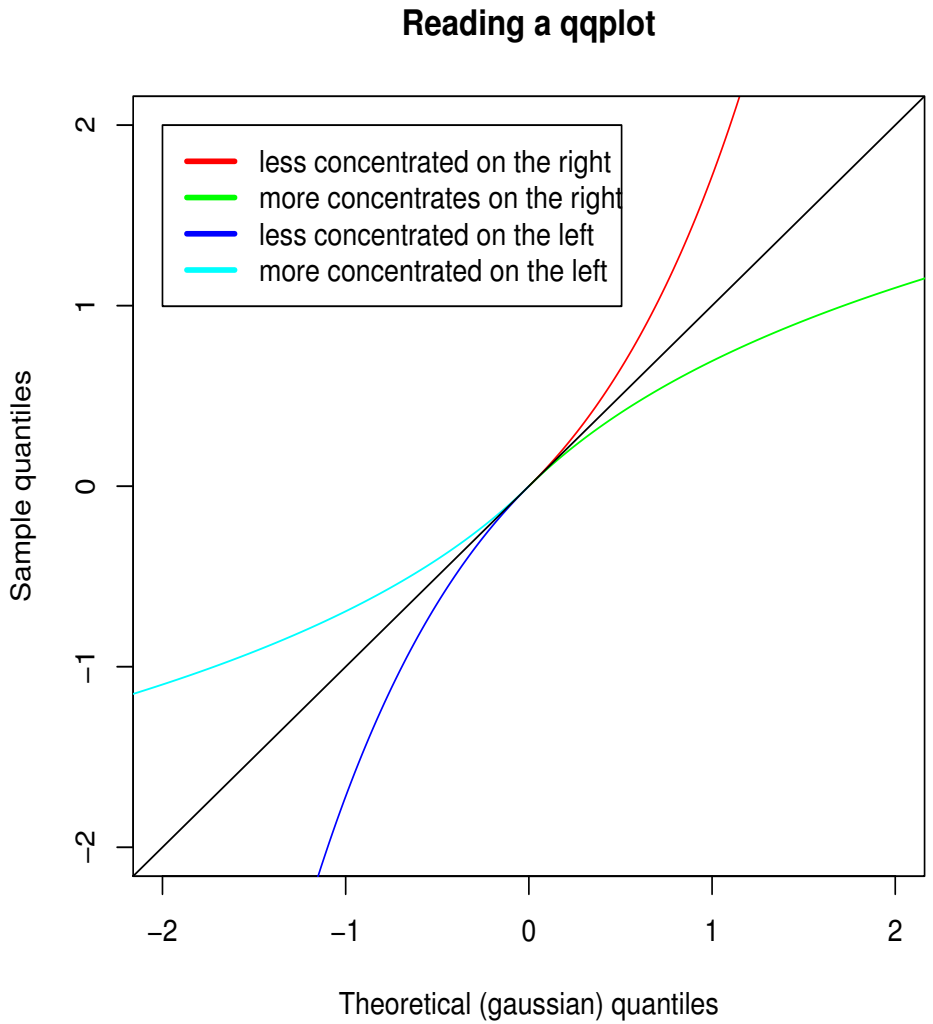

(7) Two peaks, farther away
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2
0
5
10
15
20


(6) Two peaks
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2
−2
0
2
4
6
8

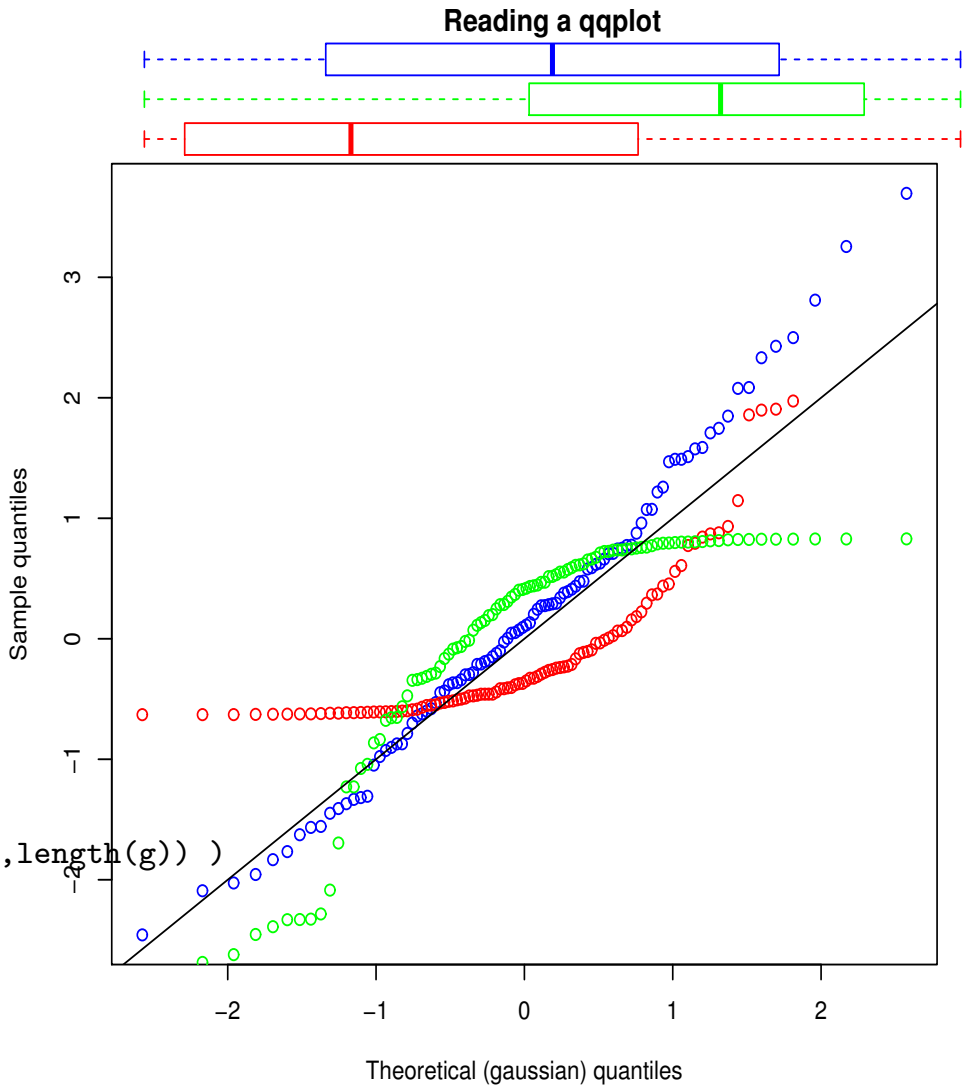

(8) Discrete distribution
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2
0.0
0.2
0.4
0.6
0.8
1.0

دستورالعمل کلی

```
x <- seq(from=0, to=2, length=100)
y <- exp(x)-1
plot( y ~ x, type = 'l', col = 'red',
      xlim = c(-2,2), ylim = c(-2,2),
      xlab = "Theoretical (gaussian) quantiles",
      ylab = "Sample quantiles")
lines( x~y, type='l', col='green')
x <- -x
y <- -y
lines( y~x, type='l', col='blue', )
lines( x~y, type='l', col='cyan')
abline(0,1)
legend( -2, 2,
        c( "less concentrated on the right",
           "more concentrates on the right",
           "less concentrated on the left",
           "more concentrated on the left"
         ),
        lwd=3,
        col=c("red", "green", "blue", "cyan")
      )
title(main="Reading a qqplot")
```



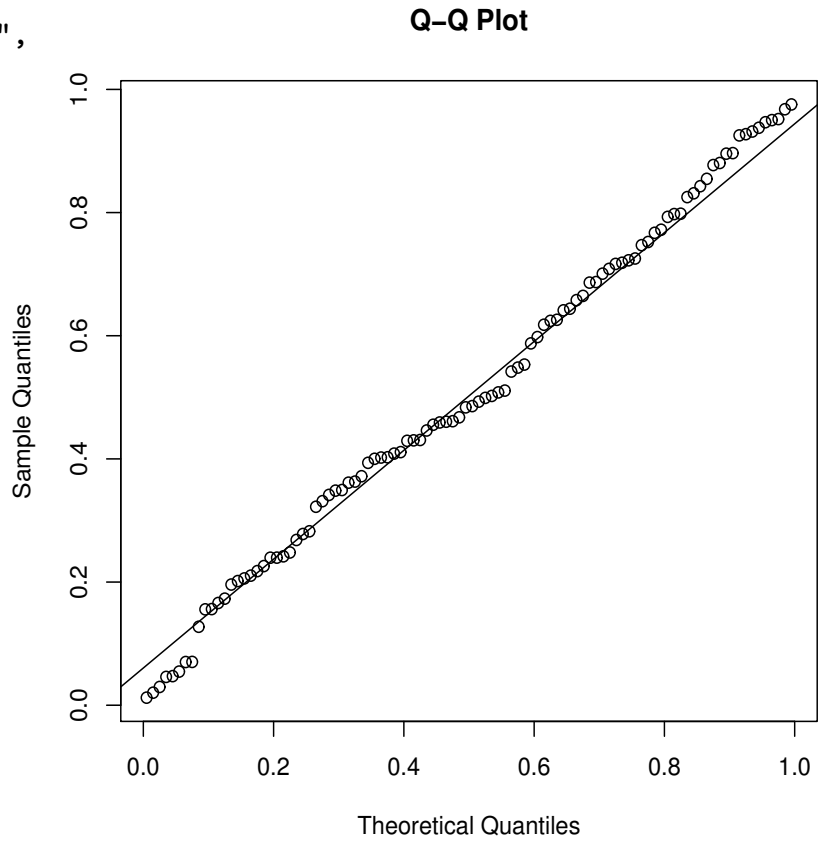
```
op <- par()
layout( matrix( c(2,2,1,1), 2, 2, byrow=T ),
        c(1,1), c(1,6),
      )
# The plot
n <- 100
y <- rnorm(n)
x <- qnorm(ppoints(n))[order(order(y))]
par(mar=c(5.1,4.1,0,2.1))
plot( y ~ x, col = "blue",
      xlab = "Theoretical (gaussian) quantiles",
      ylab = "Sample quantiles" )
y1 <- scale( rnorm(n)^2 )
x <- qnorm(ppoints(n))[order(order(y1))]
lines(y1~x, type="p", col="red")
y2 <- scale( -rnorm(n)^2 )
x <- qnorm(ppoints(n))[order(order(y2))]
lines(y2~x, type="p", col="green")
abline(0,1)
# The legend
par(bty='n', ann=F)
g <- seq(0,1, length=10)
e <- g^2
f <- sqrt(g)
h <- c( rep(1,length(e)), rep(2,length(f)), rep(3,length(g)) )
par(mar=c(0,4.1,1,0))
boxplot( c(e,f,g) ~ h, horizontal=T,
         border=c("red", "green", "blue"),
         col="white", # Something prettier?
         xaxt='n',
         yaxt='n',
         )
title(main="Reading a qqplot")
par(op)
```



چندک_چندک برای سایر توزیع‌ها

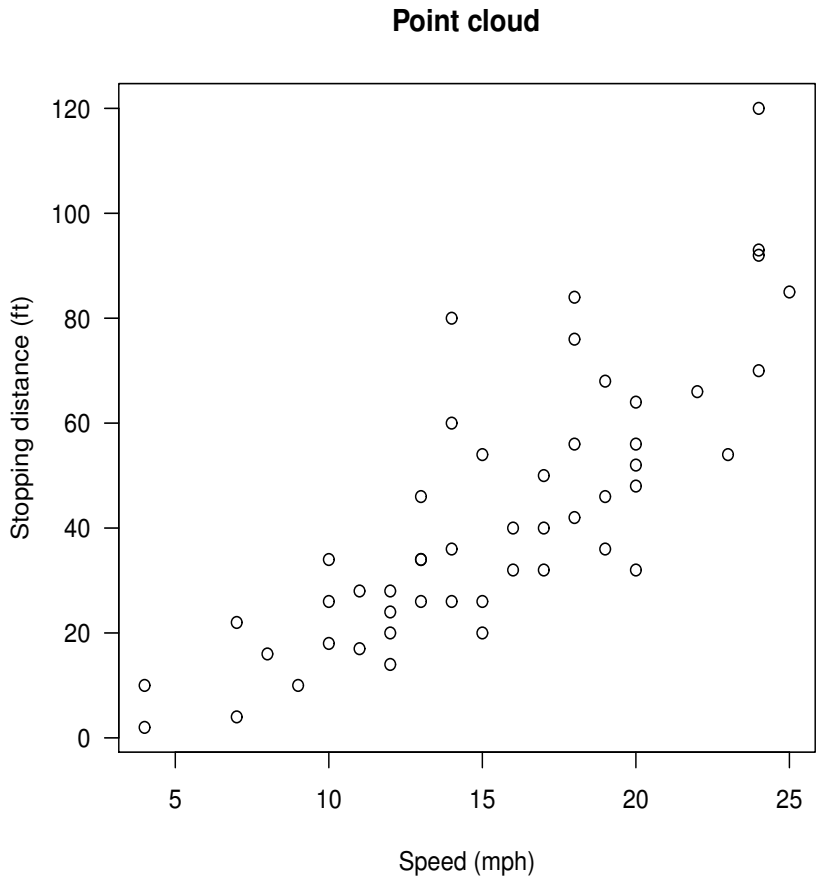
می‌توان برای سایر توزیع‌ها از همین ایده نمودار چندک_چندک نرمال استفاده کرد

```
qq <- function (y, ylim, quantiles=qnorm,
    main = "Q-Q Plot", xlab = "Theoretical Quantiles",
    ylab = "Sample Quantiles", plot.it = TRUE, ...)
{
  y <- y[!is.na(y)]
  if (0 == (n <- length(y)))
    stop("y is empty")
  if (missing(ylim))
    ylim <- range(y)
  x <- quantiles(ppoints(n))[order(order(y))]
  if (plot.it)
    plot(x, y, main = main, xlab = xlab,
    ylab = ylab, ylim = ylim, ...)
  # From qqline
  y <- quantile(y, c(0.25, 0.75))
  x <- quantiles(c(0.25, 0.75))
  slope <- diff(y)/diff(x)
  int <- y[1] - slope * x[1]
  abline(int, slope, ...)
  invisible(list(x = x, y = y))
}
y <- runif(100)
qq(y, quantiles=qunif)
```



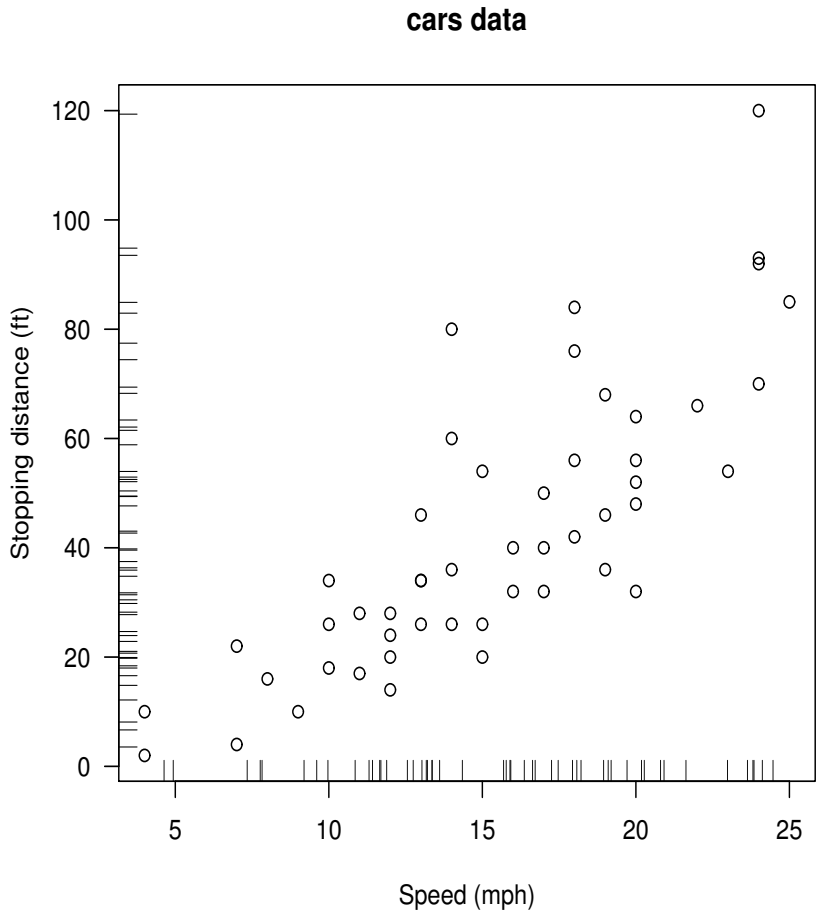
نمودارهای پراکنش

```
data(cars)
plot(cars$dist ~ cars$speed,
     xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
title(main = "Point cloud")
```



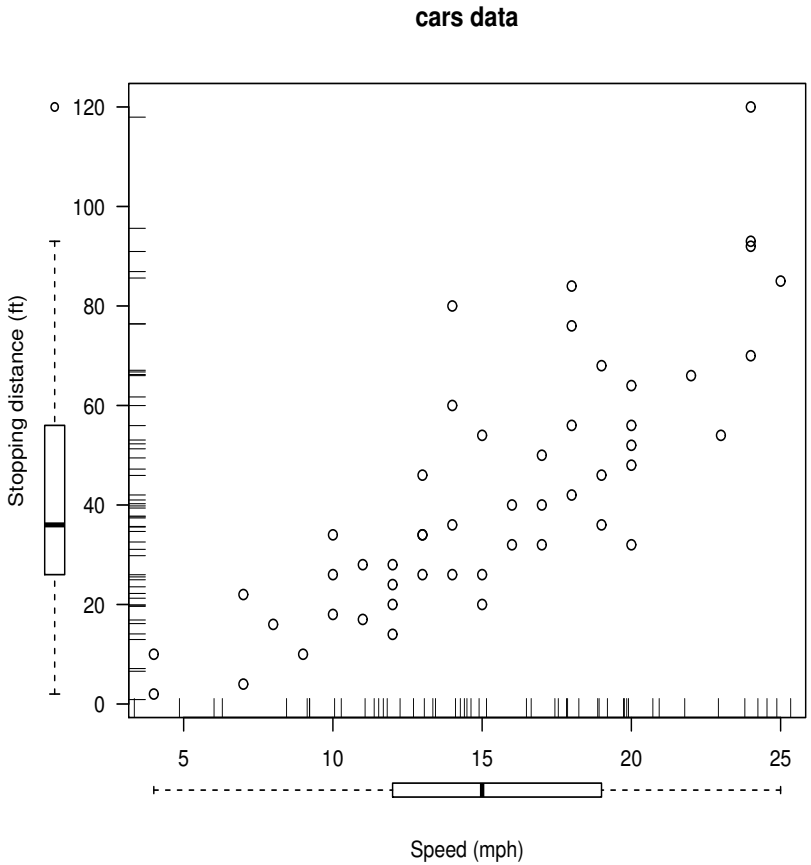
می‌توان نمودارهای پراکنش تک‌بعدی را هم به حاشیه نمودار اضافه کرد.

```
plot(cars$dist ~ cars$speed,
     xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
title(main = "cars data")
rug(side=1, jitter(cars$speed, 5))
rug(side=2, jitter(cars$dist, 20))
```



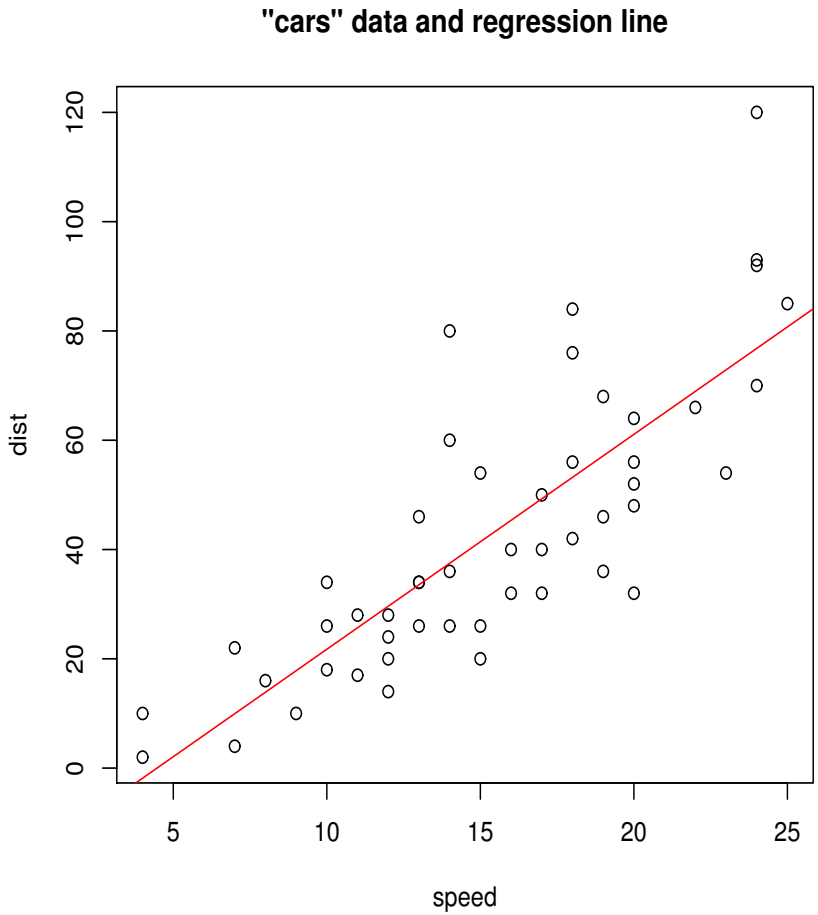
حتی می‌توان نمودارهای جعبه‌ای حاشیه‌ای هر متغیر را نیز همراه با نمودار پراکنش رسم کرد.

```
layout( matrix( c(2,1,0,3), 2, 2, byrow=T ),
        c(1,6), c(4,1),
      )

par(mar=c(1,1,5,2))
plot(cars$dist ~ cars$speed,
     xlab='', ylab='',
     las = 1)
rug(side=1, jitter(cars$speed, 5) )
rug(side=2, jitter(cars$dist, 20) )
title(main = "cars data")

par(mar=c(1,2,5,1))
boxplot(cars$dist, axes=F)
title(ylab='Stopping distance (ft)', line=0)

par(mar=c(5,1,1,2))
boxplot(cars$speed, horizontal=T, axes=F)
title(xlab='Speed (mph)', line=1)
```



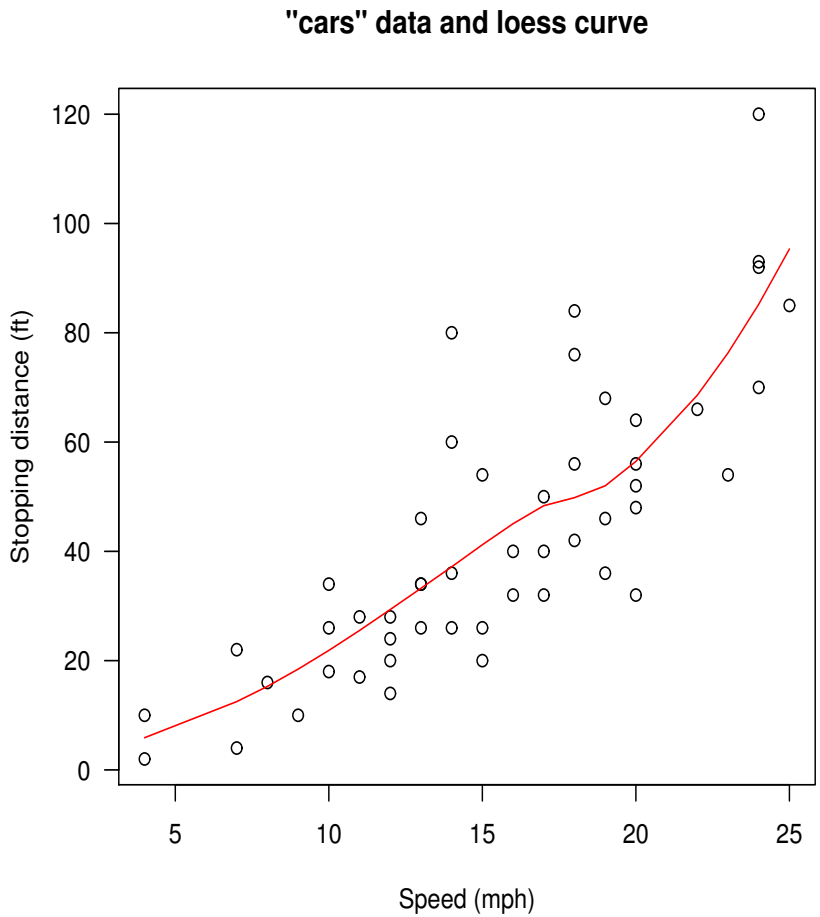
می‌توان یک تقریب خطی برای رابطه بین دو متغیر به نمودار اضافه کرد.

```
plot(dist ~ speed, data = cars,
     main = "\"cars\" data and regression line")
abline(lm( dist ~ speed, data = cars), col = 'red')
```



می‌توان مثلا از تابع *loess* برای تقریب یک منحنی (نه لزوما خطی) برای رابطه دو متغیر استفاده کرد.

```
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
r <- loess(dist ~ speed, data=cars)
lines(r$x, r$fitted, col="red")
title(main = "\"cars\" data and loess curve")
```

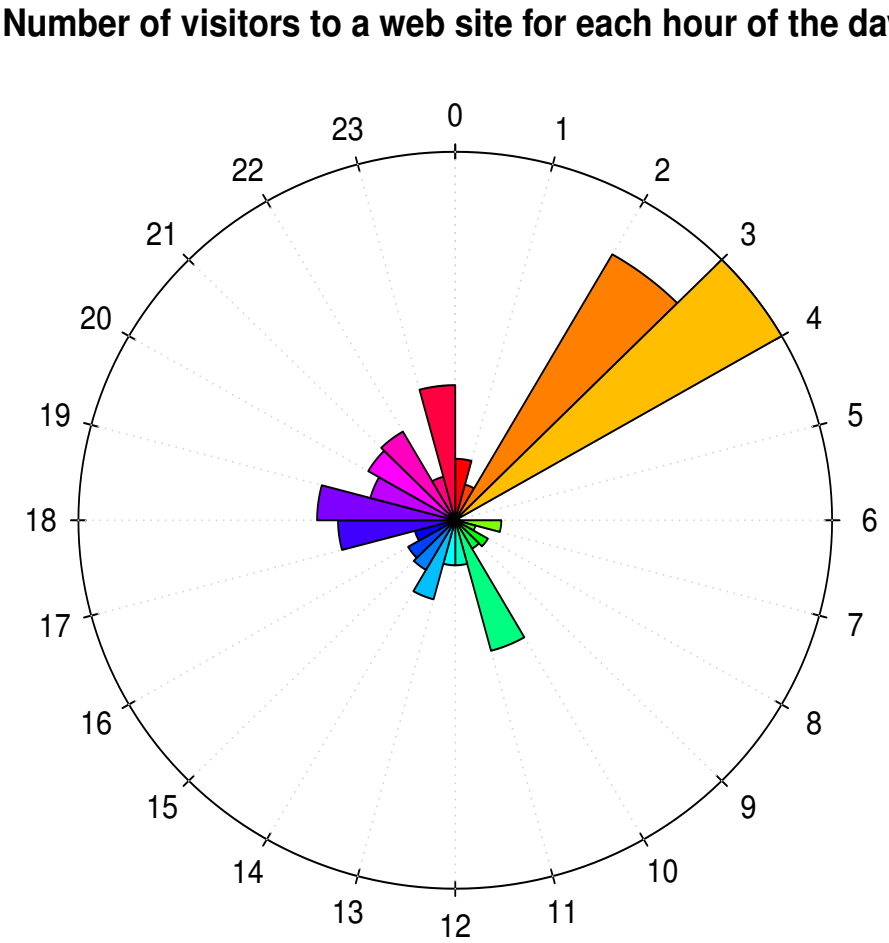


نمودارهای مرتبط با یک متغیر دوره‌ای

اگر یک متغیر، دوره‌ای باشد، مانند ساعت‌های یک روز، نمایش آن‌ها به عنوان یک دایره گاهی مواقع نمودار خیلی مفیدی است. به عنوان مثال، تعداد بازدید از یک سایت را به عنوان تابعی از زمان روز در نظر بگیرید:

```
x <- c(15, 9, 75, 90, 1, 1, 11, 5, 9, 8, 33, 11, 11,
       20, 14, 13, 10, 28, 33, 21, 24, 25, 11, 33)
clock.plot <- function (x, col = rainbow(n), ...) {
  if( min(x)<0 ) x <- x - min(x)
  if( max(x)>1 ) x <- x/max(x)
  n <- length(x)
  if(is.null(names(x))) names(x) <- 0:(n-1)
  m <- 1.05
  plot(0,
       type = 'n', # do not plot anything
       xlim = c(-m,m), ylim = c(-m,m),
       axes = F, xlab = '', ylab = '', ...)
  a <- pi/2 - 2*pi/200*0:200
  polygon( cos(a), sin(a) )
  v <- .02
  a <- pi/2 - 2*pi/n*0:n
  segments( (1+v)*cos(a), (1+v)*sin(a),
            (1-v)*cos(a), (1-v)*sin(a) )
  segments( cos(a), sin(a),
            0, 0,
            col = 'light grey', lty = 3)
  ca <- -2*pi/n*(0:50)/50
  for (i in 1:n) {
    a <- pi/2 - 2*pi/n*(i-1)
    b <- pi/2 - 2*pi/n*i
    polygon( c(0, x[i]*cos(a+ca), 0),
             c(0, x[i]*sin(a+ca), 0),
             col=col[i] )
    v <- .1
    text((1+v)*cos(a), (1+v)*sin(a), names(x)[i])
  }
}
clock.plot(x,
main = "No. of visitors to a web site for each hour")
```

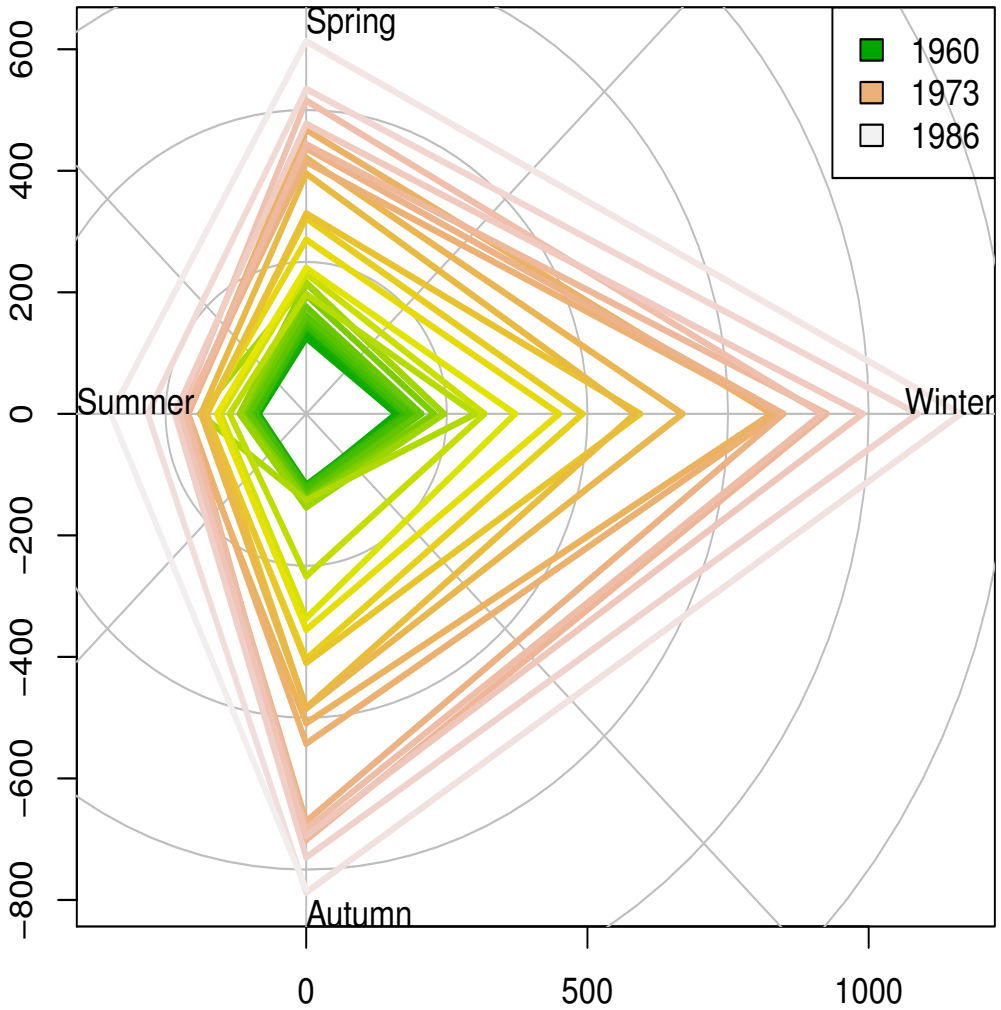
در مورد نوع داده‌های این نمودار، می‌توانید به راهنمای بسته $circular$ مراجعه کنید. همچنین نمودار مشابهی را می‌توان با استفاده از بسته $plotrix$ تولید کرد.



نمودار قطبی

یک نمودار مفید مشابه نمودار دوره‌ای بالا، نمودار قطبی است:

```
# Polar plot to spot seasonal patterns
x <- as.vector(UKgas)
n <- length(x)
theta <- seq(0, by=2*pi/4, length=n)
plot(x * cos(theta), x * sin(theta),
     type = "l",
     xlab = "", ylab = "",
     main = "UK gas consumption")
abline(h=0, v=0, col="grey")
abline(0, 1, col="grey")
abline(0, -1, col="grey")
circle <- function (x, y, r, N=100, ...) {
  theta <- seq(0, 2*pi, length=N+1)
  lines(x + r * cos(theta), y + r * sin(theta), ...)
}
circle(0,0, 250, col="grey")
circle(0,0, 500, col="grey")
circle(0,0, 750, col="grey")
circle(0,0, 1000, col="grey")
circle(0,0, 1250, col="grey")
segments( x[-n] * cos(theta[-n]),
          x[-n] * sin(theta[-n]),
          x[-1] * cos(theta[-1]),
          x[-1] * sin(theta[-1]),
          col = terrain.colors(length(x)), lwd = 3)
text(par("usr")[2], 0, "Winter", adj=c(1,0))
text(0, par("usr")[4], "Spring", adj=c(0,1))
text(par("usr")[1], 0, "Summer", adj=c(0,0))
text(0, par("usr")[3], "Autumn", adj=c(0,0))
legend("topright", legend = c(1960, 1973, 1986), fill = terrain.colors(3))
```


UK gas consumption
Spring
Summer
Winter
Autumn
1960
1973
1986
600
400
200
0
−200
−400
−600
−800
0
500
1000

نمودارهای ماتریسی

یکی از مهم‌ترین کاربردهای ابزار گرافیکی، پیدا کردن ساختارها یا گروه‌هایی در داده‌هاست.
نمودارهای ماتریسی یا نمایش پانلی قسمتی از این هدف را می‌توانند انجام دهند.

نمودارهای ماتریسی

یکی از مهم‌ترین کاربردهای ابزار گرافیکی، پیدا کردن ساختارها یا گروه‌هایی در داده‌هاست. نمودارهای ماتریسی یا نمایش پانلی قسمتی از این هدف را می‌توانند انجام دهند.

به عنوان مثال یک ماتریس پراکنش، نمودارهای پراکنش جفت متغیرهای یک آرایه را نمایش می‌دهد. این نمودارها می‌توانند در شناخت وجود رابطه بین متغیرها (به عنوان یافتن ساختار) مفید باشند.

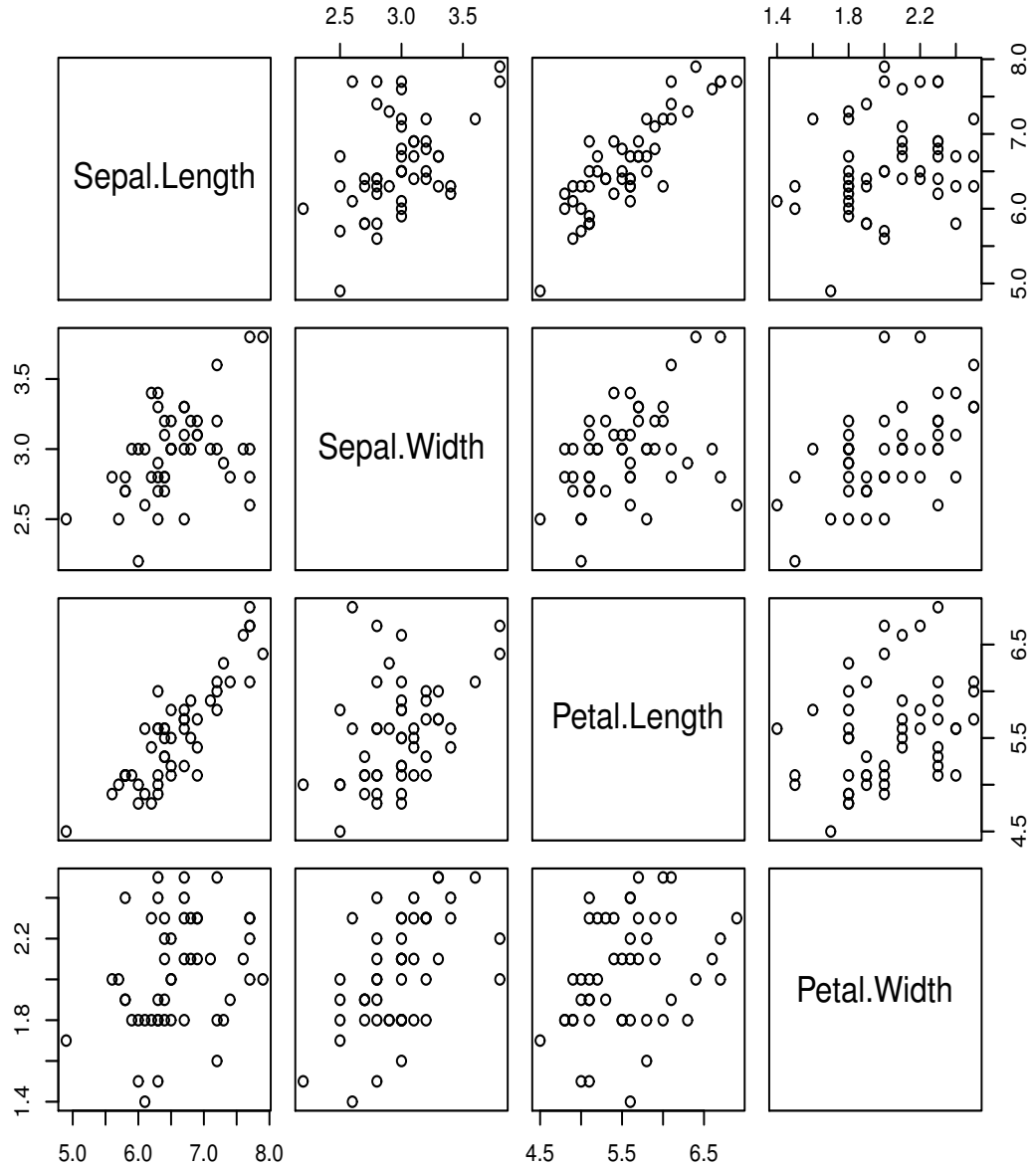
نمودارهای ماتریسی

یکی از مهم‌ترین کاربردهای ابزار گرافیکی، پیدا کردن ساختارها یا گروه‌هایی در داده‌هاست. نمودارهای ماتریسی یا نمایش پانلی قسمتی از این هدف را می‌توانند انجام دهند.

به عنوان مثال یک ماتریس پراکنش، نمودارهای پراکنش جفت متغیرهای یک آرایه را نمایش می‌دهد. این نمودارها می‌توانند در شناخت وجود رابطه بین متغیرها (به عنوان یافتن ساختار) مفید باشند.

تابع *pairs* در بسته *graphics* چنین ماتریسی را تولید می‌کند

```
data(iris)
pairs(iris[101:150,1:4])
```


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.5 3.0 3.5
1.4 1.8 2.2
5.0 6.0 7.0 8.0
4.5 5.5 6.5

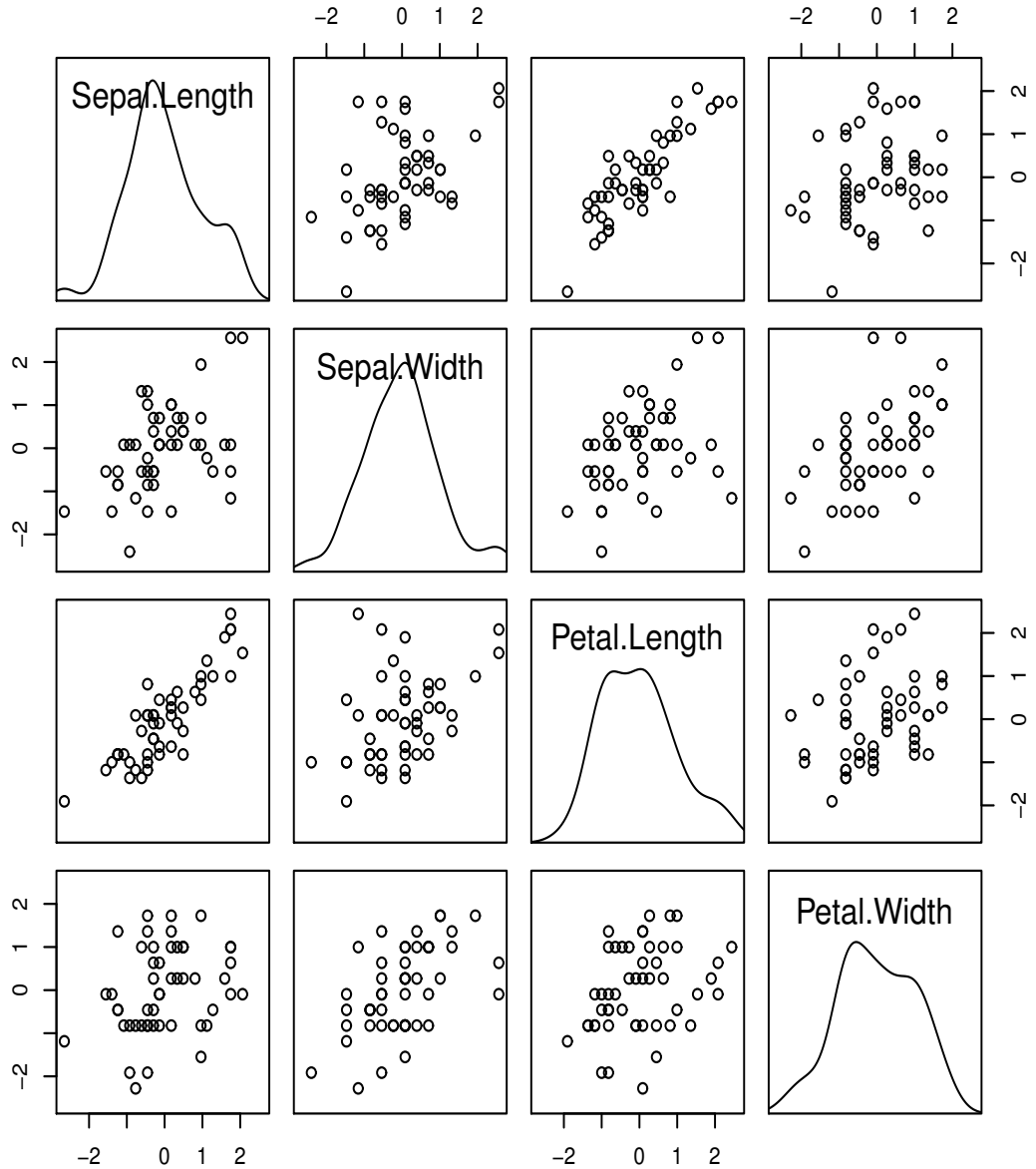
تابع $pairs$ گزینه‌ای دارد با نام $diag.panel$ که تابعی است برای مشخص کردن آن که در روی قطر چه چیزی نمایش داده شود. پیش‌فرض آن نام متغیرهاست.

تابع $pairs$ گزینه‌ای دارد با نام $diag.panel$ که تابعی است برای مشخص کردن آن که در روی قطر چه چیزی نمایش داده شود. پیش‌فرض آن نام متغیرهاست.

به عنوان مثال، می‌توان تابعی را تعریف کرد تا روی قطر، نمودار توابع چگالی برآوردشده هر متغیر را نمایش دهد:

```
panel.d <- function(x, ...) {
        usr <- par("usr")
        on.exit(par(usr))
        par(usr = c(usr[1:2], 0, .5))
        lines(density(x))
    }
x <- scale(iris[101:150, 1:4])
pairs(x, diag.panel = panel.d, xlim = range(x),
ylim = range(x))
```

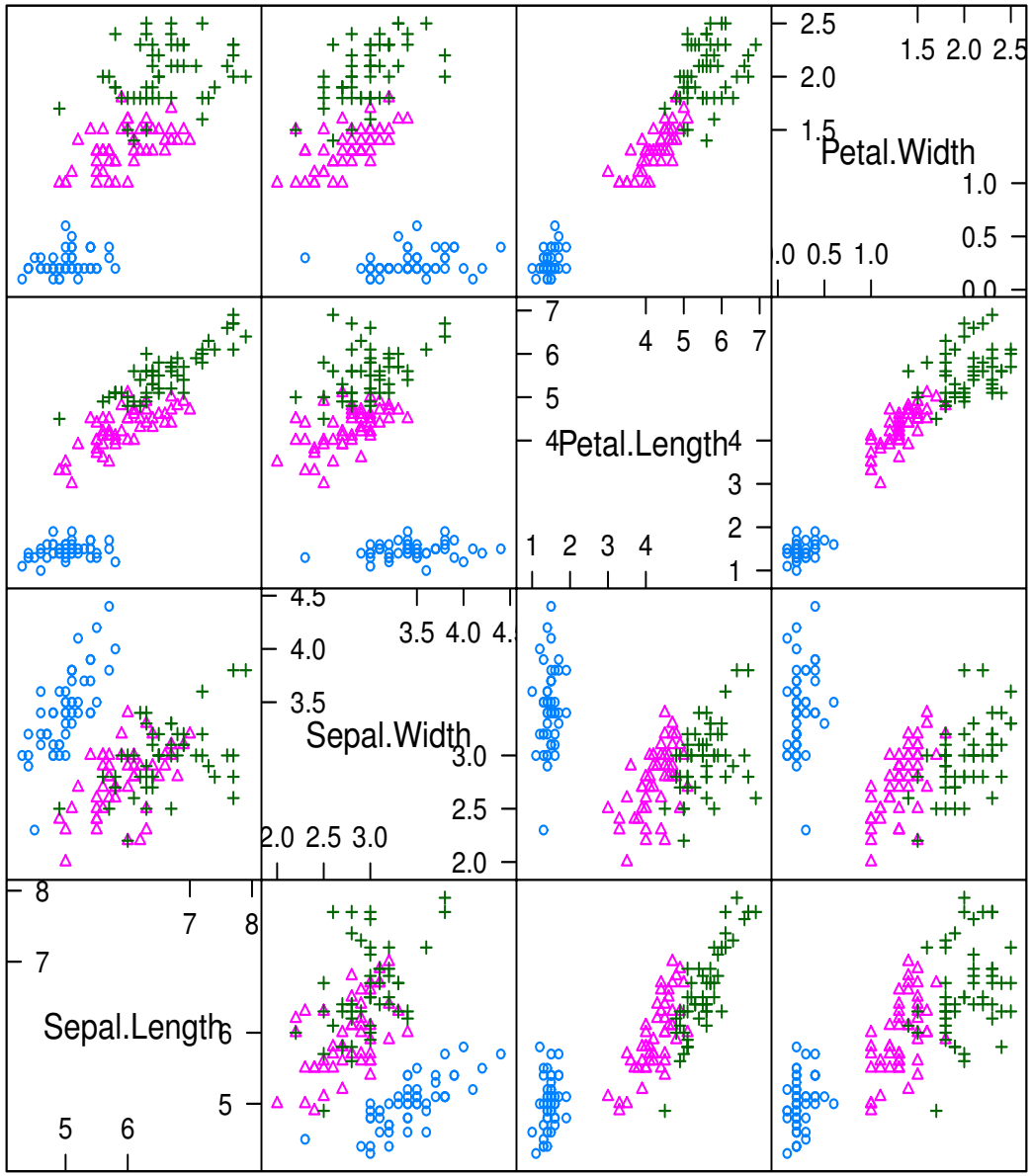
پارامتر usr مشخص می‌کند که مقادیر فرین محورها را خود کاربر مشخص می‌کند و دستور $scale$ نمونه‌های مربوط به هر متغیر (تکی) را استاندارد می‌کند.


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
−2
0
1
2

نمودارهای ماتریسی با استفاده از بسته *lattice*

بسته *lattice* توابعی را برای ساخت نمودارهای ماتریسی و پانلی، فراهم آورده است. به عنوان مثالی قابل مقایسه با ماتریس پراکنش، به مثال زیر توجه کنید:

```
library(lattice)
splom(iris[101:150, 1:4])    # plot 1
splom(iris[,1:4], groups = iris$Species) # plot 2
splom(~iris[1:4], groups = Species, data = iris,
pch = c(1, 2, 3),  cex = c(.5,.5,.5)) #plot 3
```



Scatter Plot Matrix

سایر موارد استفاده از $lattice$

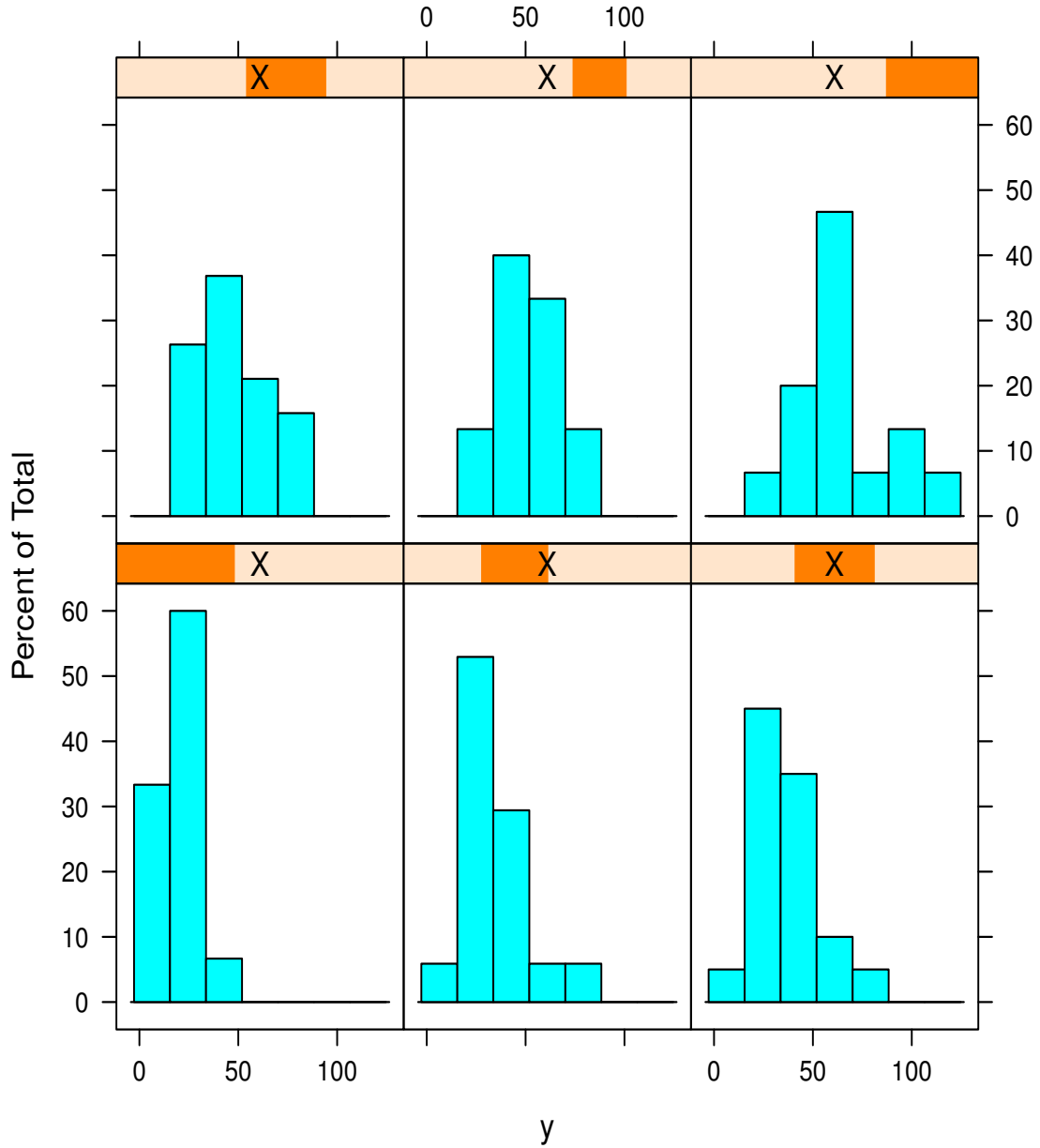
یکی دیگر از موارد استفاده نمودارهای $lattice$، تقسیم کردن مقادیر یک متغیر مانند y و رسم نمودارهای جعبه‌ای، بافت‌نگار و چگالی یک متغیر دیگر مانند x در طبقات حاصل از تقسیم‌بندی است.

سایر موارد استفاده از *lattice*

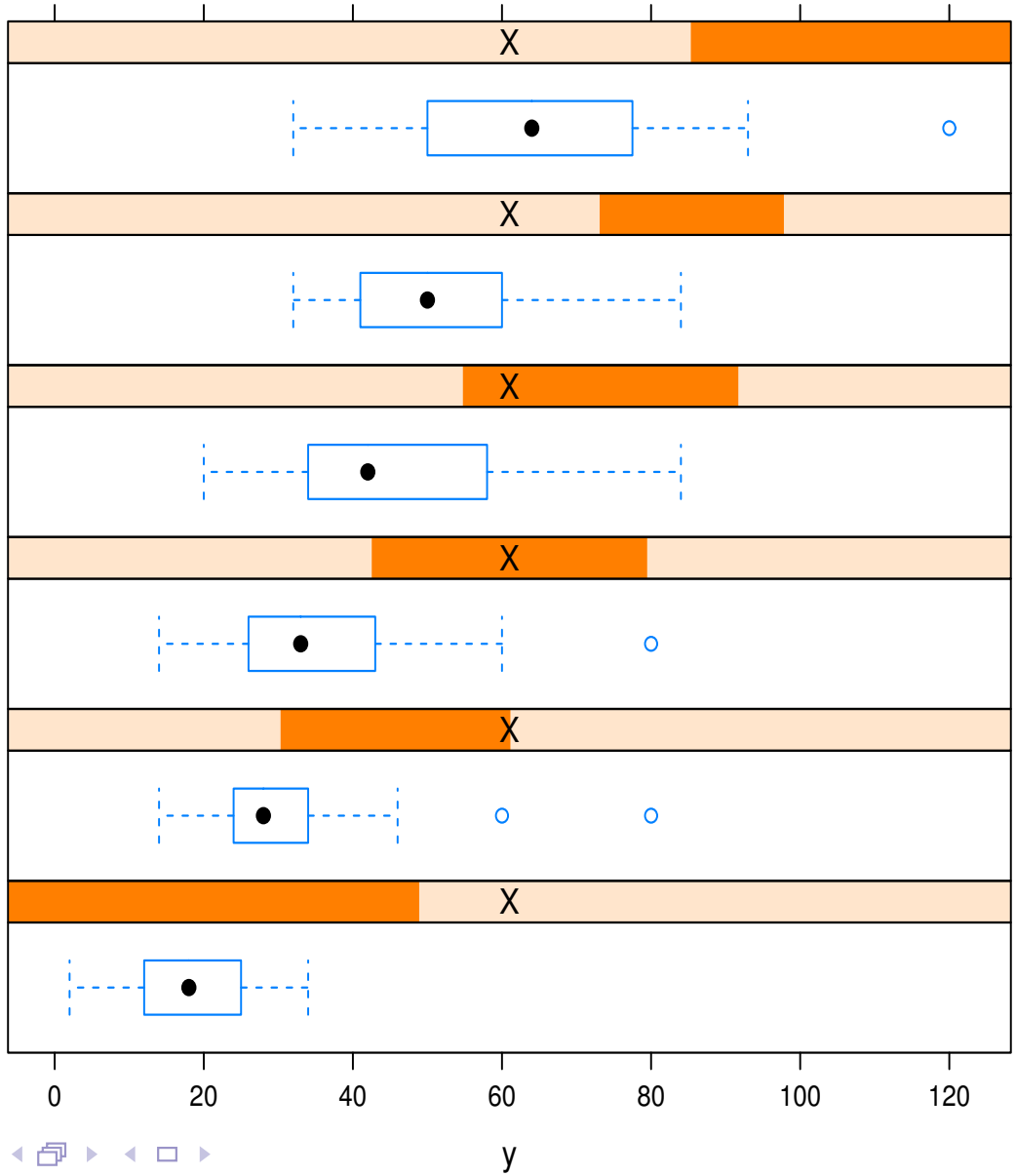
یکی دیگر از موارد استفاده نمودارهای *lattice*، تقسیم کردن مقادیر یک متغیر مانند y و رسم نمودارهای جعبه‌ای، بافت‌نگار و چگالی یک متغیر دیگر مانند x در طبقات حاصل از تقسیم‌بندی است.

این نمودارها در بعضی از موارد بسیار مفید هستند.

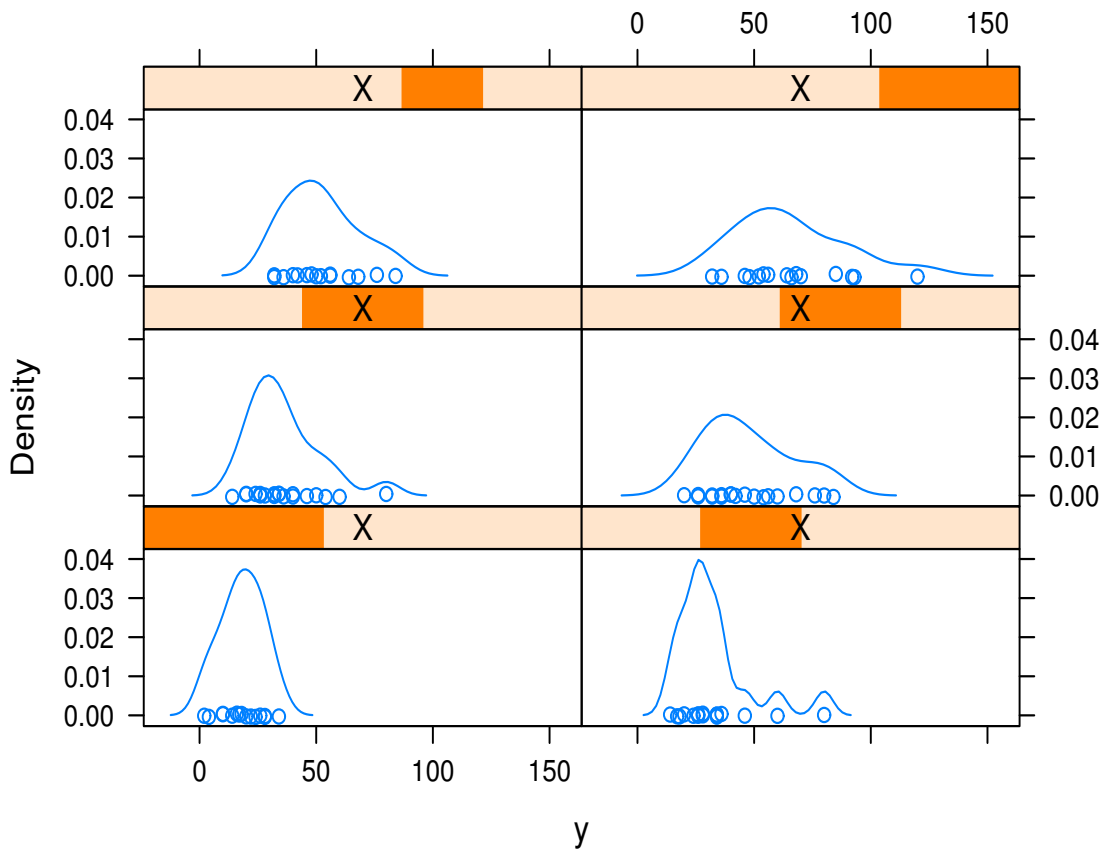
```
library(lattice)
y <- cars$dist
x <- cars$speed
X <- equal.count(x)
histogram(~ y | X)
```


X
X
X
X
X
X
Percent of Total
y
0
50
100
10
20
30
40
60

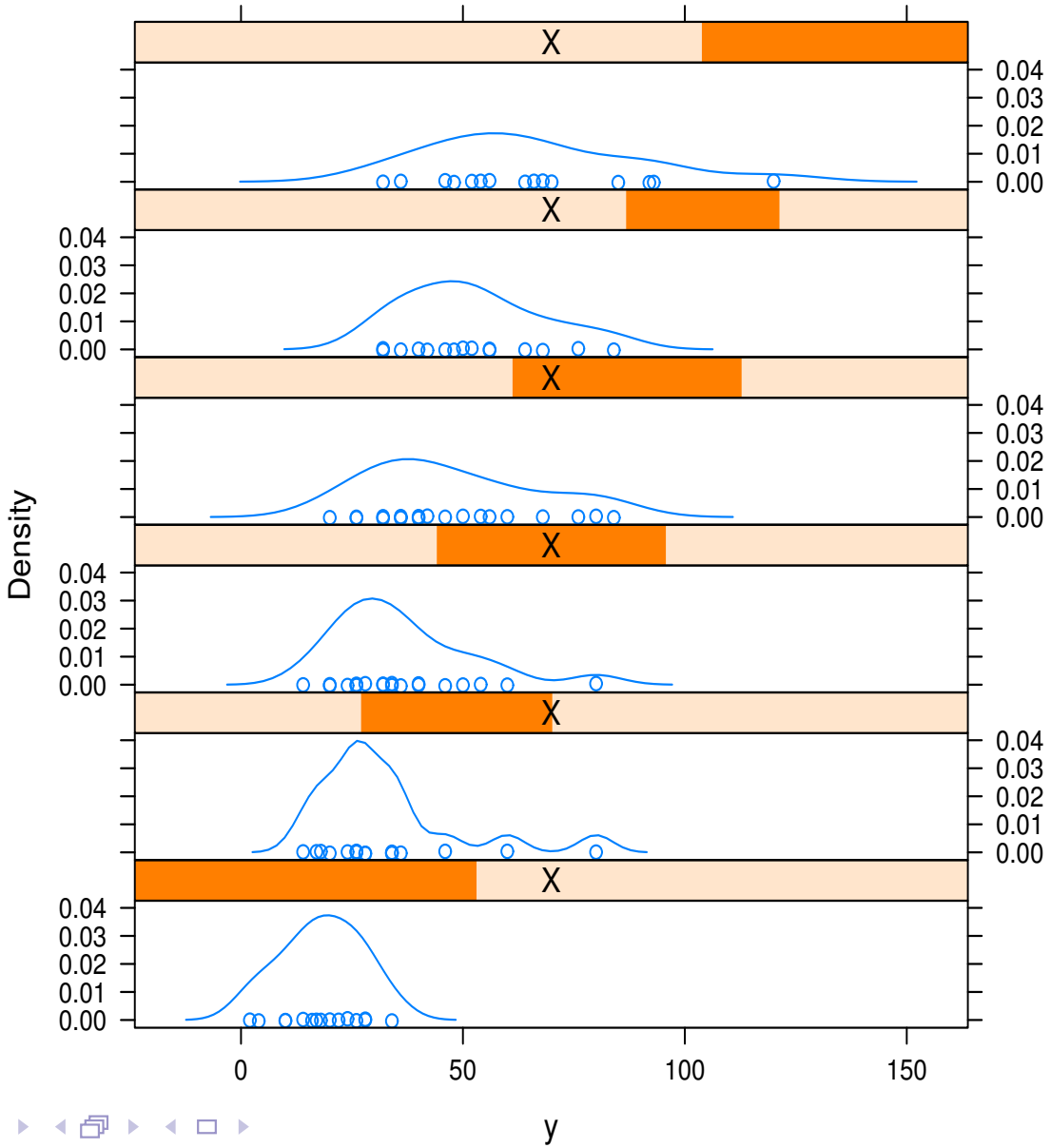
```
bwplot(~ y | vitesse, layout=c(1,6))
```



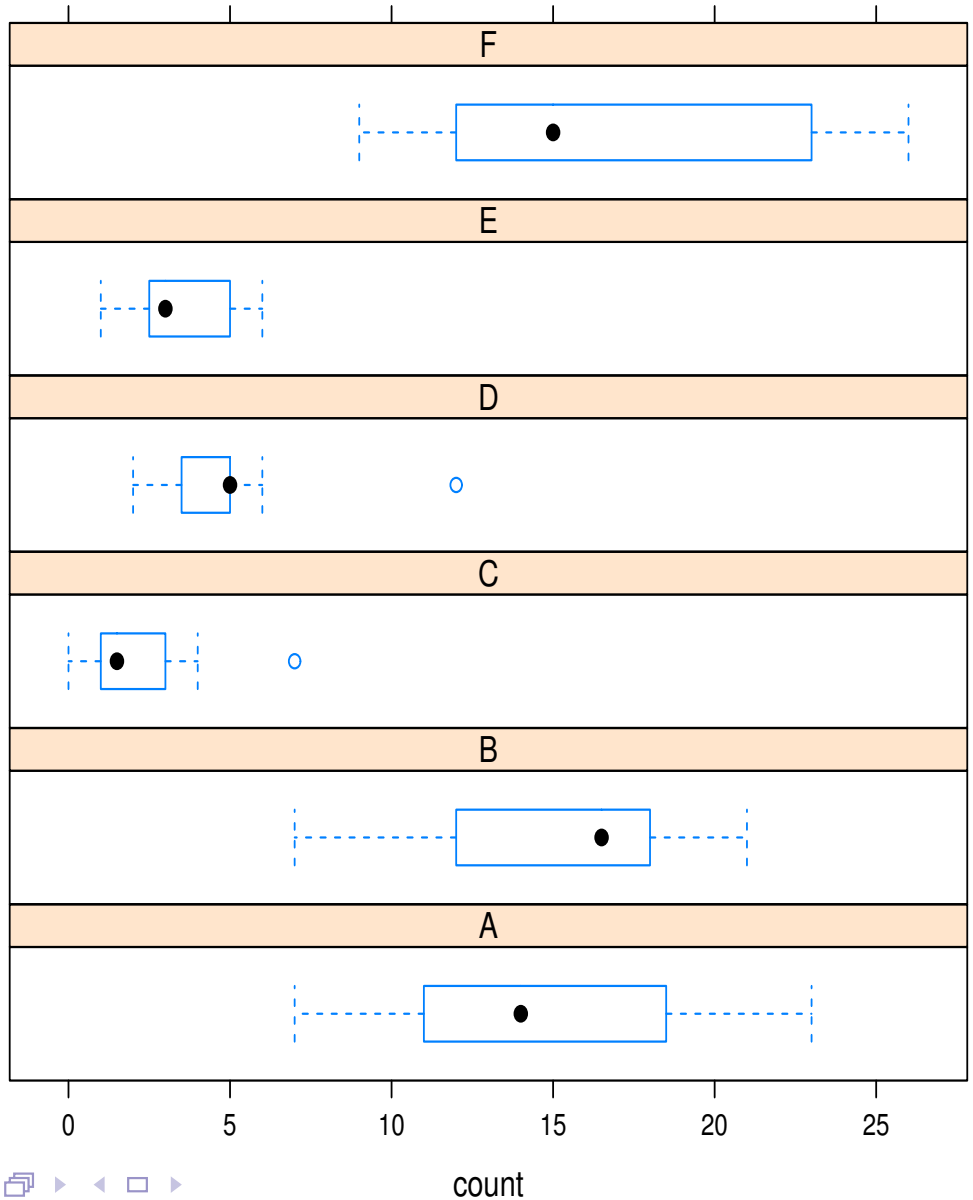
```
densityplot(~ y | X, aspect='xy')
```



```
densityplot(~ y | X, layout=c(1,6))
```



```
bwplot( ~ count | spray, data = InsectSprays, layout=c(1,6))
```



رسم رویه‌ها

چندین بسته در R وجود دارند که رسم رویه‌ها و همچنین نمودارهای تراز را فراهم می‌آورند. با کمک تابع $persp$ می‌توان نمودارهای رویه را روی یک صفحه رسم کرد.

رسم رویه‌ها

چندین بسته در R وجود دارند که رسم رویه‌ها و همچنین نمودارهای تراز را فراهم می‌آورند. با کمک تابع $persp$ می‌توان نمودارهای رویه را روی یک صفحه رسم کرد.

برای مشاهده قابلیت‌های جالب و متنوع این تابع، دستور $demo(persp)$ را در R اجرا کنید.

رسم رویه‌ها

چندین بسته در R وجود دارند که رسم رویه‌ها و همچنین نمودارهای تراز را فراهم می‌آورند. با کمک تابع $persp$ می‌توان نمودارهای رویه را روی یک صفحه رسم کرد.

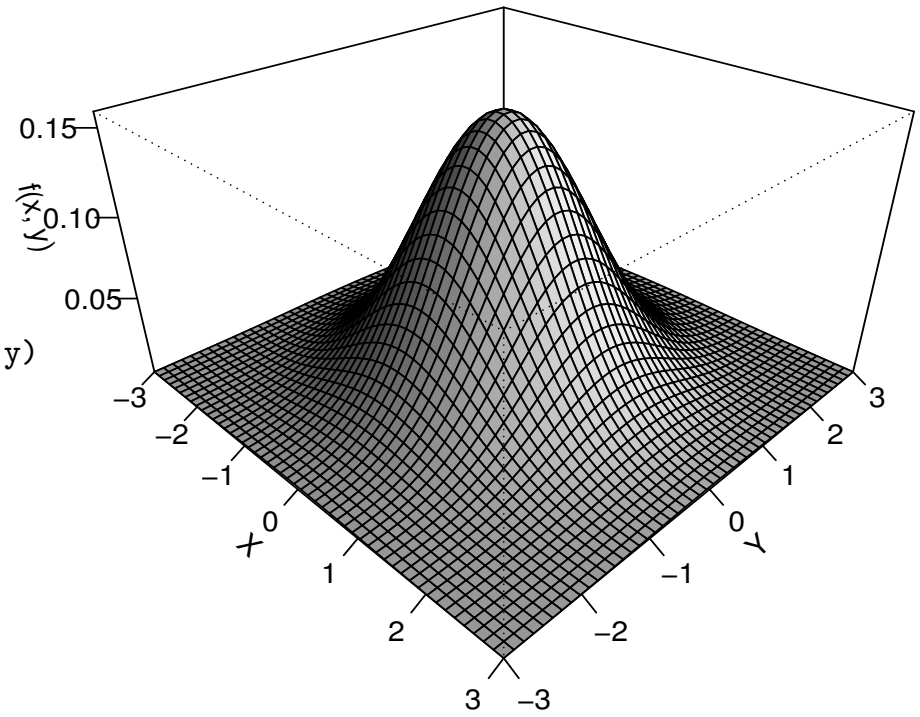
برای مشاهده قابلیت‌های جالب و متنوع این تابع، دستور $demo(persp)$ را در R اجرا کنید.

به عنوان مثال، فرض کنید بخواهیم رویه تابع چگالی نرمال استاندارد دومتغیره را رسم کنیم:

$$f(x,y) = \frac{۱}{۲\pi} e^{-\frac{۱}{۲}(x^۲+y^۲)},\ (x,y) \in \Re^۲.$$

دقت کنید در کد زیر $z_{ij} = f(x_i, y_j)$ با دستور $outer$ محاسبه می‌شود.

```
f <- function(x,y) {
        z <- (1/(2*pi)) * exp(-.5 * (x^2 + y^2))
}
y <- x <- seq(-3, 3, length= 50)
z <- outer(x, y, f)   # compute density for all (x,y)
persp(x, y, z)        # the default plot
persp(x, y, z, theta = 45, phi = 30, expand = 0.6,
ltheta = 120, shade = 0.75, ticktype = "detailed",
xlab = "X", ylab = "Y", zlab = "f(x, y)")
```



رویه‌ها در *lattice*

رویه‌ها را با تابع *wireframe* در بسته *lattice* هم می‌توانید رسم کنید.

رویه‌ها در *lattice*

رویه‌ها را با تابع *wireframe* در بسته *lattice* هم می‌توانید رسم کنید.

برای این کار، باید فرمولی به صورت $z \sim x * y$ به همراه یک ساختار داده یا ماتریس شامل نقاط (x, y, z) آماده کنید.

رویه‌ها در *lattice*

رویه‌ها را با تابع *wireframe* در بسته *lattice* هم می‌توانید رسم کنید.

برای این کار، باید فرمولی به صورت $z \sim x * y$ به همراه یک ساختار داده یا ماتریس شامل نقاط (x, y, z) آماده کنید.

با استفاده از بسته *rgl* می‌توانید یک نمایش سه بعدی تعاملی ایجاد کنید، به طوری که توانایی چرخش نمودار رویه به هر جهت فراهم می‌شود. برای دیدن این قابلیت، از دو خط زیر استفاده کنید

```
library(rgl)
demo(bivar)
```

رویه‌ها در *lattice*

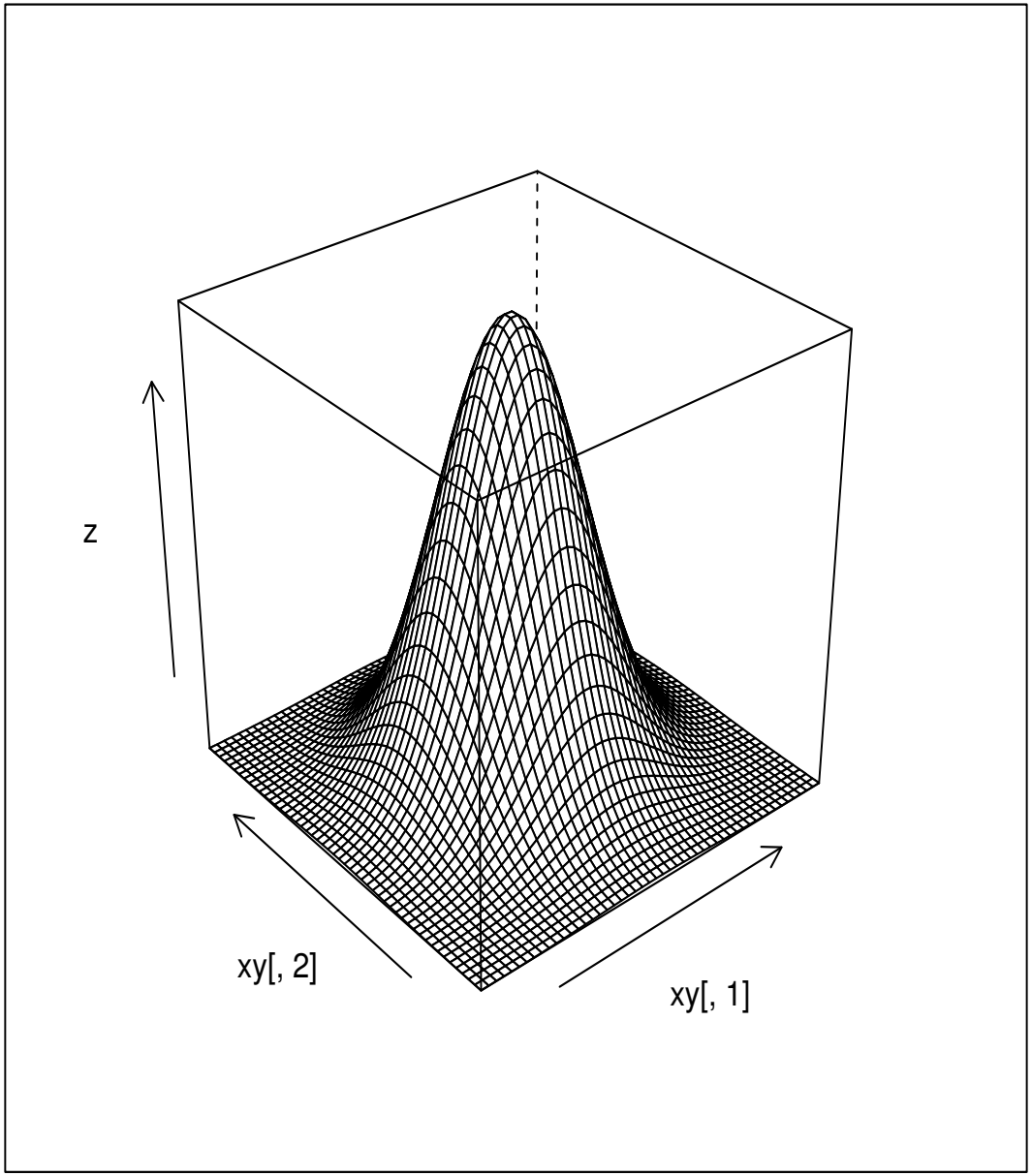
رویه‌ها را با تابع *wireframe* در بسته *lattice* هم می‌توانید رسم کنید.

برای این کار، باید فرمولی به صورت $z \sim x * y$ به همراه یک ساختار داده یا ماتریس شامل نقاط (x, y, z) آماده کنید.

با استفاده از بسته *rgl* می‌توانید یک نمایش سه بعدی تعاملی ایجاد کنید، به طوری که توانایی چرخش نمودار رویه به هر جهت فراهم می‌شود. برای دیدن این قابلیت، از دو خط زیر استفاده کنید

```
library(rgl)
demo(bivar)
```

```
library(lattice)
x <- y <- seq(-3, 3, length= 50)
xy <- expand.grid(x, y)
z <- (1/(2*pi)) * exp(-.5 * (xy[,1]^2 + xy[,2]^2))
wireframe(z ~ xy[,1] * xy[,2])
```


z
xy[, 2]
xy[, 1]

نمودارهای تراز

نمودارهای تراز ($contourplot$) خطوط تراز یک شکل سه‌بعدی را رسم می‌کند. به عبارت دیگر این نمودارها، همان رویه سه بعدی $(x, y, f(x, y))$ را در یک صفحه، با منحنی‌های تراز $f(x, y) = c$ برای ثابت‌های انتخاب شده c، تصویر می‌کند.

نمودارهای تراز

نمودارهای تراز ($contourplot$) خطوط تراز یک شکل سه‌بعدی را رسم می‌کند. به عبارت دیگر این نمودارها، همان رویه سه بعدی $(x, y, f(x, y))$ را در یک صفحه، با منحنی‌های تراز $f(x, y) = c$ برای ثابت‌های انتخاب شده c، تصویر می‌کند.

دو تابع $contour$ در بسته $graphics$ که در خود هسته R وجود دارند و تابع $contourplot$ در بسته $lattice$ این نمودارها را تولید می‌کنند.

نمودارهای تراز

نمودارهای تراز ($contourplot$) خطوط تراز یک شکل سه‌بعدی را رسم می‌کند. به عبارت دیگر این نمودارها، همان رویه سه بعدی $(x, y, f(x, y))$ را در یک صفحه، با منحنی‌های تراز $f(x, y) = c$ برای ثابت‌های انتخاب شده c، تصویر می‌کند.

دو تابع $contour$ در بسته $graphics$ که در خود هسته R وجود دارند و تابع $contourplot$ در بسته $lattice$ این نمودارها را تولید می‌کنند.

تابع $image$ تابعی هم‌جنس $contour$ است که از رنگ‌ها برای تشخیص ترازها استفاده می‌کند.

نمودارهای تراز

نمودارهای تراز ($contourplot$) خطوط تراز یک شکل سه‌بعدی را رسم می‌کند. به عبارت دیگر این نمودارها، همان رویه سه بعدی $(x, y, f(x, y))$ را در یک صفحه، با منحنی‌های تراز $f(x, y) = c$ برای ثابت‌های انتخاب شده c، تصویر می‌کند.

دو تابع $contour$ در بسته $graphics$ که در خود هسته R وجود دارند و تابع $contourplot$ در بسته $lattice$ این نمودارها را تولید می‌کنند.

تابع $image$ تابعی هم‌جنس $contour$ است که از رنگ‌ها برای تشخیص ترازها استفاده می‌کند.

یک مثال مناسب برای نمایش این نمودارها در R بر اساس داده‌های $volcano$ ارایه شده است. برای اطلاع از این داده‌ها می‌توانید از $help$ نرم‌افزار R کمک بگیرید.

نمودارهای تراز

نمودارهای تراز ($contourplot$) خطوط تراز یک شکل سه‌بعدی را رسم می‌کند. به عبارت دیگر این نمودارها، همان رویه سه بعدی $(x, y, f(x, y))$ را در یک صفحه، با منحنی‌های تراز $f(x, y) = c$ برای ثابت‌های انتخاب شده c، تصویر می‌کند.

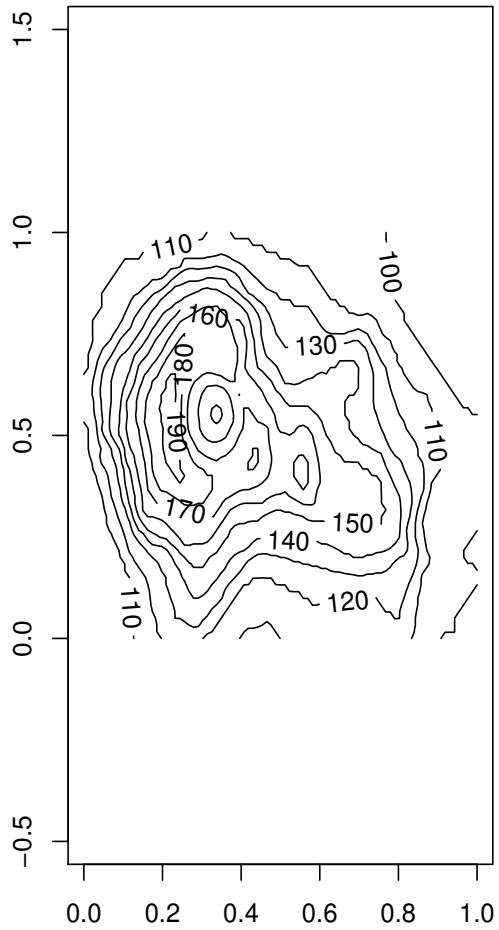
دو تابع $contour$ در بسته $graphics$ که در خود هسته R وجود دارند و تابع $contourplot$ در بسته $lattice$ این نمودارها را تولید می‌کنند.

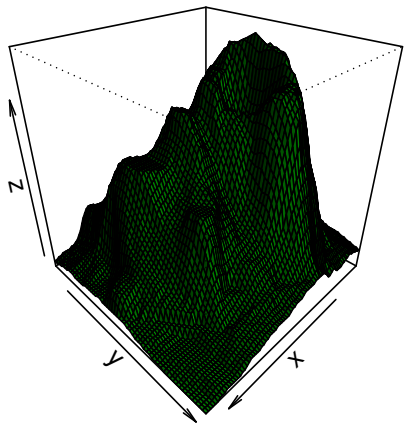
تابع $image$ تابعی هم‌جنس $contour$ است که از رنگ‌ها برای تشخیص ترازها استفاده می‌کند.

یک مثال مناسب برای نمایش این نمودارها در R بر اساس داده‌های $volcano$ ارایه شده است. برای اطلاع از این داده‌ها می‌توانید از $help$ نرم‌افزار R کمک بگیرید.

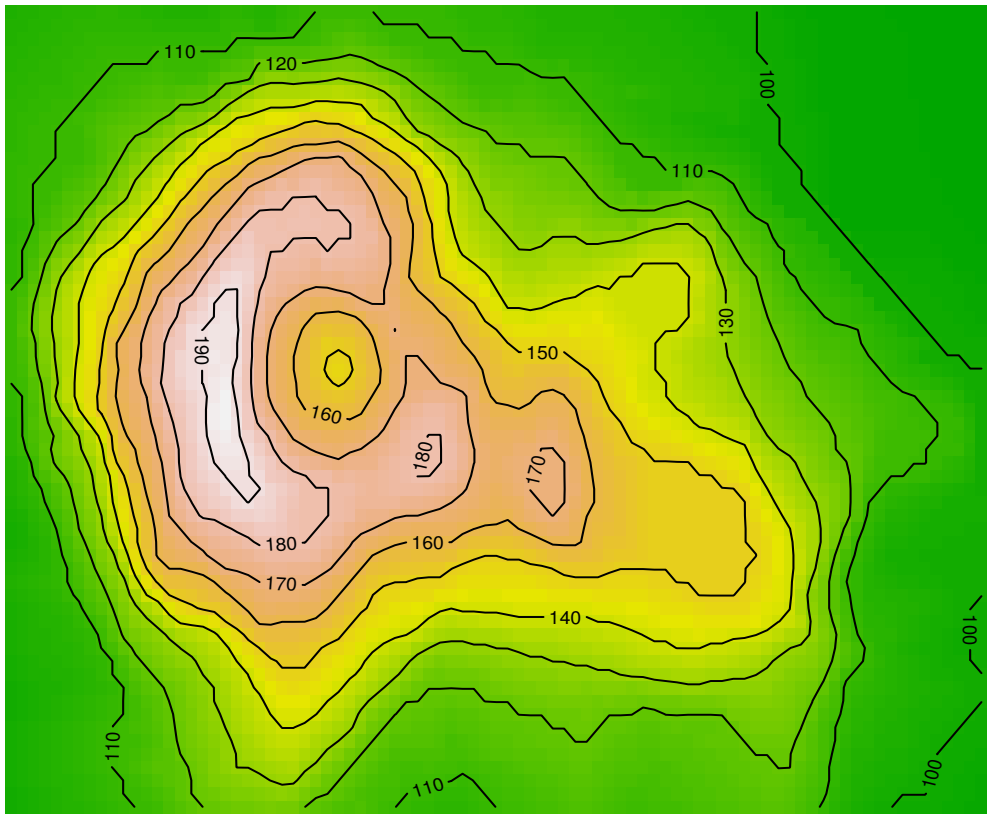
توابع $filled.contour$ در $graphics$ و $levelplot$ در $lattice$، نمودارهای تراز پرشده (رنگی) را تولید می‌کنند. طیف رنگی موجود در این نمودارها، مشابه $image$، ارتفاع نمودار $f(x, y)$ را نمایش می‌دهند.

```
# contour plot with labels
contour(volcano, asp = 1, labcex = 1)
# another version from lattice package
library(lattice)
contourplot(volcano) # similar to above
```

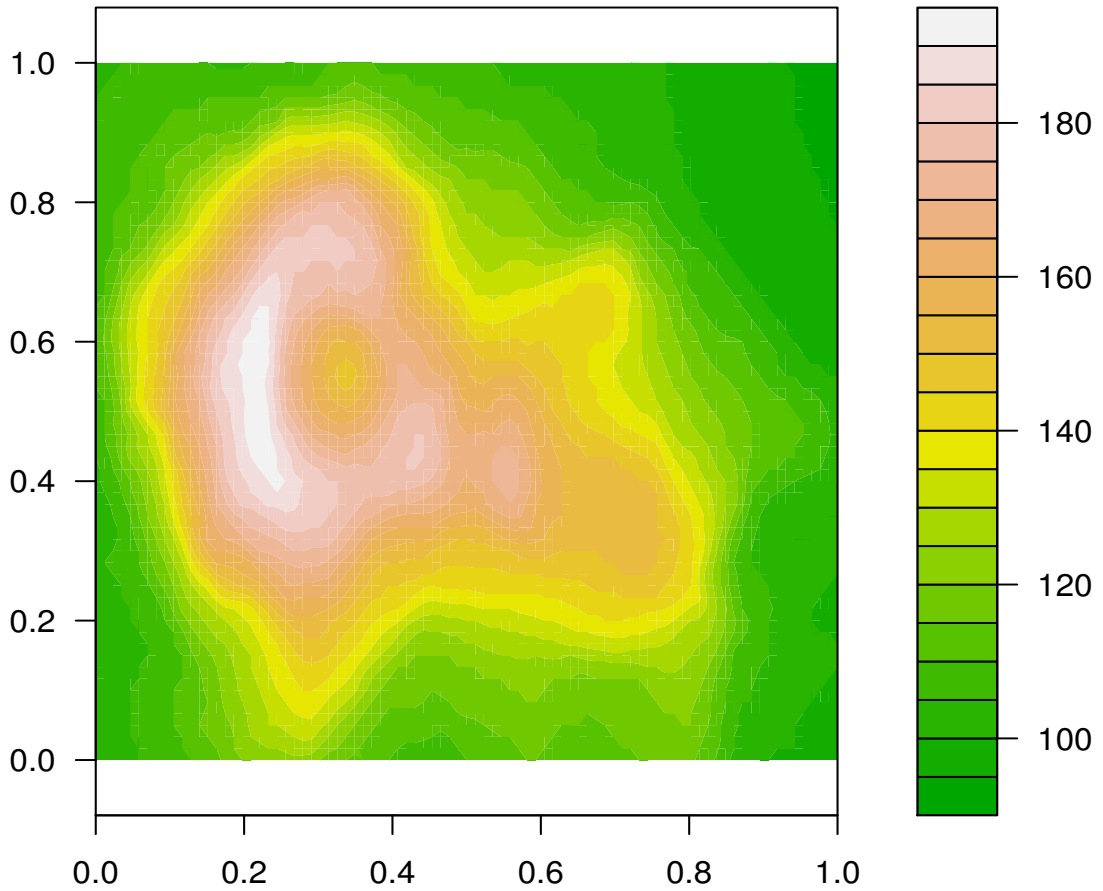




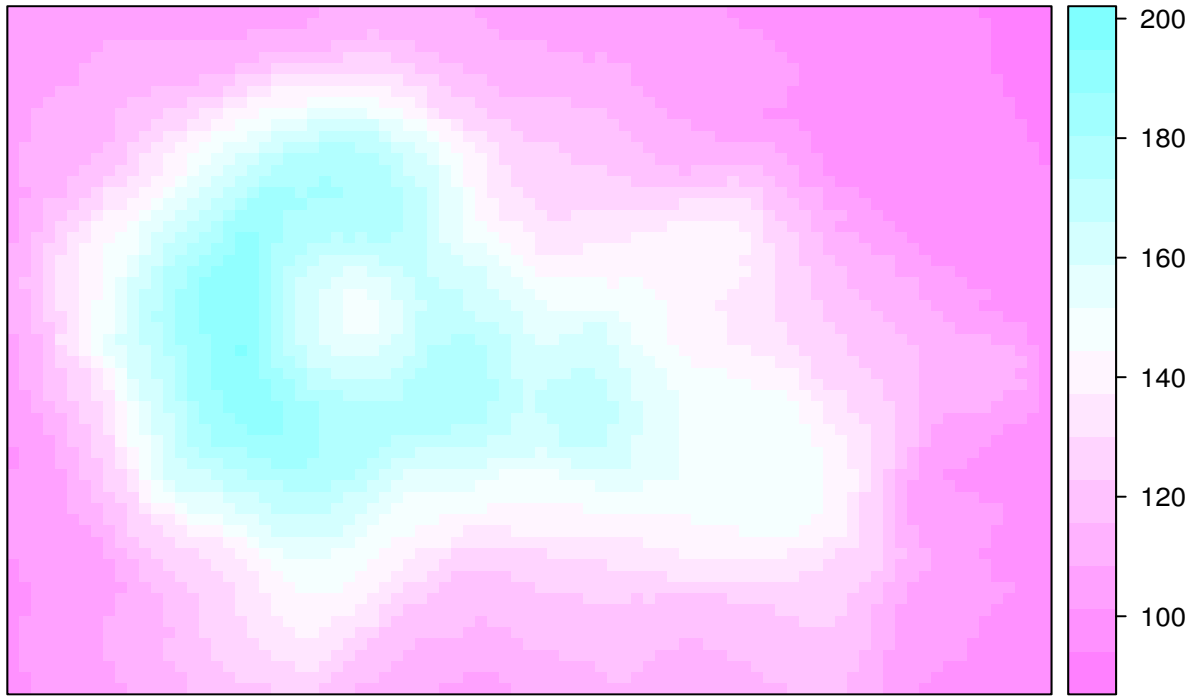
```
image(volcano, col = terrain.colors(100), axes = FALSE)
contour(volcano, levels = seq(100,200,by = 10), add = TRUE)
```



```
filled.contour(volcano, color = terrain.colors, asp = 1)
```



```
levelplot(volcano, scales = list(draw = FALSE),
                   xlab = "", ylab = "")
```



تابع *image*

از این نوع توابع، می‌توان برای بررسی وجود گروه‌هایی در داده‌ها نیز استفاده کرد.

تابع *image*

از این نوع توابع، می‌توان برای بررسی وجود گروه‌هایی در داده‌ها نیز استفاده کرد.

از آن‌ها در خوشه‌بندی داده‌ها و متغیرها نیز استفاده می‌شود.

تابع *image*

از این نوع توابع، می‌توان برای بررسی وجود گروه‌هایی در داده‌ها نیز استفاده کرد.

از آن‌ها در خوشه‌بندی داده‌ها و متغیرها نیز استفاده می‌شود.

مشابه تابع *image*، تابع *image.plot* در بسته *fields* وجود دارد با این مزیت که طیف رنگی را نیز به همراه نمودار، بر اساس مقیاس داده‌ها، مدرج‌بندی می‌کند.

تابع *image*

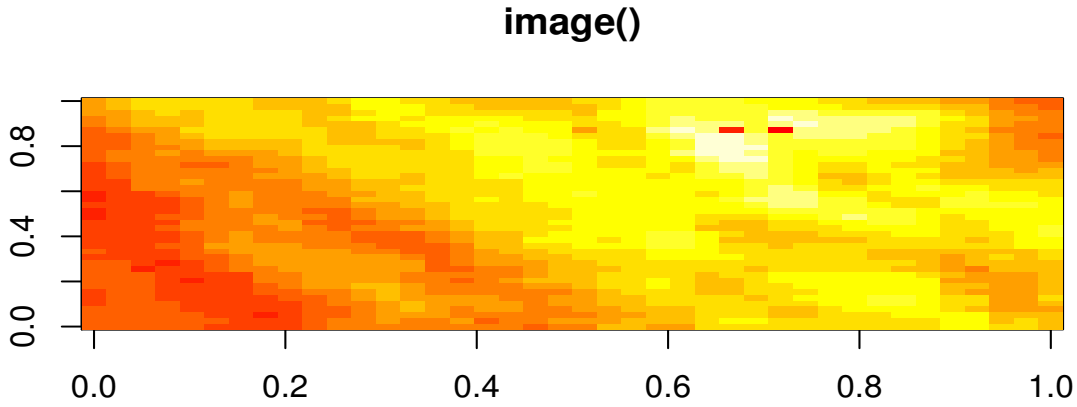
از این نوع توابع، می‌توان برای بررسی وجود گروه‌هایی در داده‌ها نیز استفاده کرد.

از آن‌ها در خوشه‌بندی داده‌ها و متغیرها نیز استفاده می‌شود.

مشابه تابع *image*، تابع *image.plot* در بسته *fields* وجود دارد با این مزیت که طیف رنگی را نیز به همراه نمودار، بر اساس مقیاس داده‌ها، مدرج‌بندی می‌کند.

از این تابع به ویژه در کاربردهای آمار فضایی، برای مشاهده همگنی متغیرهای اندازه‌گیری شده در نواحی جغرافیایی استفاده می‌شود.

```
library(fields)
data(lennon)
x <- lennon[201:240,201:240]
par(mfrow=c(2,1))
image(x, main="image()")
image.plot(x, main="image.plot()")
```


image()
0.0
0.4
0.8
0.0
0.2
0.4
0.6
0.8
1.0

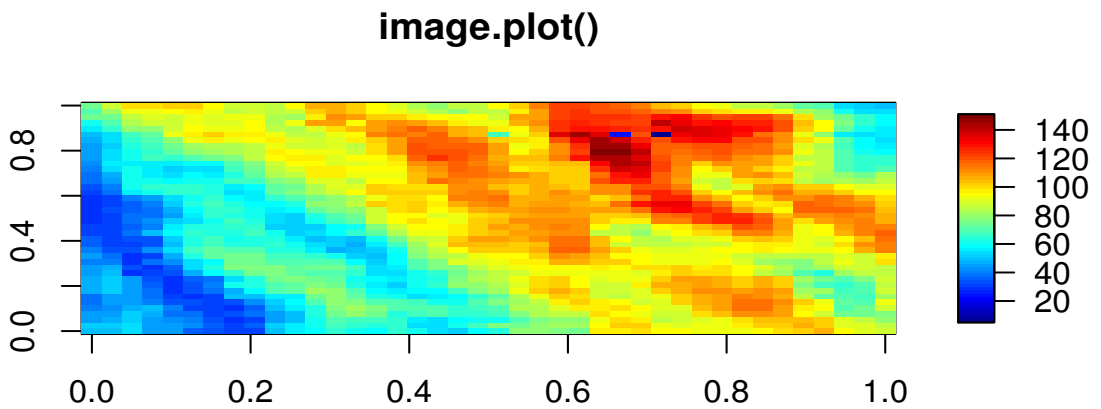


image.plot()
0.0
0.4
0.8
0.0
0.2
0.4
0.6
0.8
1.0
140
120
100
80
60
40
20

نمودارهای موازی

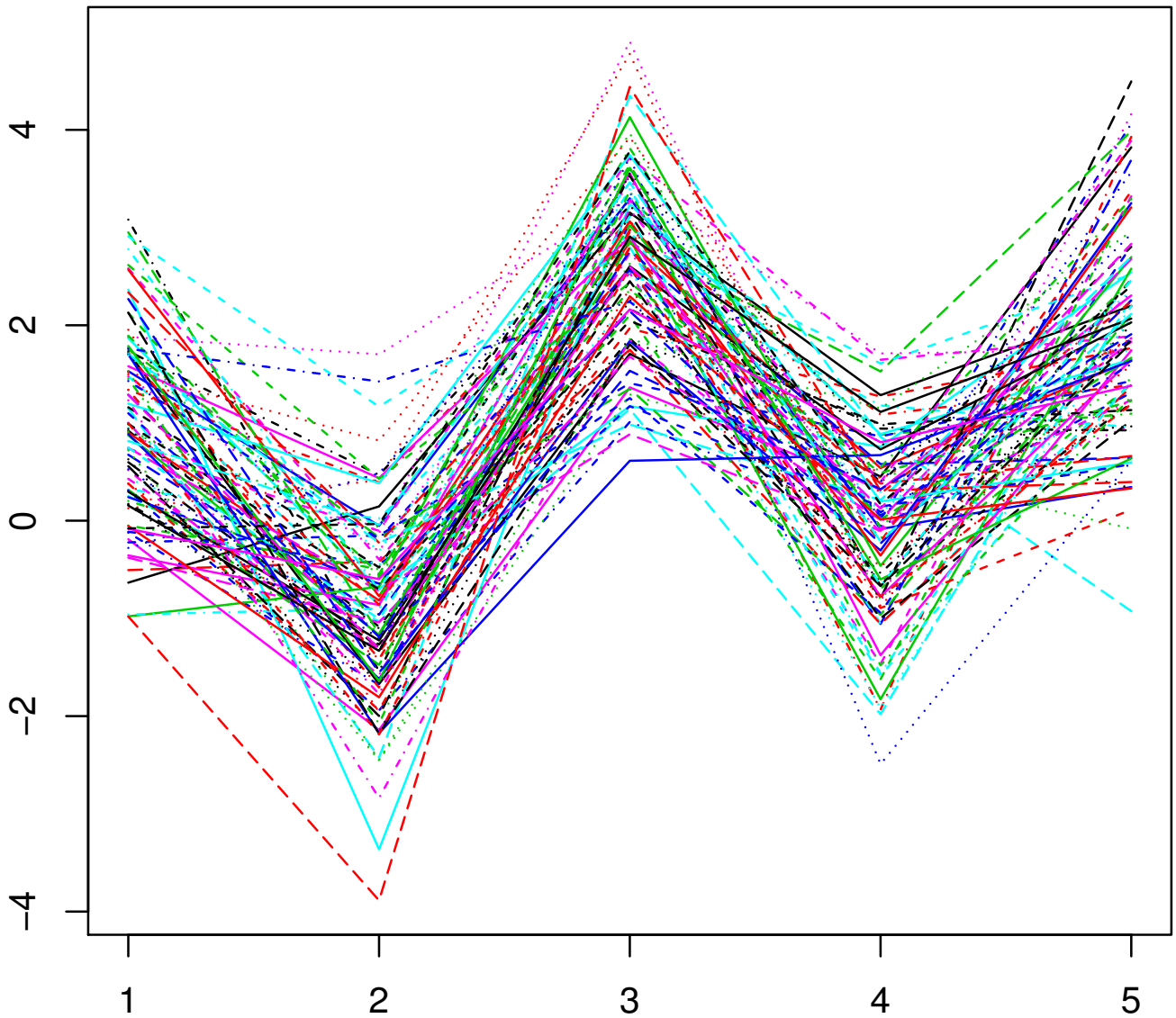
از این نوع نمودارها نیز می‌توان برای پی بردن به وجود گروه‌هایی در داده‌ها، بهره برد.

در سیستم مختصات موازی، به جای نمایش محورهای مختصات به صورت عمود بر هم، آن‌ها را به صورت محورهایی با خطوط موازی و فاصله یکسان نمایش می‌دهند.

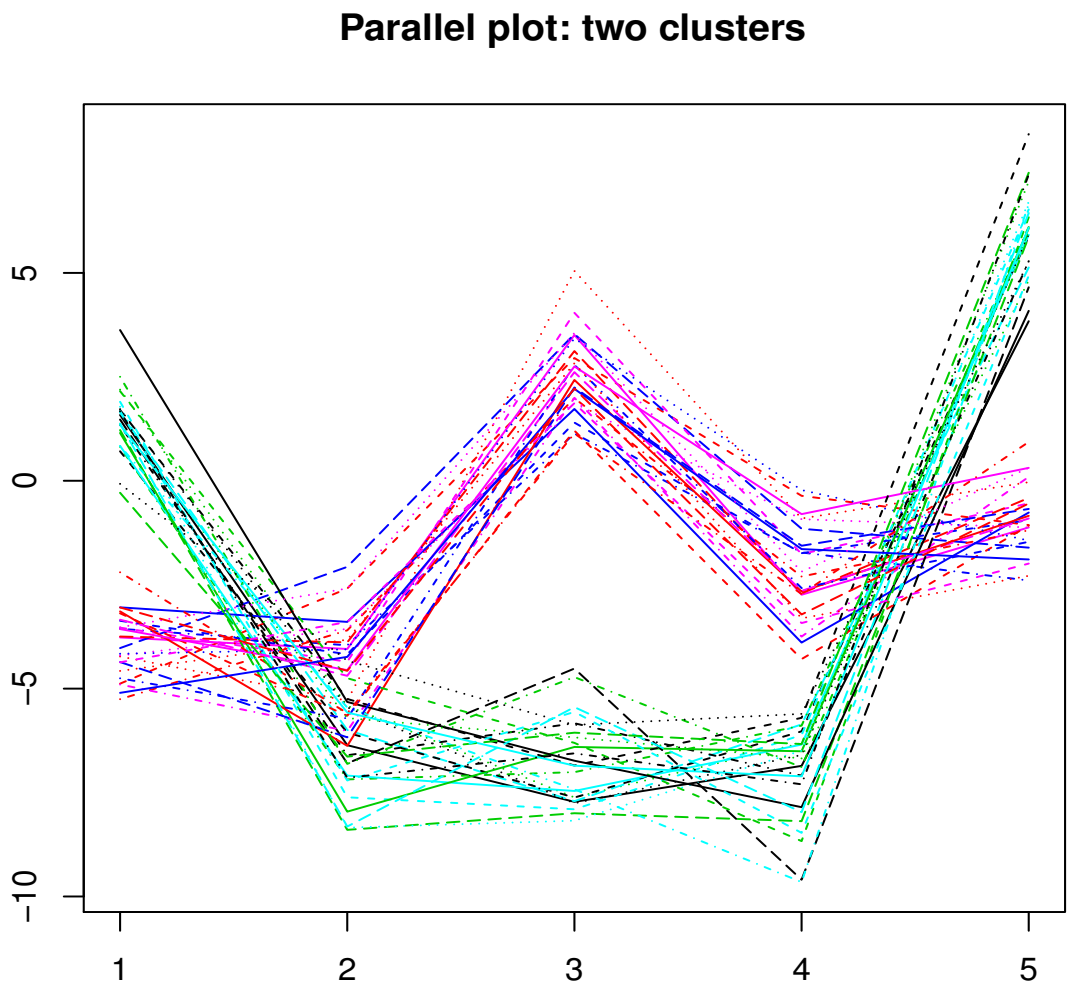
بنابراین در مختصات موازی به جای نمایش بردارها در فضای $\Re^d$، یک بردار را به صورت d خط حقیقی (با طول‌های برابر) نشان می‌دهند و آن‌ها را به هم وصل می‌کنند.

```
n <- 100
m <- matrix( rnorm(5*n)+c(1,-1,3,0,2),
             nr = n, nc = 5, byrow = TRUE )
matplot(1:5, t(m), type = 'l',
        xlab = "", ylab = "")
title(main = "Parallel plot: Homogeneous cloud")
```

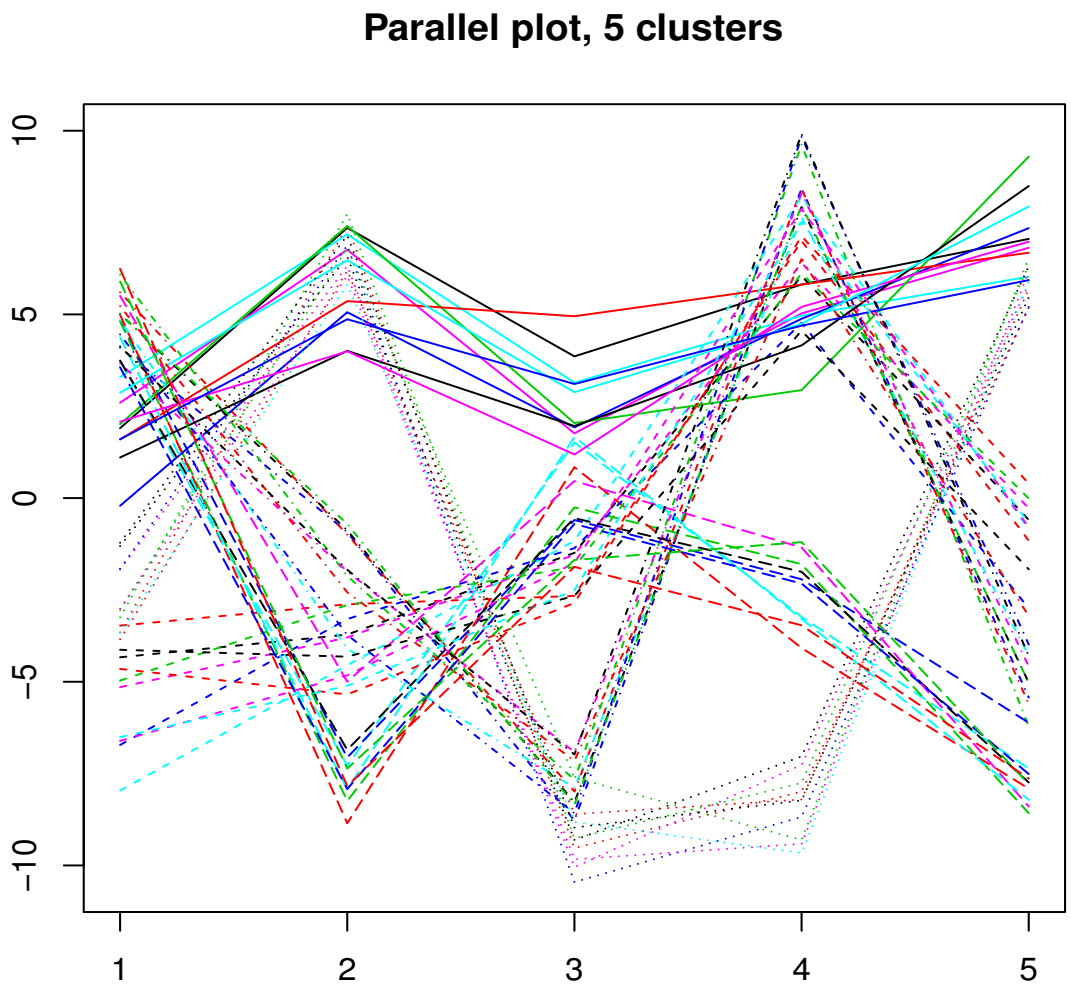
Parallel plot: Homogeneous cloud



```
n <- 50
k <- 2
m <- matrix( rnorm(5*k*n) +
               runif(5*k, min = -10, max = 10),
             nr = n, nc = 5, byrow = TRUE )
matplot(1:5, t(m), type = 'l', xlab = "", ylab = "")
title(main = "Parallel plot: two clusters")
```

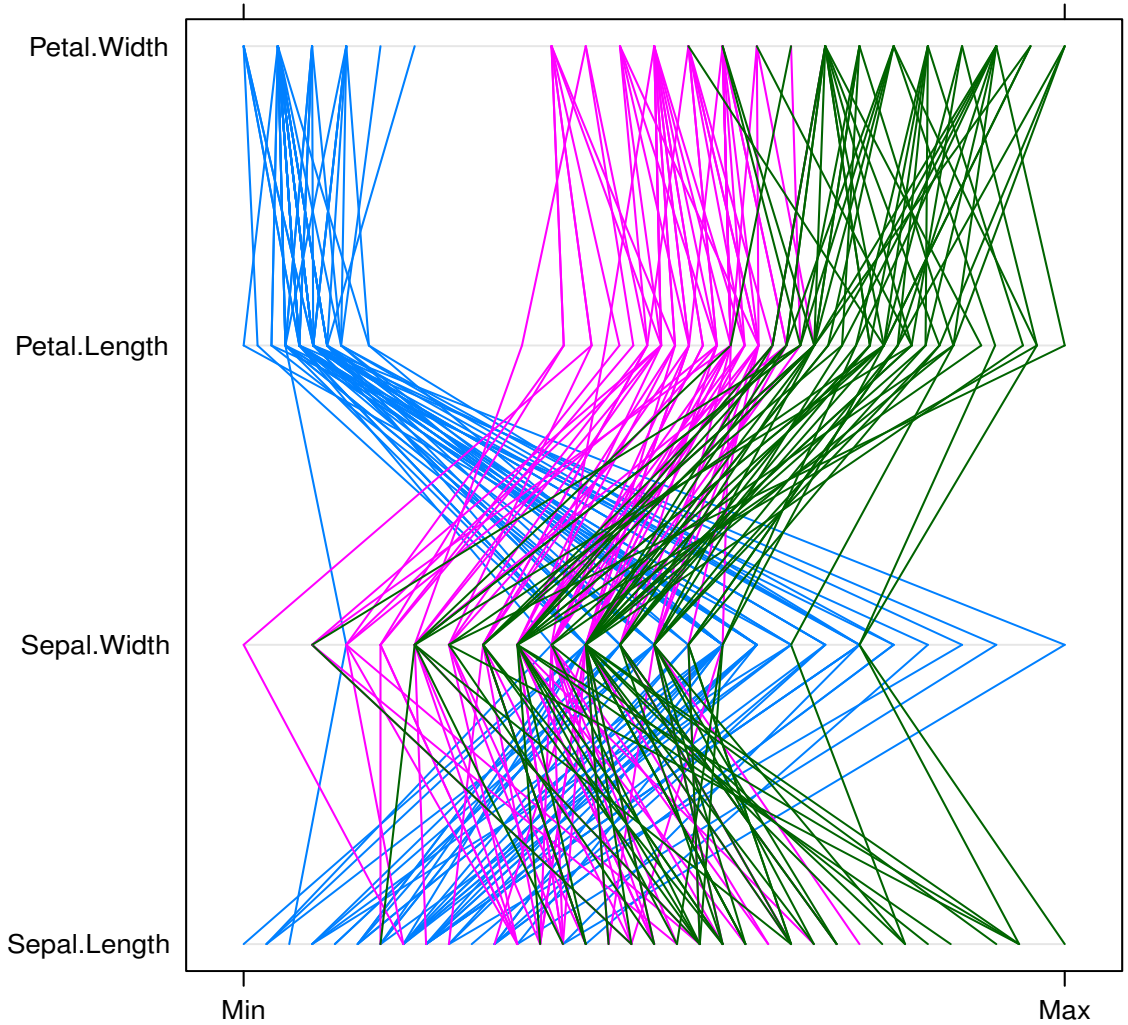


```
n <- 50
k <- 5
m <- matrix( rnorm(5*k*n) +
               runif(5*k, min = -10, max = 10),
             nr = n, nc = 5, byrow = TRUE )
matplot(1:5, t(m), type = 'l', xlab = "", ylab = "")
title(main = "Parallel plot, 5 clusters")
```



از توابع *parcoord* در *MASS* و *parallel* در *lattice* هم می‌توان برای تولید این نمودارها استفاده کرد.

```
library(lattice)
parallel(~iris[1:4], groups = Species, iris)
```



منحنی‌های اندروز

یک رهیافت دیگر برای نمایش داده‌های چندبعدی، مانند $X_1, X_2, \ldots, X_n \in \Re^d$، در یک شکل دوبعدی، نگاشتن هر بردار به یک تابع حقیقی مقدار است.

منحنی‌های اندروز $(Andrews)$ هر مشاهده $x_i = (x_{i1}, \ldots, x_{id})$ را به تابع زیر تصویر می‌کند:

$$f_i(t) = \frac{x_{i1}}{\sqrt{2}} + x_{i2} \sin t + x_{i3} \cos t + x_{i4} \sin 2t + x_{i5} \cos 2t + \cdots$$

یا به عبارت خلاصه‌تر:

$$f_i(t) = \frac{x_{i1}}{\sqrt{2}} + \sum_{1 \le k \le d/2} x_{i,2k} \sin 2k + \sum_{1 \le k \le d/2} x_{i,2k+1} \cos 2k, \quad -\pi \le t \le \pi.$$

منحنی‌های اندروز

یک رهیافت دیگر برای نمایش داده‌های چندبعدی، مانند $X_1, X_2, \ldots, X_n \in \Re^d$، در یک شکل دوبعدی، نگاشتن هر بردار به یک تابع حقیقی مقدار است.

منحنی‌های اندروز ($Andrews$) هر مشاهده $x_i = (x_{i1}, \ldots, x_{id})$ را به تابع زیر تصویر می‌کند:

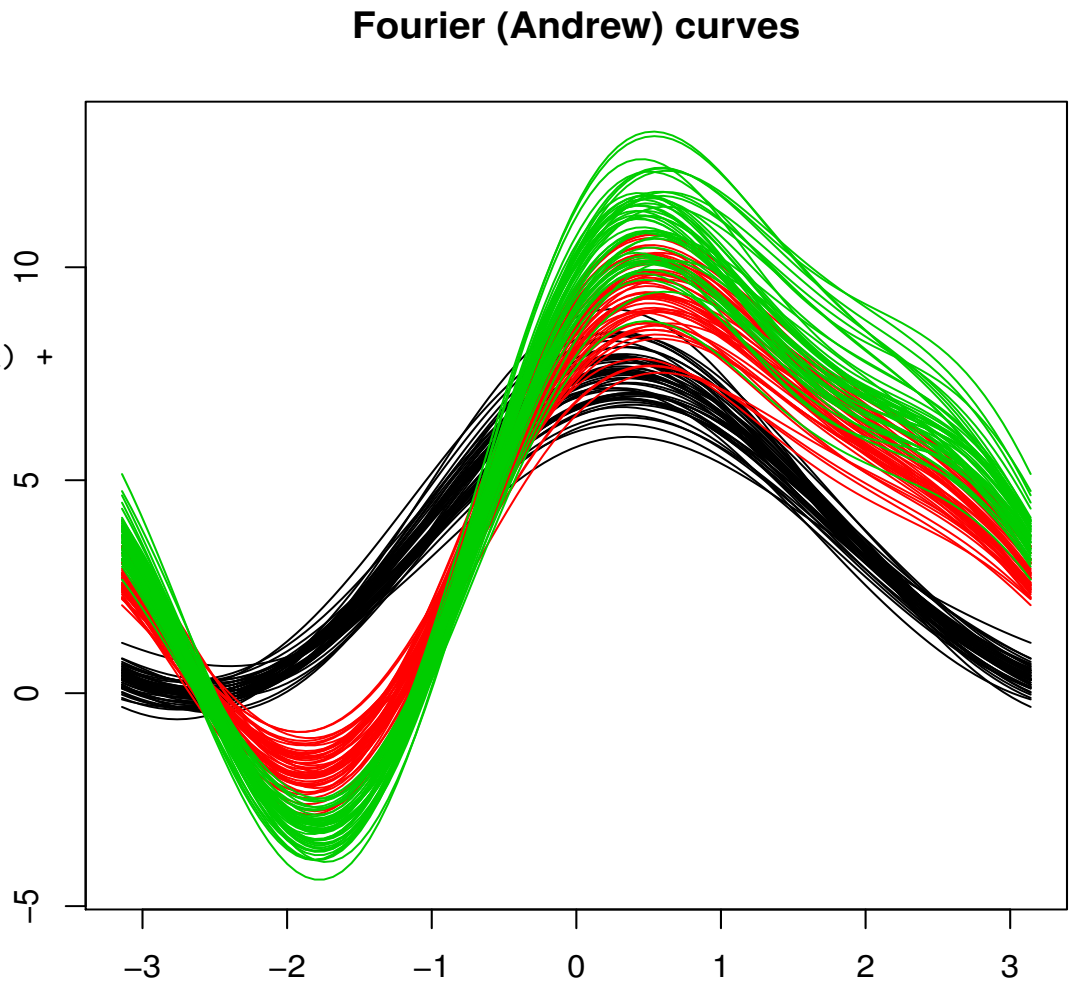
$$f_i(t) = \frac{x_{i1}}{\sqrt{2}} + x_{i2} \sin t + x_{i3} \cos t + x_{i4} \sin 2t + x_{i5} \cos 2t + \cdots$$

یا به عبارت خلاصه‌تر:

$$f_i(t) = \frac{x_{i1}}{\sqrt{2}} + \sum_{1 \le k \le d/2} x_{i,2k} \sin 2k + \sum_{1 \le k \le d/2} x_{i,2k+1} \cos 2k, \quad -\pi \le t \le \pi.$$

در واقع هر مشاهده توسط تصویرش بر مجموعه توابع پایه متعامد $\{2^{-1/2}, \{\sin kt\}_{k=1}^{\infty}, \{\cos kt\}_{k=1}^{\infty}\}$ نمایش داده می‌شود.

```
x <- seq(-pi, pi, length=100)
f <-  function (u) u[1]/sqrt(2) + u[2]*cos(x) +
u[3] * sin(x) + u[4] * cos(2*x)
y <- apply(as.matrix(iris[,1:4]), 1, f)
matplot(x, y,
        type = "l",
        lty = 1,
        col = as.numeric(iris[,5]),
        xlab = "", ylab = "",
        main = "Fourier (Andrew) curves")
```



محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه هفتم: روش‌های مونت کارلو (انتگرال‌گیری مونت کارلویی)

حسین باغیشنی

دانشگاه شاهرود

۱۶ آبان ۱۳۹۳

مقدمه

دو رده اصلی از مسایل استنباط‌های آماری عبارتند از:

- بهینه‌سازی: که معمولا با رهیافت‌های مبتنی بر درستنمایی همراه است
- انتگرال‌گیری: که معمولا با رهیافت بیزی همراه است

مقدمه

دو رده اصلی از مسایل استنباط‌های آماری عبارتند از:

- بهینه‌سازی: که معمولا با رهیافت‌های مبتنی بر درستنمایی همراه است
- انتگرال‌گیری: که معمولا با رهیافت بیزی همراه است

در استنباط‌های آماری، اغلب مواقع راه حل مساله منتهی به حل عددی یک انتگرال یا یک بهینه‌سازی می‌شود.

مثال: نظریه تصمیم بیزی

برآوردگرهای بیزی همیشه امید ریاضی توزیع پسین نیستند. اما جواب انتگرال زیر هستند:

$$\min_{\delta} \int_{\Theta} L(\theta, \delta)\pi(\theta)f(x|\theta)d\theta.$$

- زیان توان دوم خطا: برای $L(\theta, \delta) = (\theta - \delta)^2$، برآوردگر بیزی میانگین توزیع پسین است
- زیان قدر مطلق خطا: برای $L(\theta, \delta) = |\theta - \delta|$، برآوردگر بیزی میانه توزیع پسین است
- بدون تابع زیان: از MAP استفاده می‌کنیم که عبارتست از $\arg\max_\theta \ell(\theta|x)\pi(\theta)$.

روش‌های انتگرال‌گیری عددی

انتگرالی مانند $\mathcal{I} = \int_{\chi} h(x)f(x)dx$ را می‌توان به کمک روش‌های عددی مانند قاعده سیمپسون یا ذوزنقه، محاسبه کرد.

روش‌های انتگرال‌گیری عددی

انتگرالی مانند $\mathcal{I} = \int_{\chi} h(x)f(x)dx$ را می‌توان به کمک روش‌های عددی مانند قاعده سیمپسون یا ذوزنقه، محاسبه کرد.

به عنوان مثال، R برای محاسبه انتگرال‌های یک بعدی دو تابع $area$ و $integrate$ را دارد.
اما تابع $area$ برای انتگرال‌های با بعد بی‌کران قابل استفاده نیست.

روش‌های انتگرال‌گیری عددی

انتگرالی مانند $\mathcal{I} = \int_{\chi} h(x)f(x)dx$ را می‌توان به کمک روش‌های عددی مانند قاعده سیمپسون یا ذوزنقه، محاسبه کرد.

به عنوان مثال، R برای محاسبه انتگرال‌های یک بعدی دو تابع $area$ و $integrate$ را دارد. اما تابع $area$ برای انتگرال‌های با بعد بی‌کران قابل استفاده نیست.

تابع $integrate$ برای انتگرال‌های با بعد بی‌کران قابل استفاده است، اما جواب‌های قابل اعتمادی نتیجه نمی‌دهد.

روش‌های انتگرال‌گیری عددی

انتگرالی مانند $\mathcal{I} = \int_{\chi} h(x)f(x)dx$ را می‌توان به کمک روش‌های عددی مانند قاعده سیمپسون یا ذوزنقه، محاسبه کرد.

به عنوان مثال، R برای محاسبه انتگرال‌های یک بعدی دو تابع $area$ و $integrate$ را دارد. اما تابع $area$ برای انتگرال‌های با بعد بی‌کران قابل استفاده نیست.

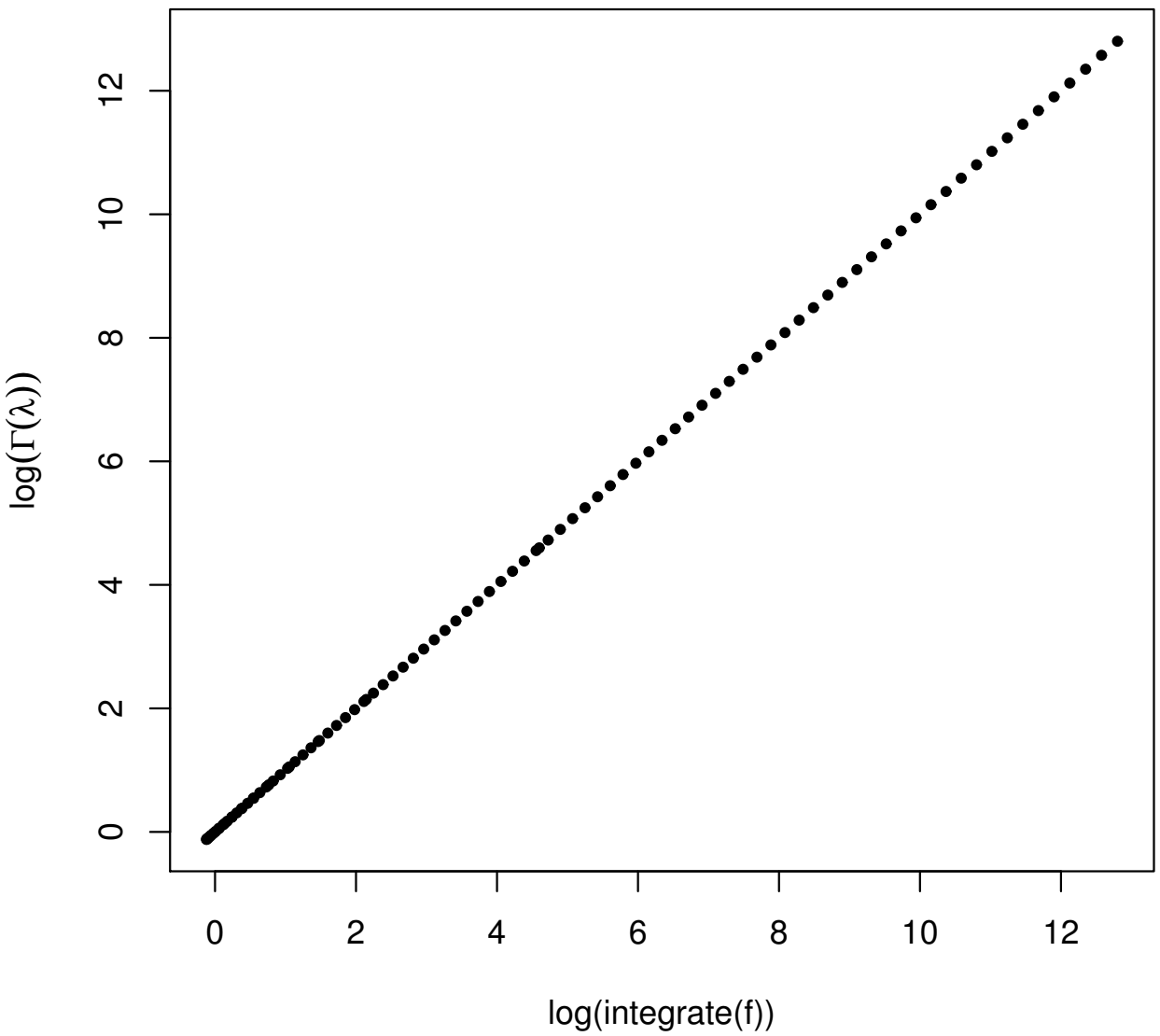
تابع $integrate$ برای انتگرال‌های با بعد بی‌کران قابل استفاده است، اما جواب‌های قابل اعتمادی نتیجه نمی‌دهد.

به طور کلی، روش‌های عددی انتگرال‌گیری برای محاسبه انتگرال‌های با بعد بالا (که در روش‌های آمار نیز خیلی معمول هستند)، مناسب نیستند.

مثال

به عنوان مثال، محاسبه انتگرال $\int_0^\infty x^{\lambda-1}e^{-x}dx$ را با کمک تابع $integrate$ در نظر بگیرید و با تابع $gamma$ مقایسه کنید.

```
ch=function(la){
   integrate(function(x){x^(la-1)*exp(-x)},0,Inf)$val}
plot(lgamma(seq(.01,10,le=100)),log(apply(as.matrix(
seq(.01,10,le=100)),1,ch)),xlab="log(integrate(f))",
ylab=expression(log(Gamma(lambda))),pch=19,cex=.6)
```


log(Γ(λ))
0
2
4
6
8
10
12
log(integrate(f))

مشکل انتگرال‌گیری‌های عددی

مشکل اساسی با روش‌های انتگرال‌گیری عددی (مانند روش موجود در تابع $integrate$) آن است که در اکثر موارد، این روش‌ها قادر به شناسایی نواحی مهم (نواحی چگال‌تر) برای تابعی که باید انتگرال‌گیری شود، نیستند. اما روش‌های شبیه‌سازی با هدف قرار دادن این نواحی، به کمک تابع چگالی f، چنین مشکلی را ندارند.

پروژه: روش‌های معمول انتگرال‌گیری عددی مانند سیمپسون، ذوزنقه و غیره

انتگرال‌گیری مونت کارلویی

مساله:
مساله کلی، محاسبه انتگرال

$$\mathcal{I} = \mathbb{E}_f\left[h(X)\right] = \int_{\chi} h(x)f(x)dx,$$

است که در آن χ یک یا چندبعدی است. f یک تابع چگالی، دارای فرم بسته یا به طور جزیی بسته است. و h یک تابع می‌باشد.

انتگرال‌گیری مونت کارلویی: ادامه

جواب انتگرال به روش مونت کارلو:
از نمونه $(X_1, \ldots, X_m)$ حاصل از توزیع f برای تقریب انتگرال $\mathcal{I}$ به وسیله میانگین نمونه استفاده می‌کنیم:

$$\bar{h}_m = \frac{1}{m} \sum_{j=1}^{m} h(x_j),$$

به طوری که بنا بر قانون قوی اعداد بزرگ، به مقدار انتگرال میل می‌کند

$$\bar{h}_m \longrightarrow \mathbb{E}_f(h(X)).$$

دقت برآورد مونت کارلویی

واریانس برآوردگر را با کمیت

$$\nu_m = \frac{1}{m(m-1)} \sum_{j=1}^{m} [h(x_j) - \bar{h}_m]^2,$$

برآورد می‌کنیم و برای m بزرگ:

$$\frac{\bar{h}_m - \mathbb{E}_f(h(X))}{\sqrt{\nu_m}} \sim N(0,1).$$

توجه: این نتیجه می‌تواند منتهی به آزمونی برای همگرایی برآورد و ساختن کران‌های اطمینان برای تقریب انتگرال شود

مثال

یک برآورد مونت کارلویی برای کمیت $\theta = \int_0^1 e^{-x} dx$ به دست آورید و با مقدار دقیق آن مقایسه کنید.

مثال

یک برآورد مونت کارلویی برای کمیت $\theta = \int_0^1 e^{-x} dx$ به دست آورید و با مقدار دقیق آن مقایسه کنید.

اگر در این مثال، f را یکنواخت استاندارد در نظر بگیریم، می‌توان با نوشتن کد زیر برآوردی برای θ به دست آورد.

```
m <- 10000
x <- runif(m)
theta.hat <- mean(exp(-x))
> print(theta.hat)
[1] 0.6327615
> print(1 - exp(-1))
[1] 0.6321206
```

کران‌های متفاوت

اگر انتگرال به شکل $\int_a^b h(x)dx$ باشد، می‌توان با تغییر متغیر حدود را به ۰ تا ۱ تغییر داد. تبدیل لازم به صورت زیر است:

$$\begin{aligned} y &= \frac{x-a}{b-a}, \\ dy &= (\frac{1}{b-a})dx, \end{aligned}$$

بنابراین:

$$\int_a^b h(x)dx = \int_0^1 h(y(b-a)+a)(b-a)dy.$$

البته به جای توزیع $U(0,1)$ برای f، می‌توان از توزیع $U(a,b)$ نیز استفاده کرد. در این صورت:

$$\int_a^b h(x)dx = (b-a)\int_a^b h(x)\frac{1}{b-a}dx.$$

مثال

یک برآورد مونت کارلویی برای کمیت $\theta = \int_{۲}^{۴} e^{-x} dx$ به دست آورید و با مقدار دقیق آن مقایسه کنید.

```
m <- 10000
x <- runif(m, min=2, max=4)
theta.hat <- mean(exp(-x)) * 2
> print(theta.hat)
[1] 0.1180278
> print(exp(-2) - exp(-4))
[1] 0.1170196
```

انتگرال‌های بی‌کران

از روش مونت کارلو برای محاسبه تابع توزیع نرمال استاندارد استفاده کنید:

$$\Phi(z) = \int_{-\infty}^{z} \frac{۱}{\sqrt{۲\pi}} e^{-\frac{x^۲}{۲}} dx.$$

می‌توان با تغییر متغیر آن‌ها را به انتگرال‌های کراندار تبدیل کرد (تمرین: چگونه؟).

انتگرال‌های بی‌کران

از روش مونت کارلو برای محاسبه تابع توزیع نرمال استاندارد استفاده کنید:

$$\Phi(z) = \int_{-\infty}^{z} \frac{۱}{\sqrt{۲\pi}} e^{-\frac{x^۲}{۲}} dx.$$

می‌توان با تغییر متغیر آن‌ها را به انتگرال‌های کراندار تبدیل کرد (تمرین: چگونه؟).

نکته: با استفاده از تابع نشانگر، احتمال‌ها را نیز می‌توان به شکل امید ریاضی بیان کرد.

انتگرال‌های بی‌کران

از روش مونت کارلو برای محاسبه تابع توزیع نرمال استاندارد استفاده کنید:

$$\Phi(z) = \int_{-\infty}^{z} \frac{۱}{\sqrt{۲\pi}} e^{-\frac{x^۲}{۲}} dx.$$

می‌توان با تغییر متغیر آن‌ها را به انتگرال‌های کراندار تبدیل کرد (تمرین: چگونه؟).

نکته: با استفاده از تابع نشانگر، احتمال‌ها را نیز می‌توان به شکل امید ریاضی بیان کرد.

فرض کنید $I(\cdot)$ تابع نشانگر باشد و $X \sim N(۰, ۱)$. بنابراین برای هر ثابت z، داریم

$$\mathbb{E}[I(X \leq z)] = \mathbb{P}(X \leq z) = \Phi(z).$$

پس با در نظر گرفتن توزیع نرمال استاندارد برای f، داریم:

$$\hat{\Phi}(z) = \frac{۱}{m} \sum_{j=۱}^{m} I(x_i \leq z).$$

```
z <- seq(.1, 2.5, length = 10)
m <- 10000
x <- rnorm(m)
dim(z) <- length(z)
p <- apply(z, MARGIN = 1,
     FUN = function(z, x) {mean(x < z)}, x = x)
Phi <- pnorm(z)
> print(round(rbind(z, p, Phi), 3))
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
z    0.100 0.367 0.633 0.900 1.167 1.433 1.700 1.967 2.233 2.500
p    0.525 0.627 0.718 0.803 0.871 0.919 0.952 0.973 0.988 0.994
Phi 0.540 0.643 0.737 0.816 0.878 0.924 0.955 0.975 0.987 0.994
```

کران‌های خطا برای انتگرال‌گیری مونت کارلویی

برای مثال بالا، واریانس برآوردگر را محاسبه و کران‌های اطمینان ٪۹۵ را برای برآورد $\Phi(۲)$ به دست می‌آوریم.

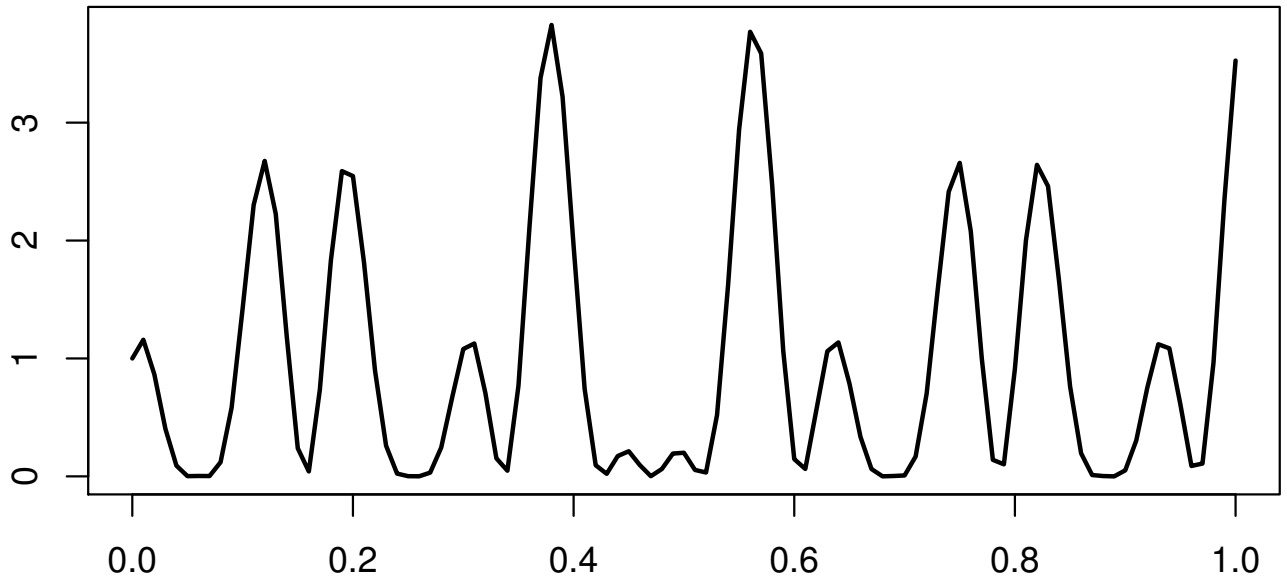
```
z <- 2
m <- 10000
x <- rnorm(m)
g <- (x < z)  #the indicator function
cdf <- mean(g)
v <- mean((g - mean(g))^2) / m
> c(cdf, v)
[1] 9.769000e-01 2.256639e-06
> c(cdf - 1.96 * sqrt(v), cdf + 1.96 * sqrt(v))
[1] 0.9739557 0.9798443
```

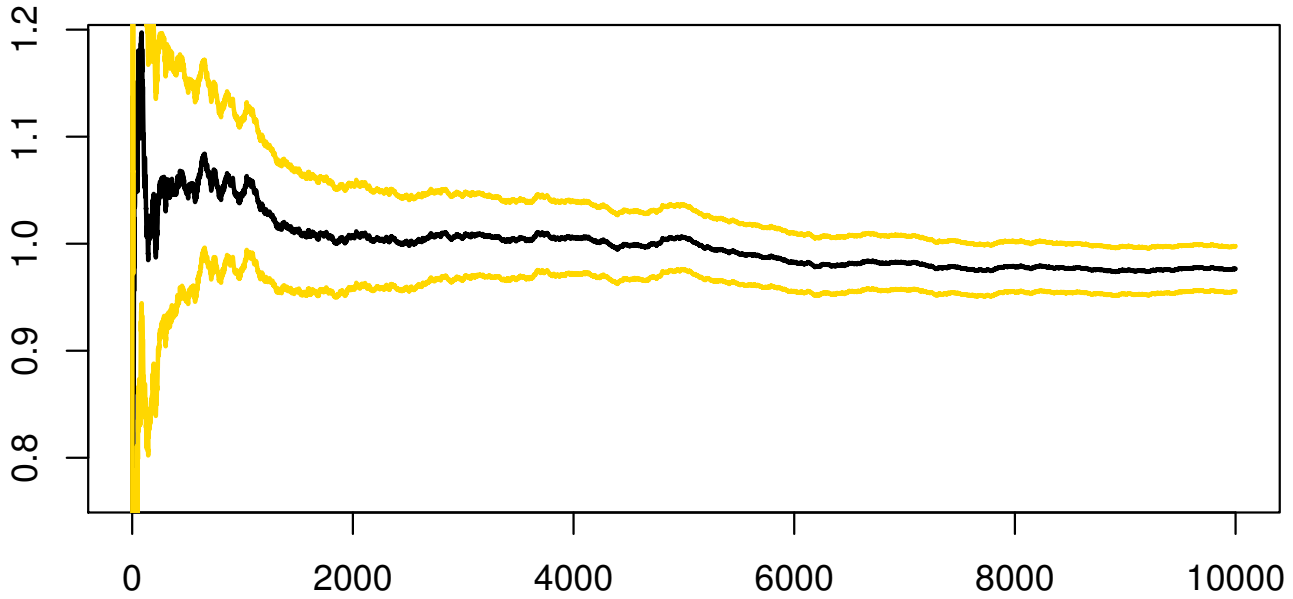
دقت کنید در اینجا تابع $h(x) = I(x \leq z)$ یک متغیر برنولی است و بنابراین واریانس آن برابر $\Phi(۲)(۱ - \Phi(۲))/m$ است که برای این کد برابر مقدار $۲/۲۲۳e - ۰۶$ است.

مثال

برای تابع $h(x) = [\cos ۵۰x + \sin ۲۰x]^۲$، مقدار انتگرال آن را در بازه $[۰, ۱]$ محاسبه کنید.

```
h=function(x){(cos(50*x)+sin(20*x))^2}
par(mar=c(2,2,2,1),mfrow=c(2,1))
curve(h,xlab="Function",ylab="",lwd=2)
> integrate(h,0,1)
0.9652009 with absolute error < 1.9e-10
x=h(runif(10^4))
estint=cumsum(x)/(1:10^4)
esterr=sqrt(cumsum((x-estint)^2))/(1:10^4)
plot(estint, xlab="Mean and error range",type="l",lwd=2,
ylim=mean(x)+20*c(-esterr[10^4],esterr[10^4]),ylab="")
lines(estint+2*esterr,col="gold",lwd=2)
lines(estint-2*esterr,col="gold",lwd=2)
```


0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0


0.8
0.9
1.0
1.1
1.2
0
2000
4000
6000
8000
10000

چند نکته:
باید دقت داشت که برآورد مونت کارلویی مادامی که ν_m برآوردی مناسب برای واریانس $\bar{h}_m$ است، قابل اتکا است.

چند نکته:

باید دقت داشت که برآورد مونت کارلویی مادامی که ν_m برآوردی مناسب برای واریانس $\bar{h}_m$ است، قابل اتکا است.

در موارد بحرانی که ν_m همگرا نیست یا به کندی همگرا می‌شود، نمی‌توان از قضیه حد مرکزی و توزیع تقریبی نرمال بهره برد و در نتیجه محاسبه نواحی اطمینان، قابل اتکا نیستند.

چند نکته:

باید دقت داشت که برآورد مونت کارلویی مادامی که ν_m برآوردی مناسب برای واریانس $\bar{h}_m$ است، قابل اتکا است.

در موارد بحرانی که ν_m همگرا نیست یا به کندی همگرا می‌شود، نمی‌توان از قضیه حد مرکزی و توزیع تقریبی نرمال بهره برد و در نتیجه محاسبه نواحی اطمینان، قابل اتکا نیستند.

در مثال محاسبه تابع توزیع نرمال استاندارد در یک نقطه،

$$\hat{\Phi}(z) = \frac{1}{m}\sum_{j=1}^{m} I(x_j \leq z) \longrightarrow \Phi(z),$$

واریانس دقیق این برآوردگر برابر $\Phi(z)(1-\Phi(z))/m$ است که تقریبا برابر $Var \approx \frac{1}{4m}$ می‌باشد. در این حالت برای داشتن برآوردی با دقت چهار رقم اعشار، $2 \times \sqrt{\frac{1}{4m}} \leq 10^{-4}$، نیاز به تکرار مونت کارلوی $m = (10^4)^2 = 10^8$ داریم.

مثال بیزی: توزیع نرمال–پیشین کوشی

برای برآورد میانگین توزیع نرمال، یک توزیع پیشین تنومند، توزیع کوشی است:

$$X \sim N(\theta, ۱), \quad \theta \sim C(۰, ۱).$$

تحت تابع زیان توان دوم خطا، میانگین پسین عبارتست از:

$$\delta^{\pi} = \frac{\int_{-\infty}^{\infty} \frac{\theta}{۱+\theta^{۲}} e^{-(x-\theta)^{۲}/۲} d\theta}{\int_{-\infty}^{\infty} \frac{۱}{۱+\theta^{۲}} e^{-(x-\theta)^{۲}/۲} d\theta}.$$

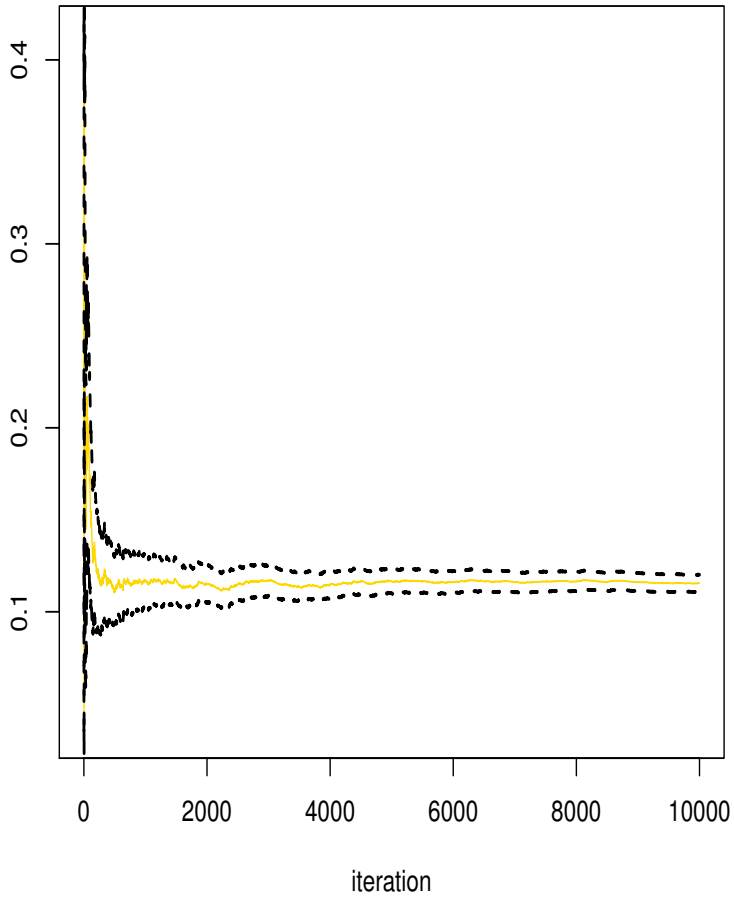
شکل δ^{π}، شبیه‌سازی از توزیع $N(x, ۱)$ را پیشنهاد می‌کند:

$$\theta_{۱}, \ldots, \theta_{m} \sim N(x, ۱),$$

و بنابراین

$$\hat{\delta}^{\pi}_{m} = \sum_{i=۱}^{m} \frac{\theta_i}{۱ + \theta_i^{۲}} \Big/ \sum_{i=۱}^{m} \frac{۱}{۱ + \theta_i^{۲}} \longrightarrow \delta^{\pi}.$$

```
set.seed(21351)
x=2
Niter=10^4
co=rnorm(Niter,mean=x)
x1=co/(1+co^2)
x2=1/(1+co^2)
>  mean(x1)/mean(x2)
[1] 1.2729
th=rcauchy(Niter)
d1=th*dnorm(th,mean=x)
d2=dnorm(th,mean=x)
> mean(d1)/mean(d2)
[1] 1.27617
estint1=cumsum(d1)/(1:Niter)
esterr1=sqrt(cumsum((d1-estint1)^2))/(1:Niter)
plot(estint1,type="l",xlab="iteration",ylab="",
col="gold")
lines(estint1-2*esterr1,lty=2,lwd=2)
lines(estint1+2*esterr1,lty=2,lwd=2)
```



نمونه‌گیری نقاط مهم

این روش را به این دلیل نمونه‌گیری نقاط مهم گویند که تمرکز نمونه‌گیری بر نواحی چگال‌تر (و در نتیجه انتخاب نقاط مهم‌تر) قرار دارد.

نمونه‌گیری نقاط مهم

این روش را به این دلیل نمونه‌گیری نقاط مهم گویند که تمرکز نمونه‌گیری بر نواحی چگال‌تر (و در نتیجه انتخاب نقاط مهم‌تر) قرار دارد.

انتگرال $\mathcal{I}$ را به خاطر آورید.

تضاد:
تولید مستقیم نمونه از تابع چگالی f لزوماً بهینه نیست. (چرا؟)

نمونه‌گیری نقاط مهم

این روش را به این دلیل نمونه‌گیری نقاط مهم گویند که تمرکز نمونه‌گیری بر نواحی چگال‌تر (و در نتیجه انتخاب نقاط مهم‌تر) قرار دارد.

انتگرال $\mathcal{I}$ را به خاطر آورید.

تضاد:
تولید مستقیم نمونه از تابع چگالی f لزوماً بهینه نیست. (چرا؟)

یک روش جانشین، استفاده از نمونه‌گیری از نقاط مهم (*Importance Sampling*) است که بر اساس نمایش جانشینی برای $\mathcal{I}$ عمل می‌کند:

$$\mathbb{E}_f(h(X)) = \int_{\chi} \left[h(x) \frac{f(x)}{g(x)} \right] g(x)\,dx = \mathbb{E}_g \left[h(X) \frac{f(X)}{g(X)} \right].$$

نمونه‌گیری نقاط مهم

این روش را به این دلیل نمونه‌گیری نقاط مهم گویند که تمرکز نمونه‌گیری بر نواحی چگال‌تر (و در نتیجه انتخاب نقاط مهم‌تر) قرار دارد.

انتگرال $\mathcal{I}$ را به خاطر آورید.

تضاد:
تولید مستقیم نمونه از تابع چگالی f لزوماً بهینه نیست. (چرا؟)

یک روش جانشین، استفاده از نمونه‌گیری از نقاط مهم (*Importance Sampling*) است که بر اساس نمایش جانشینی برای $\mathcal{I}$ عمل می‌کند:

$$\mathbb{E}_f(h(X)) = \int_{\chi} \left[h(x)\frac{f(x)}{g(x)}\right] g(x)dx = \mathbb{E}_g\left[h(X)\frac{f(X)}{g(X)}\right].$$

این صورت نمایش به ما اجازه استفاده از توزیع‌های دیگری به جز f را می‌دهد.

نمونه‌گیری نقاط مهم

این روش را به این دلیل نمونه‌گیری نقاط مهم گویند که تمرکز نمونه‌گیری بر نواحی چگال‌تر (و در نتیجه انتخاب نقاط مهم‌تر) قرار دارد.

انتگرال $\mathcal{I}$ را به خاطر آورید.

تضاد:
تولید مستقیم نمونه از تابع چگالی f لزوماً بهینه نیست. (چرا؟)

یک روش جانشین، استفاده از نمونه‌گیری از نقاط مهم (*Importance Sampling*) است که بر اساس نمایش جانشینی برای $\mathcal{I}$ عمل می‌کند:

$$\mathbb{E}_f(h(X)) = \int_{\chi} \left[h(x)\frac{f(x)}{g(x)}\right] g(x)dx = \mathbb{E}_g\left[h(X)\frac{f(X)}{g(X)}\right].$$

این صورت نمایش به ما اجازه استفاده از توزیع‌های دیگری به جز f را می‌دهد.

به f توزیع اصلی و به g توزیع ابزاری (*Instrumental Distribution*) نیز می‌گویند.

الگوریتم نمونه‌گیری نقاط مهم

برای محاسبه انتگرال $\mathcal{I}$ می‌توان به صورت زیر عمل کرد:

۱. نمونه $X_1, \ldots, X_m$ را از توزیع g تولید کن

الگوریتم نمونه‌گیری نقاط مهم

برای محاسبه انتگرال $\mathcal{I}$ می‌توان به صورت زیر عمل کرد:

۱. نمونه $X_1, \ldots, X_m$ را از توزیع g تولید کن

۲. از تقریب زیر استفاده کن:

$$\frac{1}{m} \sum_{j=1}^{m} \frac{f(X_j)}{g(X_j)} h(X_j).$$

استفاده از سایر توزیع‌ها به جای f می‌تواند باعث کاهش واریانس برآوردگرهای حاصل شود.

مشابه قبل

همگرایی:

$$\frac{1}{m}\sum_{j=1}^{m}\frac{f(X_j)}{g(X_j)}h(X_j) \longrightarrow \int_{\chi} h(x)f(x)dx.$$

مشابه قبل

همگرایی:

$$\frac{1}{m}\sum_{j=1}^{m}\frac{f(X_j)}{g(X_j)}h(X_j) \longrightarrow \int_{\chi} h(x)f(x)dx.$$

این همگرایی به ازای هر توزیع g برقرار است، مشروط بر آن که $supp(f) \subset supp(g)$.

مشابه قبل

همگرایی:

$$\frac{1}{m}\sum_{j=1}^{m}\frac{f(X_j)}{g(X_j)}h(X_j) \longrightarrow \int_{\chi} h(x)f(x)dx.$$

این همگرایی به ازای هر توزیع g برقرار است، مشروط بر آن که $supp(f) \subset supp(g)$.

دو نکته مهم:

- توزیع g باید به گونه‌ای انتخاب شود که تولید نمونه از آن ساده باشد

مشابه قبل

همگرایی:

$$\frac{1}{m}\sum_{j=1}^{m}\frac{f(X_j)}{g(X_j)}h(X_j) \longrightarrow \int_{\chi} h(x)f(x)dx.$$

این همگرایی به ازای هر توزیع g برقرار است، مشروط بر آن که $supp(f) \subset supp(g)$.

دو نکته مهم:

- توزیع g باید به گونه‌ای انتخاب شود که تولید نمونه از آن ساده باشد
- از نمونه تولیدشده (توسط g) می‌توان به دفعات استفاده کرد. نه تنها برای توابع h مختلف بلکه برای توزیع‌های f متفاوت هم نمونه مشابه قابل استفاده است

انتخاب توزیع ابزاری

اگرچه g هر تابع چگالی می‌تواند باشد، بعضی از انتخاب‌ها بهتر از بقیه هستند.

- برآوردگر فقط وقتی دارای واریانس متناهی است که

$$\mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right] = \int_{\chi} h^2(x)\frac{f(x)}{g(x)}\,dx < \infty.$$

انتخاب توزیع ابزاری

اگرچه g هر تابع چگالی می‌تواند باشد، بعضی از انتخاب‌ها بهتر از بقیه هستند.

- برآوردگر فقط وقتی دارای واریانس متناهی است که

$$\mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right] = \int_\chi h^2(x)\frac{f(x)}{g(x)}\,dx < \infty.$$

- توزیع‌های ابزاری با دم‌های سبک‌تر از دم‌های f (یعنی زمانی که $\sup \frac{f}{g} = \infty$) مناسب نیستند.

انتخاب توزیع ابزاری

اگرچه g هر تابع چگالی می‌تواند باشد، بعضی از انتخاب‌ها بهتر از بقیه هستند.

- برآوردگر فقط وقتی دارای واریانس متناهی است که

$$\mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right] = \int_\chi h^2(x)\frac{f(x)}{g(x)}\,dx < \infty.$$

- توزیع‌های ابزاری با دم‌های سبک‌تر از دم‌های f (یعنی زمانی که $\sup \frac{f}{g} = \infty$) مناسب نیستند.
- اگر $\sup \frac{f}{g} = \infty$، وزن‌های $\frac{f(x_j)}{g(x_j)}$ به شدت متغیر خواهند بود و این به آن معنی است که به تعداد اندکی از x_jها اهمیت بیش از اندازه داده می‌شود

انتخاب توزیع ابزاری

اگرچه g هر تابع چگالی می‌تواند باشد، بعضی از انتخاب‌ها بهتر از بقیه هستند.

- برآوردگر فقط وقتی دارای واریانس متناهی است که

$$\mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right] = \int_\chi h^2(x)\frac{f(x)}{g(x)}dx < \infty.$$

- توزیع‌های ابزاری با دم‌های سبک‌تر از دم‌های f (یعنی زمانی که $\sup \frac{f}{g} = \infty$) مناسب نیستند.
- اگر $\sup \frac{f}{g} = \infty$، وزن‌های $\frac{f(x_j)}{g(x_j)}$ به شدت متغیر خواهند بود و این به آن معنی است که به تعداد اندکی از x_jها اهمیت بیش از اندازه داده می‌شود
- اگر $\sup \frac{f}{g} = M < \infty$، برای تولید نمونه از f به طور مستقیم، از روش پذیرش ـ رد هم می‌توان استفاده کرد

مثال: توزیع کوشی

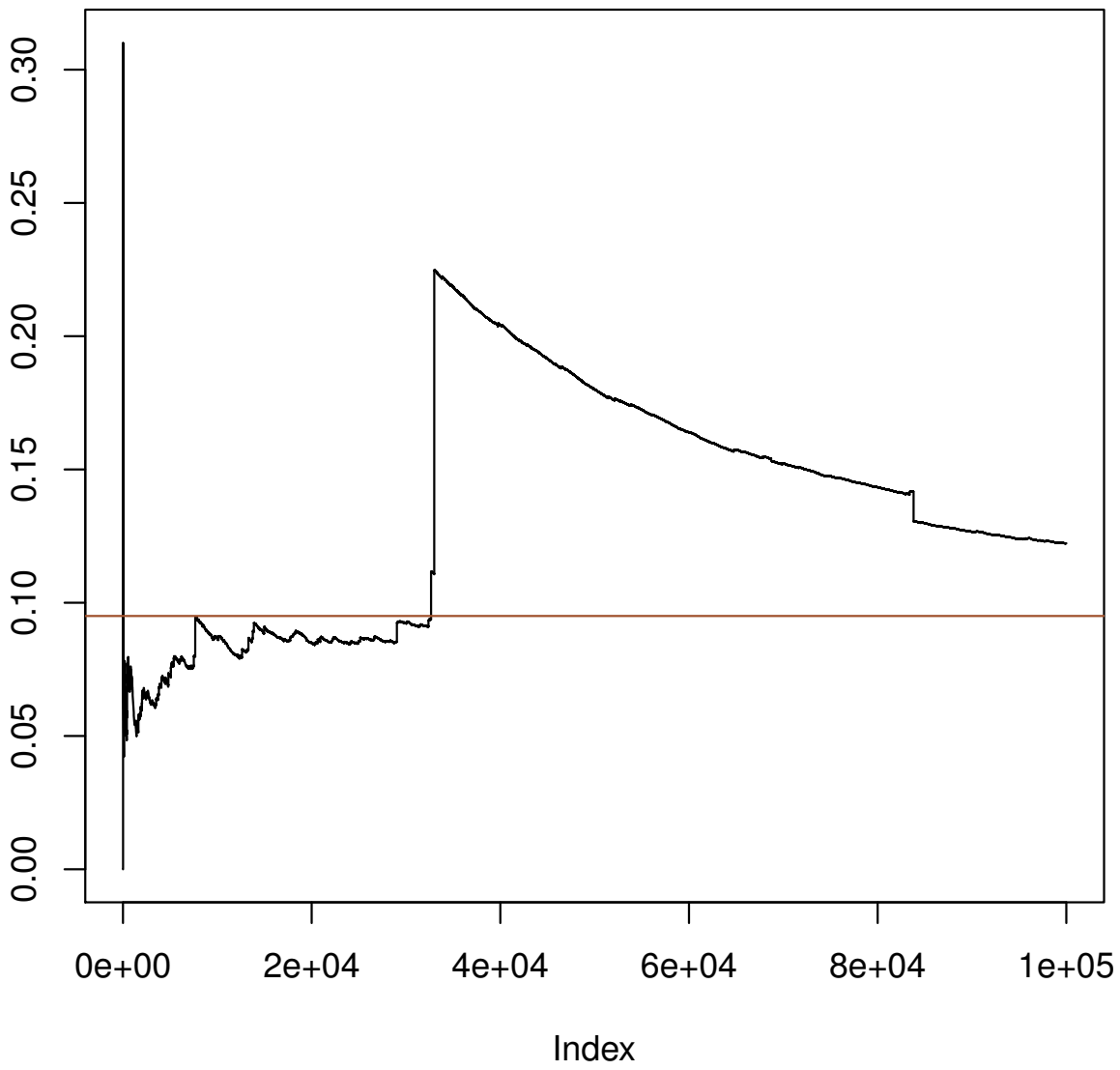
اگر توزیع کوشی استاندارد را به عنوان تابع اصلی و نرمال استاندارد را به عنوان تابع ابزاری در نظر بگیریم، داریم

$$\rho(x) = \frac{f(x)}{g(x)} = \sqrt{۲\pi}\frac{\exp x^۲/۲}{\pi(۱+x^۲)}.$$

این نسبت به شدت بدرفتار است، یعنی

$$\int_{-\infty}^{\infty} \rho^۲(x)g(x)dx = \infty \quad \text{(تمرین)}$$

و در نتیجه عملکرد ضعیفی برای برآوردگر حاصل از نمونه‌گیری نقاط مهم می‌توان متصور بود.


0.00
0.05
0.10
0.15
0.20
0.25
0.30
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
Index

تابع ابزاری بهینه

تابع g که واریانس برآوردگر نمونه‌گیری نقاط مهم را می‌نیمم می‌کند، عبارتست از

$$g^*(x) = \frac{|h(x)|f(x)}{\int |h(z)|f(z)dz}.$$

دستیابی به این انتخاب بهینه میسر نیست.

تابع ابزاری بهینه

تابع g که واریانس برآوردگر نمونه‌گیری نقاط مهم را می‌نیمم می‌کند، عبارتست از

$$g^*(x) = \frac{|h(x)|f(x)}{\int |h(z)|f(z)dz}.$$

دستیابی به این انتخاب بهینه میسر نیست.

زیرا اطلاع از آن نیازمند دانستن مقدار انتگرالی است که به دنبال محاسبه آن هستیم.

نسخه عملی برآوردگر نمونه‌گیری نقاط مهم

یک روش جانشین برای برآوردگر معرفی‌شده به عنوان برآوردگر نمونه‌گیری نقاط مهم (در چند اسلاید قبل)، استفاده از

$$\frac{\sum_{j=1}^{m} h(x_j)\frac{f(x_j)}{g(x_j)}}{\sum_{j=1}^{m} \frac{f(x_j)}{g(x_j)}},$$

است که هم مشکل متناهی بودن واریانس برآوردگر را مرتفع می‌کند و هم به طور کلی برآوردگر پایدارتری را نتیجه می‌دهد.

در این نسخه، به جای m از مجموع وزن‌ها، $w_j = \frac{f(x_j)}{g(x_j)}$، استفاده می‌شود.

دقت کنید که چون $\frac{1}{m}\sum_{j=1}^{m} \frac{f(x_j)}{g(x_j)} \longrightarrow 1$، زمانی که $m \longrightarrow \infty$، این برآوردگر هم بنابر قانون قوی اعداد بزرگ به $\mathbb{E}_f(h(X))$ میل می‌کند.

با این که این برآوردگر اریب است، اما اریبی کوچک آن و واریانس کمتر، آن را به برآوردگر ارجح نسبت به نسخه قبلی تبدیل کرده است.

میانگین هارمونیک

با در نظر گرفتن تابع بهینه، برای این نسخه از برآوردگر نمونه‌گیری نقاط مهم داریم:

$$\frac{\sum_{j=1}^{m} h(x_j)\frac{f(x_j)}{g(x_j)}}{\sum_{j=1}^{m} \frac{f(x_j)}{g(x_j)}} = \frac{\sum_{j=1}^{m} h(x_j)|h(x_j)|^{-1}}{\sum_{j=1}^{m} |h(x_j)|^{-1}},$$

به طوری که $x_j \sim g \propto |h|f$.

دقت کنید که صورت عبارت بالا عبارتست از تعداد دفعاتی که $h(x_j)$ مثبت است منهای تعداد دفعاتی که منفی است.

اگر h مثبت باشد، این برآوردگر به میانگین هارمونیک تبدیل می‌شود.

علی رغم آنچه که تصور می‌شود بهینگی مطرح‌شده، برای این برآوردگر برقرار نیست و ثابت شده است که می‌تواند به برآوردگری اریب و به شدت ناپایدار منجر شود.

در متون مختلفی در مورد غیرقابل اعتماد بودن برآوردگر میانگین هارمونیک بحث شده است.

رهنمود کلی

پروژه: نمایش عملکرد برآوردگر میانگین هارمونیک

از نقطه نظر کاربردی، توزیعی برای g بهینه است که $|h|f/g$ تقریبا ثابت باشد و واریانس متناهی داشته باشد

مثال

$$\int_{۰}^{۱} \frac{e^{-x}}{۱+x^{۲}}\,dx = ?,$$

در واقع در این مثال،

$$g(x) = \begin{cases} \frac{e^{-x}}{۱+x^{۲}} & ۰ < x < ۱ \\ ۰ & o.w. \end{cases}$$

برای حل آن از توابع ابزاری زیر استفاده می‌کنیم:

$$\begin{aligned}
f_0(x) &= 1, \quad 0 < x < 1, \\
f_1(x) &= e^{-x}, \quad 0 < x < \infty, \\
f_2(x) &= \frac{1}{\pi(1+x^2)}, \quad -\infty < x < \infty, \\
f_3(x) &= \frac{e^{-x}}{1-e^{-1}}, \quad 0 < x < 1, \\
f_4(x) &= \frac{4}{\pi(1+x^2)}, \quad 0 < x < 1.
\end{aligned}$$

دو تابع f_1 و f_2 دارای تکیه‌گاه بزرگتری هستند و بسیاری از مقادیر تولید شده از این توزیع‌ها، در مجموع سهمی برابر صفر خواهند داشت. و این یعنی ناکارایی.

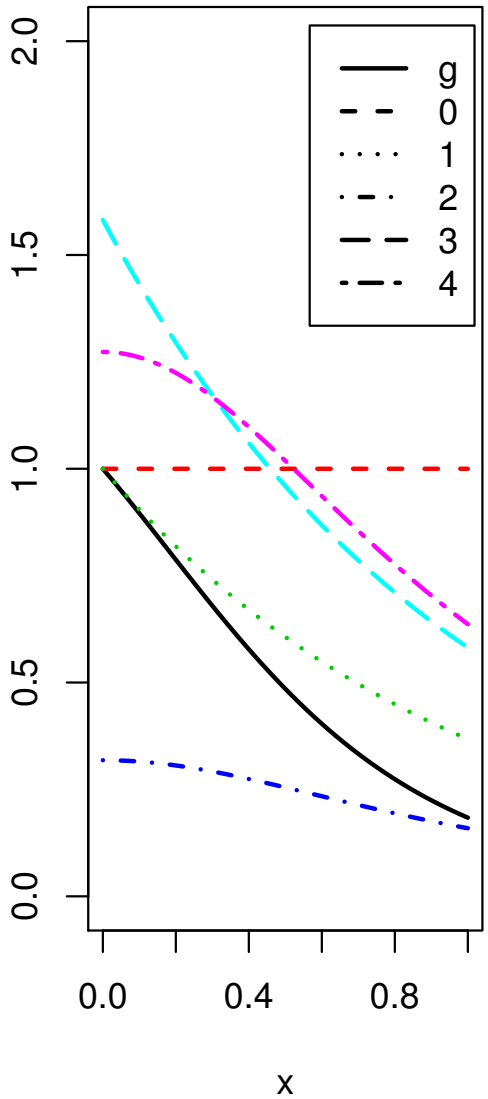
از تمام این توزیع‌ها به راحتی می‌توان نمونه تولید کرد.

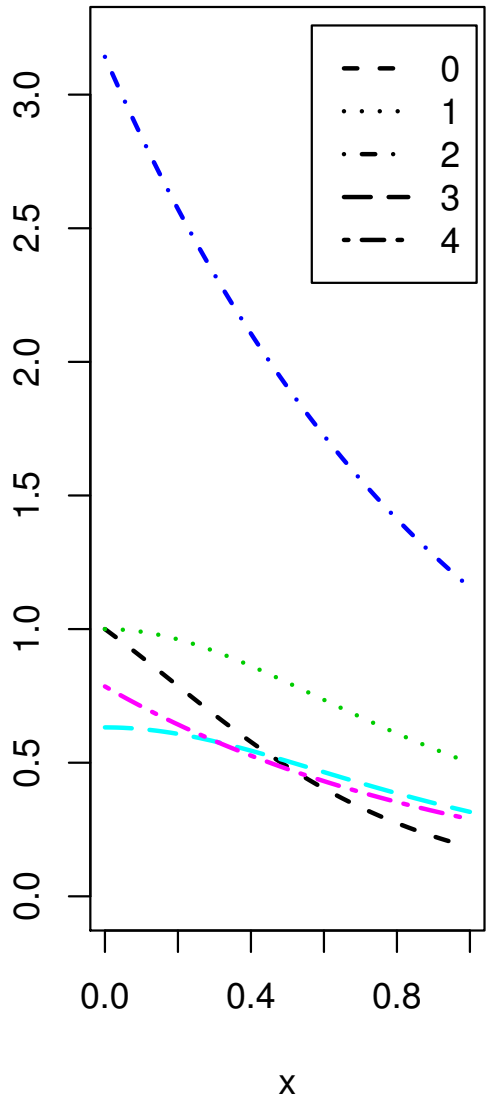
f_2 توزیع کوشی استاندارد است

تابعی که برای آن نسبت g/f، دقت کنید در این مثال $h = 1$ است، از بقیه به مقداری ثابت نزدیکتر است، تابع f_3 است. از روی شکل هم دیده می‌شود.

```
x <- seq(0, 1, .01)
w <- 2
f1 <- exp(-x)
f2 <- (1 / pi) / (1 + x^2)
f3 <- exp(-x) / (1 - exp(-1))
f4 <- 4 / ((1 + x^2) * pi)
g <- exp(-x) / (1 + x^2)
```

```
par(mfrow=c(1,2))
## figure (a)
plot(x, g, type = "l", main = "", ylab = "",
     ylim = c(0,2), lwd = w)
lines(x, g/g, lty = 2, col = 2, lwd = w)
lines(x, f1, lty = 3, col = 3, lwd = w)
lines(x, f2, lty = 4, col = 4, lwd = w)
lines(x, f3, lty = 5, col = 5, lwd = w)
lines(x, f4, lty = 6, col = 6, lwd = w)
legend("topright", legend = c("g", 0:4),
      lty = 1:6, lwd = w, inset = 0.02)
# figure (b)
plot(x, g, type = "l", main = "", ylab = "",
     ylim = c(0,3.2), lwd = w, lty = 2)
lines(x, g/f1, lty = 3, col = 3, lwd = w)
lines(x, g/f2, lty = 4, col = 4, lwd = w)
lines(x, g/f3, lty = 5, col = 5, lwd = w)
lines(x, g/f4, lty = 6, col = 6, lwd = w)
legend("topright", legend = c(0:4),
       lty = 2:6, lwd = w, inset = 0.02)
```


g
0
1
2
3
4
0.0
0.5
1.0
1.5
2.0
0.0
0.4
0.8
x


0
1
2
3
4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.4
0.8
x

```
m <- 10000
theta.hat <- se <- numeric(5)
g <- function(x) {
     exp(-x - log(1+x^2)) * (x > 0) * (x < 1)
}
x <- runif(m)     #using f0
fg <- g(x)
theta.hat[1] <- mean(fg)
se[1] <- sd(fg)
x <- rexp(m, 1)   #using f1
fg <- g(x) / exp(-x)
theta.hat[2] <- mean(fg)
se[2] <- sd(fg)
x <- rcauchy(m)   #using f2
i <- c(which(x > 1), which(x < 0))
x[i] <- 2  #to catch overflow errors in g(x)
fg <- g(x) / dcauchy(x)
theta.hat[3] <- mean(fg)
se[3] <- sd(fg)
u <- runif(m)     #f3, inverse transform method
x <- - log(1 - u * (1 - exp(-1)))
fg <- g(x) / (exp(-x) / (1 - exp(-1)))
theta.hat[4] <- mean(fg)
se[4] <- sd(fg)
u <- runif(m)    #f4, inverse transform method
x <- tan(pi * u / 4)
fg <- g(x) / (4 / ((1 + x^2) * pi))
theta.hat[5] <- mean(fg)
se[5] <- sd(fg)
### Results
> rbind(theta.hat, se)
              [,1]      [,2]      [,3]       [,4]      [,5]
theta.hat 0.5268425 0.5206398 0.5236278 0.52512375 0.5257661
se        0.2467741 0.4191347 0.9496426 0.09542621 0.1416929
```

مثال

استفاده از روش معمول مونت کارلو برای تقریب احتمال‌های دمی مانند $P(X > a)$ وقتی که a بزرگ باشد، می‌تواند بسیار پرهزینه و نادقیق باشد.

به عنوان مثال اگر فرض کنید $Z \sim N(۰, ۱)$، و مایل باشیم $P(Z > ۴/۵)$ را محاسبه کنیم، که مقدار آن خیلی کوچک است،

```
> pnorm(-4.5)
[1] 3.397673e-06
```

تولید نمونه‌ای از $Z^{(i)} \sim N(۰, ۱)$ به طوری که بزرگتر از ۴/۵ قرار گیرد، هر سه میلیون بار یک بار اتفاق می‌افتد!!!

از آنجا که ما علاقه‌مند به محاسبه احتمال یک پیشامد خیلی نادر هستیم، استفاده از روش معمول برای تولید نمونه از f و رسیدن به یک جواب مناسب (با دقت مناسب)، نیازمند تولید بسیار زیادی نمونه است.

اما نمونه‌گیری نقاط مهم به طور چشمگیری دقت و کارایی محاسبه چنین احتمالی را افزایش می‌دهد.

اگر توزیعی با تکیه‌گاه محدود به $(۴/۵, \infty)$ در نظر بگیریم، قسمتی از واریانس اضافی و غیرضروری برآوردگر مونت کارلو که ناشی از تولید نمونه‌های با وزن صفر است (وقتی که $x < ۴/۵$)، حذف خواهد شد.
یک انتخاب مناسب برای g یعنی تابع ابزاری، نمایی بریده شده در $۴/۵$ است:

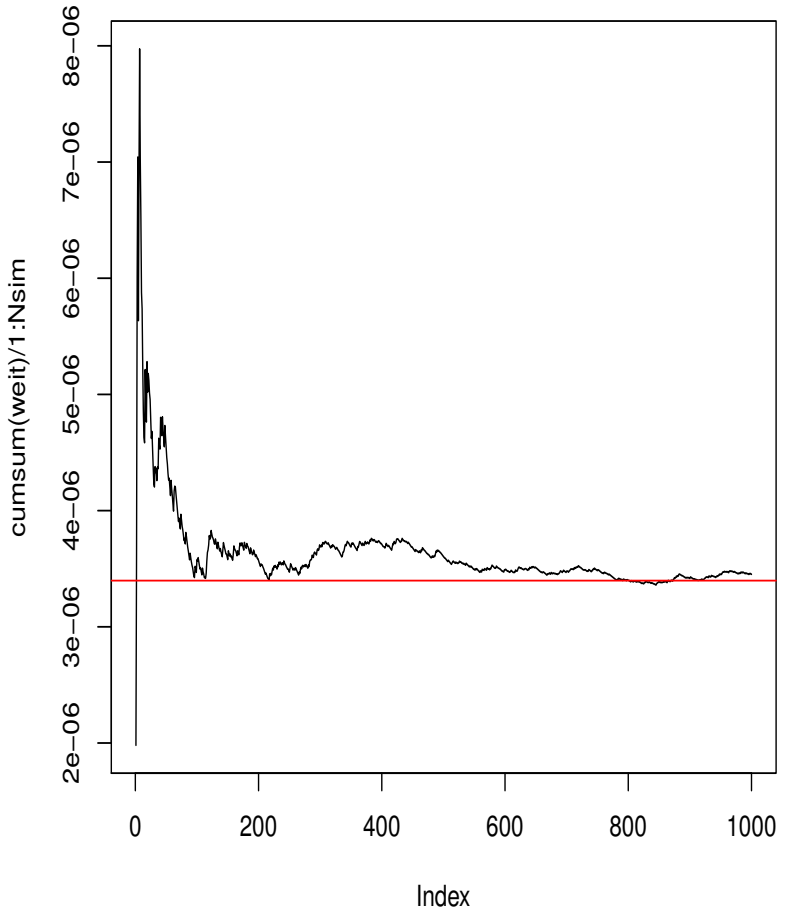
$$g(y) = \frac{e^{-y}}{\int_{۴/۵}^{\infty} e^{-x}\,dx} = e^{-(y-۴/۵)}.$$

بنابراین برآوردگر IS عبارتست از:

$$\frac{۱}{m}\sum_{j=۱}^{m}\frac{f(y_j)}{g(y_j)} = \frac{۱}{m}\sum_{j=۱}^{m}\frac{e^{-y_j^۲/۲+y_j-۴/۵}}{\sqrt{۲\pi}},$$

که در آن Y_j ها از g تولید شده‌اند.

```
Nsim=1000
y=rexp(Nsim)+4.5
weit=dnorm(y)/dexp(y-4.5)
plot(cumsum(weit)/1:Nsim,type="l")
abline(a=pnorm(-4.5),b=0,col="red")
## Estimated Value = 3.452e-06 vs.
## True value = 3.398e-06
```



محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه هشتم: روش‌های مونت کارلو (ارزیابی، کنترل و سرعت دادن به همگرایی)

حسین باغیشنی

دانشگاه شاهرود

۲۳ آبان ۱۳۹۳

کاهش واریانس

برای دو برآوردگر $\hat{\theta}_1$ و $\hat{\theta}_2$، اگر $Var(\hat{\theta}_1) < Var(\hat{\theta}_2)$ آنگاه درصد کاهش واریانس توسط $\hat{\theta}_1$ عبارتست از:

$$100 \left(\frac{Var(\hat{\theta}_2) - Var(\hat{\theta}_1)}{Var(\hat{\theta}_1)} \right).$$

فرض کنید

$$X^{(j)} = (X_1^{(j)}, \ldots, X_n^{(j)}), \ j = 1, \ldots, m,$$

یک نمونه تصادفی ساده از توزیع X باشد. بنابراین

$$Y^{(j)} = h(X_1^{(j)}, \ldots, X_n^{(j)}), \ j = 1, \ldots, m,$$

یک نمونه تصادفی ساده از توزیع $Y = h(X_1, \ldots, X_n)$ است و فرض کنید

$$\mathbb{E}(\bar{Y}) = \mathbb{E}\left(\frac{1}{m} \sum_{j=1}^{m} Y_j \right) = \theta.$$

بنابراین برآوردگر مونت کارلو، یعنی $\hat{\theta} = \bar{Y}$، ناریب است و واریانس آن برابر

$$Var(\hat{\theta}) = \frac{Var_f(h(X))}{m}$$

است.

از فرمول واریانس واضح است که هر چقدر حجم نمونه مونت کارلو، m، افزایش یابد واریانس برآوردگر کاهش می‌یابد.

اما برای کاهش میزان کوچکی در انحراف معیار، باید حجم نمونه بزرگی تولید شود. مثلا برای کاهش از ۰/۰۱ به ۰/۰۰۰۱ باید تقریباً ۱۰۰۰۰ تکرار بیشتر تولید شود.

بنابراین اگرچه با افزایش حجم نمونه مونت کارلویی واریانس برآوردگر کاهش می‌یابد، اما هزینه محاسبات نیز زیاد خواهد بود. پس باید بین میزان کاهش واریانس و کارایی روش (سرعت به دست آوردن برآوردگر) تعادلی ایجاد کرد.

روش‌های مختلفی برای کاهش واریانس معرفی شده‌اند در حالی که هزینه محاسباتی زیادی هم در بر ندارند که در این جلسه به معرفی برخی از آن‌ها می‌پردازیم.

متغیرهای ناهم‌سو

$$Var(\frac{U_۱ + U_۲}{۲}) = \frac{۱}{۴}\{Var(U_۱) + Var(U_۲)\} + \frac{۱}{۲}Cov(U_۱, U_۲).$$

واریانس میانگین کمتر از واریانس $U_۱$ و $U_۲$ خواهد بود هرگاه این دو متغیر دارای همبستگی منفی باشند. یا به عبارت دیگر ناهم‌سو باشند.

همین واقعیت ساده، روشی جهت کاهش واریانس معرفی می‌کند.

فرض کنید $(X_۱, \ldots, X_m)$ و $(Y_۱, \ldots, Y_m)$ دو نمونه از f برای برآورد

$$\mathcal{J} = \int_{\Re} h(t)f(t)dt,$$

به وسیله

$$\hat{\mathcal{J}}_۱ = \frac{۱}{m}\sum_{j=۱}^{m} h(X_j) \;\&\; \hat{\mathcal{J}}_۲ = \frac{۱}{m}\sum_{j=۱}^{m} h(Y_j),$$

باشند، به طوری که میانگین آن‌ها $\mathcal{J}$ و واریانسشان $\sigma^۲$ باشد.

کاهش واریانس

$$Var(\frac{\hat{\mathcal{J}}_۱ + \hat{\mathcal{J}}_۲}{۲}) = \frac{\sigma^۲}{۲} + \frac{۱}{۲} Cov(\hat{\mathcal{J}}_۱, \hat{\mathcal{J}}_۲).$$

اگر دو نمونه همبسته منفی باشند، آنگاه

$$Cov(\hat{\mathcal{J}}_۱, \hat{\mathcal{J}}_۲) \leq ۰$$

و در نتیجه دو نمونه مستقل و هم‌حجم برآوردگر را بهبود می‌دهند.

فرض کنید $X_1, \ldots, X_n$ با روش تبدیل انتگرال احتمال تولید شده باشند. یعنی به ازای هر تکرار m،

1. U_j از توزیع $U(0,1)$ تولید شده است
2. $X^{(j)} = F_X^{-1}(U_j),\ j = 1, \ldots, n$

می‌دانیم U و $1-U$ هم‌توزیع هستند و دارای همبستگی منفی می‌باشند. بنابرین

$$Y_j = h(F_X^{-1}(U_1^{(j)}), \ldots, F_X^{-1}(U_n^{(j)})),$$

و

$$Y_j' = h(F_X^{-1}(1 - U_1^{(j)}), \ldots, F_X^{-1}(1 - U_n^{(j)})),$$

هم‌توزیع هستند.

اگر تابع h یکنوا باشد، متغیرهای Y_j و Y_j' همبسته منفی هستند

اثبات آن در صفحه ۱۲۹ کتاب قرار دارد

دستور اجرای روش چگونه است؟

اگر m نمونه مونت کارلو لازم باشد، $m/۲$ آن‌ها را برابر

$$Y_j = h(F_X^{-۱}(U_۱^{(j)}), \ldots, F_X^{-۱}(U_n^{(j)})),$$

و بقیه $m/۲$ را برابر

$$Y_j' = h(F_X^{-۱}(۱ - U_۱^{(j)}), \ldots, F_X^{-۱}(۱ - U_n^{(j)})),$$

قرار می‌دهیم، به طوری که $U_i^{(j)},\ i = ۱, \ldots, n; j = ۱, \ldots, m/۲$ نمونه‌های تصادفی از $U(۰, ۱)$ هستند.

در نتیجه برآوردگر حاصل از روش متغیرهای ناهم‌سو عبارت است از:

$$\hat{\theta} = \frac{۱}{m/۲} \sum_{j=۱}^{m/۲} \left(\frac{Y_j + Y_j'}{۲} \right).$$

با این روش به جای تولید nm نمونه، نمونه‌ای به حجم $nm/۲$ تولید می‌شود و برآوردگری با واریانس کمتر خواهیم داشت.

مثال

$$\Phi(x) = \int_{-\infty}^{x} \frac{۱}{\sqrt{۲\pi}} e^{-t^۲/۲}\, dt.$$

با تبدیل متغیر، محاسبه این انتگرال تبدیل می‌شود به محاسبه $\theta = \mathbb{E}_U \left[x e^{-(xU)^۲/۲} \right]$ که در آن $U \sim U(۰, ۱)$.

با در نظر گرفتن مقادیر مثبت x، تابع h یکنوا خواهد بود و در نتیجه با تولید $u_۱, \ldots, u_{m/۲} \sim U(۰, ۱)$ داریم:

$$Y_j = h^{(j)}(u) = x e^{-(xu_j)^۲/۲},\ j = ۱, \ldots, m/۲.$$

بقیه $m/۲$ نمونه را به صورت زیر تولید می‌کنیم:

$$Y_j' = x e^{-(x(۱-u_j))^۲/۲},\ j = ۱, \ldots, m/۲.$$

بنابراین

$$\hat{\theta} = \frac{۱}{m/۲} \sum_{j=۱}^{m/۲} \frac{xe^{-(xu_j)^۲/۲} + xe^{-(x(۱-u_j))^۲/۲}}{۲} \longrightarrow \theta$$

```
MC.Phi <- function(x, R = 10000, antithetic = TRUE) {
        u <- runif(R/2)
        if (!antithetic) v <- runif(R/2) else
            v <- 1 - u
        u <- c(u, v)
        cdf <- numeric(length(x))
        for (i in 1:length(x)) {
            g <- x[i] * exp(-(u * x[i])^2 / 2)
            cdf[i] <- mean(g) / sqrt(2 * pi) + 0.5
        }
        cdf
}
```

```
x <- seq(.1, 2.5, length=5)
Phi <- pnorm(x)
set.seed(123)
MC1 <- MC.Phi(x, anti = FALSE)
set.seed(123)
MC2 <- MC.Phi(x)
> print(round(rbind(x, MC1, MC2, Phi), 5))
        [,1]    [,2]    [,3]    [,4]    [,5]
x    0.10000 0.70000 1.30000 1.90000 2.50000
MC1  0.53983 0.75825 0.90418 0.97311 0.99594
MC2  0.53983 0.75805 0.90325 0.97132 0.99370
Phi  0.53983 0.75804 0.90320 0.97128 0.99379
```

کاهش تقریبی واریانس را می‌توان برای یک مقدار معلوم x توسط روش متغیرهای ناهم‌سو در مقابل روش مونت کارلوی استاندارد، محاسبه کرد:

```
m <- 1000
MC1 <- MC2 <- numeric(m)
x <- 1.95
for (i in 1:m) {
     MC1[i] <- MC.Phi(x, R = 1000, anti = FALSE)
     MC2[i] <- MC.Phi(x, R = 1000)
}
> print(sd(MC1))
[1] 0.006874616
> print(sd(MC2))
[1] 0.0004392972
> print((var(MC1) - var(MC2))/var(MC1))
[1] 0.9959166
```

مشاهده می‌کنید که با روش متغیرهای ناهم‌سو، برآوردگر جدید ٪۹۹/۵ کاهش واریانس در نقطه $x = ۱/۹۵$ ایجاد کرده است.

چند نکته

- این روش همیشه قابل اجرا نیست ولی اگر امکان اجرای آن باشد، روشی کارا محسوب می‌شود
- اگر f متقارن باشد، قرار دهید $Y_j = ۲\mu - X_j$ که در آن μ میانگین f می‌باشد.
- اگر $(A_i)_i$ تکیه‌گاه متغیر X را افراز کند، با نمونه‌گیری لایه‌ای، یعنی نمونه‌گیری X_jها در هر A_i، می‌توان واریانس را کاهش داد

متغیرهای کنترلی

رهیافتی دیگر برای کاهش واریانس برآوردگرهای مونت کارلویی $\mathbb{E}(h(X))$، استفاده از متغیرهای کنترلی است.

فرض کنید

$$\mathcal{I} = \int h(x)f(x)dx,$$

نامعلوم و

$$\mathcal{I}_0 = \int h_0(x)f(x)dx,$$

معلوم باشد. $\mathcal{I}$ با $\hat{\mathcal{I}}$ برآورد می‌شود و $\mathcal{I}_0$ با $\hat{\mathcal{I}}_0$.

متغیرهای کنترلی: ادامه

با ترکیب دو برآوردگر داریم:

$$\hat{\mathcal{I}}^* = \hat{\mathcal{I}} + \beta(\hat{\mathcal{I}}_\cdot - \mathcal{I}_\cdot)$$

در این حالت $\hat{\mathcal{I}}^*$ برای $\mathcal{I}$ ناریب است و

$$Var(\hat{\mathcal{I}}^*) = Var(\hat{\mathcal{I}}) + \beta^2\, Var(\hat{\mathcal{I}}_\cdot) + 2\beta\, Cov(\hat{\mathcal{I}}, \hat{\mathcal{I}}_\cdot).$$

کنترل بهینه

انتخاب بهینه برای β عبارتست از:

$$\beta^* = -\frac{Cov(\hat{\mathcal{J}}, \hat{\mathcal{J}}_\cdot)}{Var(\hat{\mathcal{J}}_\cdot)}$$

همچنین

$$Var(\hat{\mathcal{J}}^*) = (1 - \rho^2)\, Var(\hat{\mathcal{J}}),$$

که در آن ρ ضریب همبستگی بین $\hat{\mathcal{J}}$ و $\hat{\mathcal{J}}_\cdot$ است. بنابراین میزان کاهش واریانس برابر است با

$$100\rho^2$$

راه حل معمول برای به دست آوردن β^*، استفاده از رگرسیون است. ضریب رگرسیون $h(x_i)$ روی $h_\cdot(x_i)$ همان β^* بهینه را به دست خواهد داد.

هر چه میزان همبستگی بین $\hat{\mathcal{J}}$ و $\hat{\mathcal{J}}_\cdot$ بیشتر باشد، میزان کاهش واریانس بیشتر خواهد بود.

مثال

$$\theta = \int_{۰}^{۱} \frac{e^{-x}}{۱ + x^{۲}} dx,$$

به طوری که $X \sim U(۰, ۱)$.
اگر قرار دهیم $h_{۰}(x) = \frac{e^{-۰/۵}}{۱+x^{۲}}$، آنگاه در فاصله $(۰, ۱)$ به $h(x)$ نزدیک است، یعنی همبستگی بالایی دارند، و

$$\mathbb{E}(h_{۰}(x)) = e^{-۰/۵} \int_{۰}^{۱} \frac{۱}{۱ + t^{۲}} dt = e^{-۰/۵} \arctan(۱) = e^{-۰/۵} \frac{\pi}{۴}.$$

```
f <- function(u)
     exp(-.5)/(1+u^2)
g <- function(u)
     exp(-u)/(1+u^2)
set.seed(510) #needed later
u <- runif(10000)
B <- f(u)
A <- g(u)
## beta^*
cor(A, B)
a = -cov(A,B) / var(B)    #est of beta*
> a
[1] -2.436228
## Control variate estimate
m <- 100000
u <- runif(m)
T1 <- g(u)
T2 <- T1 + a * (f(u) - exp(-.5)*pi/4)
> c(mean(T1), mean(T2))
[1] 0.5253543 0.5250021
> c(var(T1), var(T2))
[1] 0.060231423 0.003124814
# Percent of reduced variance
> (var(T1) - var(T2)) / var(T1)
[1] 0.9481199
```

محاسبات با استفاده از رگرسیون

```
set.seed(510)
u <- runif(10000)
f <- exp(-.5)/(1+u^2)
g <- exp(-u)/(1+u^2)
c.star <-  - lm(g ~ f)$coeff[2]   # beta[1]
mu <- exp(-.5)*pi/4
> c.star
        f
-2.436228
u <- runif(10000)
f <- exp(-.5)/(1+u^2)
g <- exp(-u)/(1+u^2)
L <- lm(g ~ f)
theta.hat <- sum(L$coeff * c(1, mu))  #pred. value at mu
> theta.hat
[1] 0.5253113
> summary(L)$sigma^2
[1] 0.003117644
> summary(L)$r.squared
[1] 0.9484514
```

چند نکته

- می‌توان به جای یک متغیر از چند متغیر کنترلی استفاده کرد. در این صورت از رگرسیون چندگانه برای برآورد ضرایب بهینه و میزان کاهش واریانس می‌توان بهره برد. همچنین برآورد جدید مقدار پیشگوی حاصل از تابع رگرسیون در نقاط میانگین متغیرهای کنترلی است.
- میزان کاهش واریانس، همان ضریب تعیین رگرسیون است.
- واریانس برآوردگر جدید با روش متغیرهای کنترلی، در مدل رگرسیونی همان MSE/n است
- بین روش‌های متغیرهای ناهم‌سو و کنترلی رابطه مستقیم وجود دارد

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه نهم: شبیه‌سازی و استنباط آماری

حسین باغیشنی

دانشگاه شاهرود

۲۳ آبان ۱۳۹۳

چرا شبیه‌سازی؟

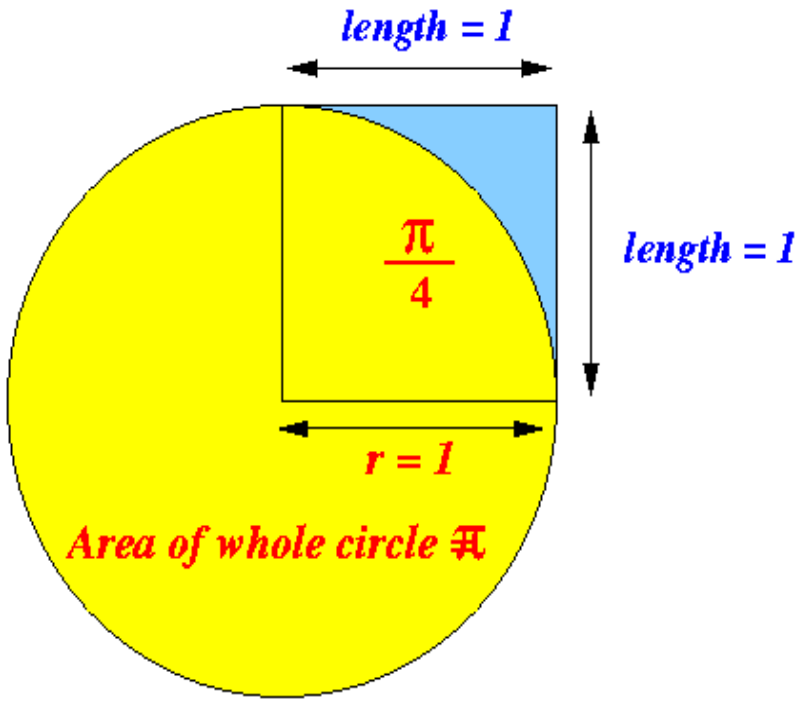
نیاز به تولید شانس در یک کامپیوتر برای:

- ارزیابی رفتار یک سیستم پیچیده (مانند یک شبکه، عملیات اقتصادی، بازی‌های کامپیوتری، خط تولید یک کارخانه و غیره)
- شناسایی ویژگی‌های احتمالاتی یک روش آماری جدید
- شناسایی ویژگی‌های یک روش آماری تحت توزیع نامعلوم (بوت‌استرپ)
- ارزیابی درستی یک مدل آماری
- محاسبه یک انتگرال
- ...

در این جلسه به قسمتی از کاربردهای روش‌های شبیه‌سازی مونت کارلو در استنباط آماری می‌پردازیم

تقریب π: طراحی شبیه‌سازی

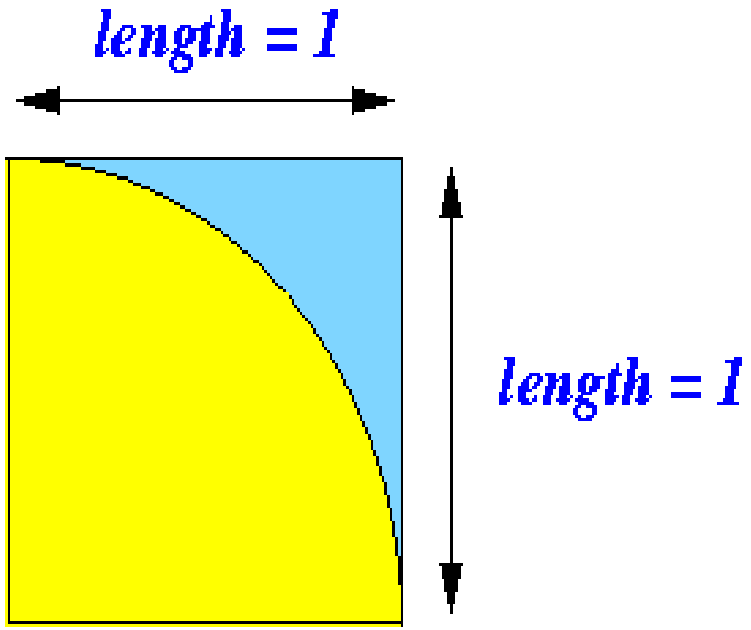
فرض کنید در مورد مقدار $\pi = ۳/۱۴۱۵۹۲۶۵۳۵...$ اطلاعی نداشته باشیم. یک روش برای تقریب این عدد، استفاده از مساحت دایره است:



تقریب π: طراحی شبیه‌سازی

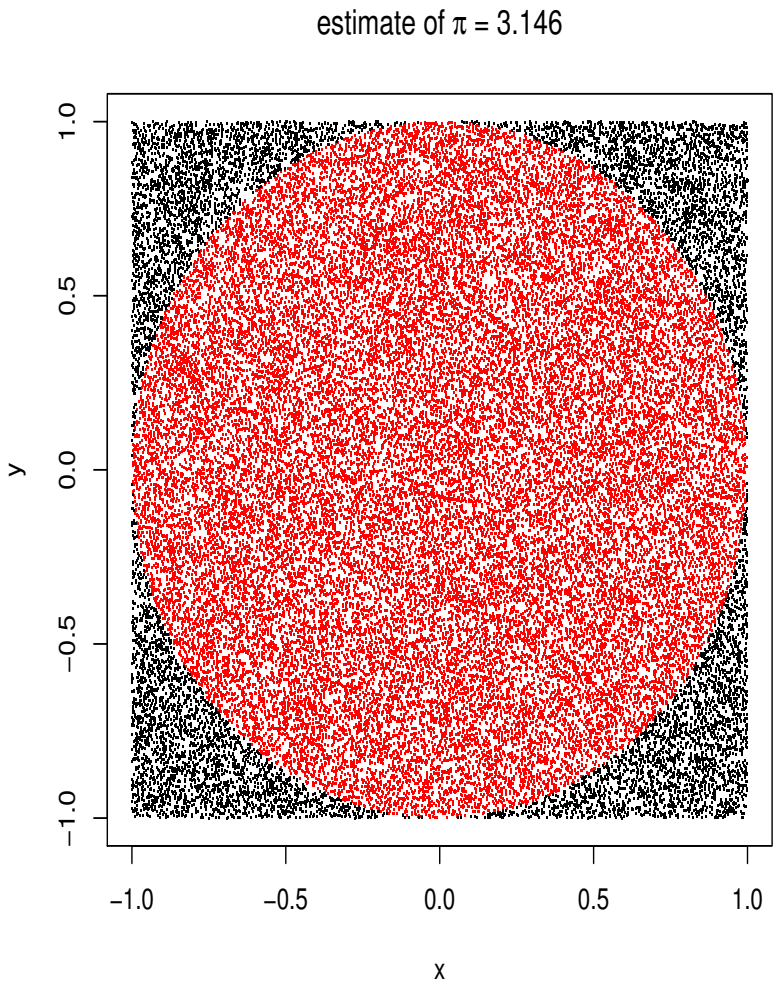
اگر ربع دایره را در نظر بگیریم، نقطه‌ای مانند (x, y) درون آن در نامساوی زیر صدق می‌کند:

$$x^۲ + y^۲ \leq ۱.$$



تقریب π

```
n <- 50000
 x <- runif(n, -1, 1)
 y <- runif(n, -1, 1)
 isInCircle <- (x^2 + y^2 <= 1)
 est <- 4 * sum(isInCircle) / n

plot(x, y, type = "n")
 points(x[isInCircle], y[isInCircle],
 col = "red", pch = ".")
 points(x[!isInCircle], y[!isInCircle],
 col = "black", pch = ".")
 title(substitute(paste("estimate of ", pi,
 " = ", x), list(x = est)))
```



احتمال بلد نیستی؟ شبیه‌سازی چطور؟

مساله انطباق (روز تولد): احتمال آن‌که روز تولد حداقل دو دانشجوی کلاسی با n نفر یکی باشد، چقدر است؟

احتمال بلد نیستی؟ شبیه‌سازی چطور؟

مساله انطباق (روز تولد): احتمال آن‌که روز تولد حداقل دو دانشجوی کلاسی با n نفر یکی باشد، چقدر است؟

$$\begin{aligned} P(\text{حداقل تولد یکسان برای دو نفر}) &= 1 - P(\text{روز تولد هیچکس یکسان نباشد}) \\ &= 1 - \frac{365 \times 364 \times \cdots \times [365 - (n-1)]}{365^n}. \end{aligned}$$

مثلا برای $n = ۶$،

$$۱ - ((۳۶۵ \times ۳۶۴ \times ۳۶۳ \times ۳۶۲ \times ۳۶۱ \times ۳۶۰)/(۳۶۵^۶)) = ۰/۰۴۰۴۶$$

احتمال بلد نیستی؟ شبیه‌سازی چطور؟

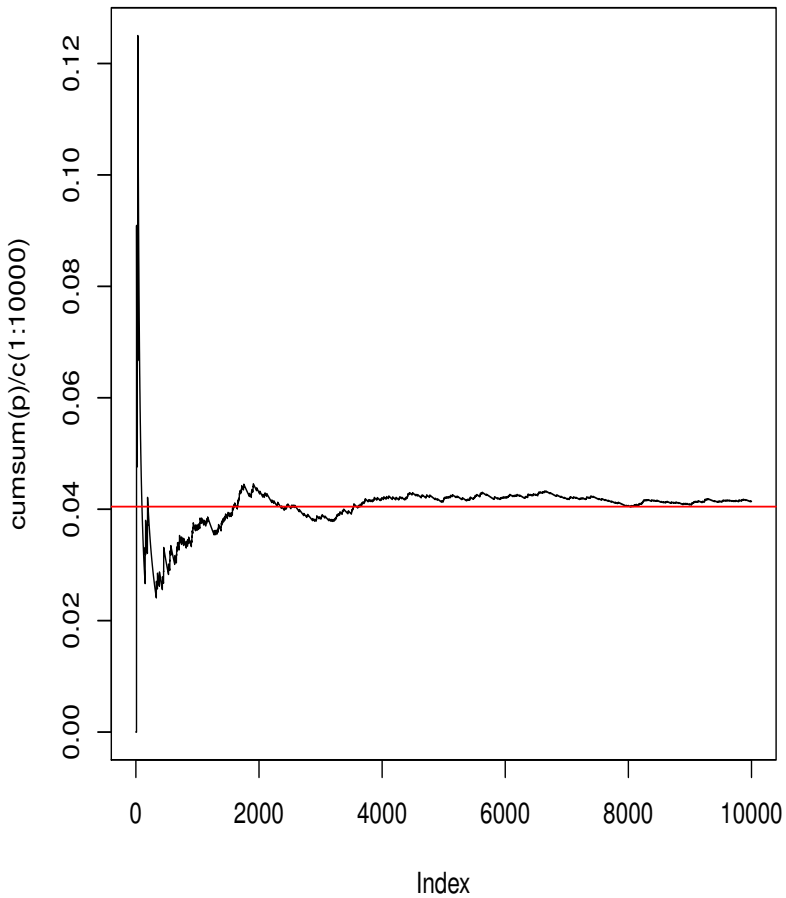
فرض کنید راه حل پاسخ را ندانید !

با یک شبیه‌سازی ساده می‌توان به یک جواب تقریبی (نزدیک) دست یافت:

```
count = 0
for (i in 1:10^6) {
    bdays = sample(1:365, 6, replace = T)
    if (length(unique(bdays)) < 6)
        count = count + 1
}
> count
[1] 40754
> p = count/(10^6)
> p
[1] 0.040754
```

نمایش همگرایی به مقدار واقعی

```
p = rep(0, 10000)
for (i in 1:10000) {
    bdays = sample(1:365, 6, replace = T)
    if (length(unique(bdays)) < 6)
        p[i] = 1
}
#
plot(cumsum(p)/c(1:10000), type = "l")
abline(h = 0.04046248, col = "red")
```



شانس داشتن دقیقا دو نفر از بین ۸ نفر با روز تولد یکسان؟

```
count2 = 0
for (i in 1:10^6) {
    bdays2 = sample(1:365, 8, replace = T)
    if (length(unique(bdays2)) == 7)
        count2 = count2 + 1
}

p2 = count2/(10^6)
> p2
[1] 0.072501
```

تمرین. مقدار نظری این احتمال را محاسبه کنید.

شانس داشتن دقیقا دو نفر از بین ۸ نفر با روز تولد یکسان؟

```
count2 = 0
for (i in 1:10^6) {
    bdays2 = sample(1:365, 8, replace = T)
    if (length(unique(bdays2)) == 7)
        count2 = count2 + 1
}

p2 = count2/(10^6)
> p2
[1] 0.072501
```

تمرین. مقدار نظری این احتمال را محاسبه کنید.

تمرین. برای $n = ۸$ احتمال این‌که حداقل ۲ نفر روزهای تولدی با یک روز اختلاف داشته باشند؟

جواب تمرین دوم اسلاید قبلی

```
count3 = 0
for (i in 1:10^6) {
  bdays3 = sample(1:365, 8, replace = T)
  bdays3 = sort(bdays3)
  if (min(bdays3[2:8] - bdays3[1:7]) == 1)
       count3 = count3 + 1
}
count3
[1] 134238
p3 = count3/(10^6)
p3
[1] 0.134238
```

مسایل ساده تا پیچیده

شبیه‌سازی، می‌تواند ابزاری برای استفاده در مسایل مختلف با درجه‌های پیچیدگی متفاوت باشد

استنباط آماری و شبیه‌سازی

مجموعه روش‌های استنباطی مورد نظر، عبارتند از:

- برآورد پارامترهای توزیع نمونه‌ای یک آماره، MSE، چندک‌ها و غیره
- محاسبه احتمال پوشش یک فاصله اطمینان به منظور مقایسه با سطح اطمینان اسمی
- محاسبه نرخ خطای نوع اول یک آزمون آماری
- برآورد توان یک آزمون
- مقایسه عملکرد روش‌های مختلف برای یک مساله مفروض

عدم قطعیت

برآوردها همیشه همراه با عدم قطعیت هستند.

علاقه به محاسبه توزیع‌های نمونه‌ای برآوردگرها، ناشی از تحقیق در مورد این عدم قطعیت است

اگر بتوان فرآیند تصادفی (مکانیسم مرجع) که داده‌ها از آن تولید شده‌اند را شبیه‌سازی کرد، می‌توان با تولید مکرر نمونه از آن به مطالعه رفتار فرآیند و جستجوی کمیت‌های مورد نظر پرداخت.

تولید نمونه از یک مدل آماری (فرآیند تصادفی) مشخص را بوت‌استرپ پارامتری نیز می‌گویند.

برآوردیابی مونت کارلویی

فرض کنید $X_1, \ldots, X_n$ یک نمونه تصادفی از توزیع متغیر تصادفی X باشد.

$$\hat{\theta} = \hat{\theta}(X_1, \ldots, X_n),$$

به عنوان برآوردی از θ تابعی از نمونه تصادفی است.

اگر نمونه‌های مختلفی از X داشته باشیم، دنباله‌ای از برآوردگرها نیز خواهیم داشت که بر اساس آن می‌توان توزیع برآوردگر را با محاسبه توزیع تجربی، برآورد کرد.

مثلا فرض کنید $X = (X_1, \ldots, X_n)$، همچنین فرض کنید $X^{(1)}, X^{(2)}, \ldots$ دنباله‌ای از نمونه‌های تصادفی تولیدشده از توزیع X باشند. بنابراین

$$\hat{\theta}^{(j)} = \hat{\theta}(X_1^{(j)}, \ldots, X_n^{(j)}), \; j = 1, 2, \ldots,$$

یک دنباله از برآوردگرهای θ می‌باشند. با این دنباله (مثلا m تایی) می‌توان کمیت‌های مورد نظر (توزیع نمونه‌ای $\hat{\theta}$) را برآورد کرد.

مثال

$$X_۱, X_۲ \sim N(۰, ۱).$$

$$\mathbb{E}|X_۱ - X_۲| = \theta = ?$$

با تولید m نمونه $x^{(j)} = (x_۱^{(j)}, x_۲^{(j)}),\ j = ۱, \ldots, m$ می‌توان m برآورد θ را محاسبه کرد:

$$\hat{\theta}^{(j)} = |x_۱^{(j)} - x_۲^{(j)}|,\ j = ۱, \ldots, m,$$

اکنون می‌توان از کمیت‌های نمونه‌ای برای برآورد کمیت‌های دلخواه استفاده کرد:

$$\hat{\theta} = \frac{۱}{m}\sum_{j=۱}^{m}\hat{\theta}^{(j)} = \frac{۱}{m}\sum_{j=۱}^{m}|x_۱^{(j)} - x_۲^{(j)}|.$$

$$\hat{se}(\hat{\theta}) = \frac{۱}{\sqrt{m}}\left\{\frac{۱}{m-۱}\sum_{j=۱}^{m}(\hat{\theta}^{(j)} - \hat{\theta})^۲\right\}^{۱/۲}$$

```
m <- 1000
g <- numeric(m)
for (i in 1:m) {
      x <- rnorm(2)
      g[i] <- abs(x[1] - x[2])
}
est <- mean(g)
> est
[1] 1.130063
> sqrt(sum((g-mean(g))^2))/m
[1] 0.02741058
```

تمرین نشان دهید در این مثال $\mathbb{E}|X_1 - X_2| = \frac{2}{\sqrt{\pi}} \approx 1/12837$ و

$$Var(|X_1 - X_2|) = 2 - \frac{4}{\pi}$$

$$se(\hat{\theta}) = \sqrt{\frac{2 - 4/\pi}{m}} \approx 0/02696$$

برآورد MSE

$$MSE(\hat{\theta}) = \mathbb{E}\left[(\hat{\theta} - \theta)^{۲}\right].$$

$$\hat{MSE} = \frac{۱}{m}\sum_{j=۱}^{m}(\hat{\theta}^{(j)} - \theta)^{۲}$$

تذکر: برای داده‌های واقعی که پارامتر نامعلوم است، محاسبه این کمیت ناممکن است!!

مثال: میانگین پیراسته

میانگین پیراسته در مواردی که داده‌های پرت وجود دارند، به عنوان برآوردگری تنومند مورد استفاده قرار می‌گیرد.
برای یک نمونه n تایی، فرض کنید $X_{(1)}, \ldots, X_{(n)}$ آماره‌های مرتب نمونه باشند. میانگین پیراسته نمونه، میانگین تمام داده‌ها به جز کوچکترین و بزرگترین مشاهدات است.

$$\bar{X}_{[-k]} = \frac{1}{n-2k} \sum_{j=k+1}^{n-k} X_{(j)}.$$

در این مثال برای یک توزیع نرمال استاندارد، به دنبال برآورد MSE میانگین پیراسته سطح اول ($k = 1$) هستیم.

در مثال قبلی (برای توزیع نرمال) پارامتر مورد نظر عبارتست از $\theta = \mathbb{E}(\bar{X}_{[-1]}) = 0$.

مثال: ادامه

اگر میانگین پیراسته نمونه را با T نشان دهیم، برای برآورد $MSE(T)$ مراحل زیر را باید اجرا کنیم:

۱. تولید m برآورد $T^{(j)},\ j = ۱, \ldots, m$:
   - تولید $x_۱^{(j)}, \ldots, x_n^{(j)}$ از توزیع X.
   - مرتب کردن نمونه‌ها و به دست آوردن $x_{(۱)}^{(j)} \leq \ldots x_{(n)}^{(j)}$
   - محاسبه $T^{(j)} = \frac{۱}{n-۲} \sum_{i=۲}^{n-۲} x_{(i)}^{(j)}$

۲. محاسبه $\hat{MSE}(T) = \frac{۱}{m} \sum_{j=۱}^{m} (T^{(j)} - \theta)^۲ = \frac{۱}{m} \sum_{j=۱}^{m} (T^{(j)})^۲$.

دقت کنید که مثل قبل، $T^{(۱)}, \ldots, T^{(m)}$ نمونه‌ای از توزیع نمونه‌ای میانگین پیراسته سطح اول برای یک توزیع نرمال است و می‌توان کمیت‌های مورد علاقه را بر اساس آن‌ها برآورد کرد.

```
n <- 20
m <- 1000
tmean <- numeric(m)
for (i in 1:m) {
     x <- sort(rnorm(n))
     tmean[i] <- sum(x[2:(n-1)]) / (n-2)
}
mse <- mean(tmean^2)
> mse
[1] 0.05571728
> sqrt(sum((tmean - mean(tmean))^2)) / m    #se
[1] 0.00745855
```

میانه

برای میانه هم می‌توان به همین روش عمل کرد:

```
n <- 20
m <- 1000
tmean <- numeric(m)
for (i in 1:m) {
      x <- sort(rnorm(n))
      tmean[i] <- median(x)
}
mse <- mean(tmean^2)
> mse
[1] 0.07431353
> sqrt(sum((tmean - mean(tmean))^2)) / m    #se
[1] 0.008613994
```

دقت کنید که میانه هم در واقع یک میانگین پیراسته است به طوری که همه داده‌ها به جز یک یا دو تا از آن‌ها را (وسط داده‌ها) را پیرایش می‌کند

نرمال آلوده

در این مثال می‌خواهیم MSE را برای میانگین پیراسته سطح k در دو خانواده نرمال و نرمال آلوده، مقایسه کنیم.

یک نرمال آلوده در واقع یک آمیخته از دو توزیع نرمال است. مثلاً:

$$pN(۰, \sigma^۲ = ۱) + (۱ - p)N(۰, \sigma^۲ = ۱۰۰)$$

در این جا پارامتر مورد نظر $\theta = ۰$ است.

دقت کنید که برای تولید از توزیع نرمال آلوده، مثل مبحث توزیع‌های آمیخته، باید ابتدا بر حسب توزیع احتمال $P(\sigma = ۱) = p;\ P(\sigma = ۱۰) = ۱ - p$ مقدار ۱ یا ۱۰ برای σ انتخاب شود و سپس از توزیع نرمال متناظر با آن σ نمونه تولید شود.

```
set.seed(522)
n <- 20
K <- n/2 - 1
m <- 1000
mse <- matrix(0, n/2, 6)
trimmed.mse <- function(n, m, k, p) {
# MC est of mse for k-level trimmed mean of contaminated normal pN(0,1) + (1-p)N(0,100)
tmean <- numeric(m)
for (i in 1:m) {
        sigma <- sample(c(1, 10), size = n,
        replace = TRUE, prob = c(p, 1-p))
        x <- sort(rnorm(n, 0, sigma))
        tmean[i] <- sum(x[(k+1):(n-k)]) / (n-2*k)
     }
    mse.est <- mean(tmean^2)
    se.mse <- sqrt(mean((tmean-mean(tmean))^2)) / sqrt(m)
    return(c(mse.est, se.mse))
}
for (k in 0:K) {
     mse[k+1, 1:2] <- trimmed.mse(n=n, m=m, k=k, p=1.0)
     mse[k+1, 3:4] <- trimmed.mse(n=n, m=m, k=k, p=.95)
     mse[k+1, 5:6] <- trimmed.mse(n=n, m=m, k=k, p=.9)
}
> n*mse
           [,1]      [,2]     [,3]      [,4]      [,5]      [,6]
 [1,] 0.9758575 0.1396680 6.229283 0.3528182 11.484562 0.4792567
 [2,] 1.0194569 0.1425744 1.954048 0.1976301  4.125681 0.2871660
 [3,] 1.0091575 0.1420550 1.304154 0.1614779  1.956211 0.1975056
 [4,] 1.0806823 0.1470113 1.168253 0.1528290  1.577817 0.1776404
 [5,] 1.0478085 0.1446125 1.279664 0.1599728  1.452725 0.1700661
 [6,] 1.1028720 0.1485146 1.395142 0.1670407  1.422924 0.1686735
 [7,] 1.3157601 0.1621938 1.348891 0.1642456  1.574427 0.1773341
 [8,] 1.3767837 0.1659384 1.503163 0.1733524  1.734379 0.1862460
 [9,] 1.3815624 0.1662126 1.525009 0.1746251  1.693839 0.1840468
[10,] 1.4907734 0.1720630 1.646402 0.1814599  1.843068 0.1916661
```

برآورد سطح اطمینان

فاصله‌های اطمینان بر حسب آماره‌هایی به دست می‌آیند که معمولاً توزیع نمونه‌ای آن‌ها نامعلوم هستند یا به دست آوردن آن‌ها کار بسیار دشواری است.

به عنوان مثال بسیاری از روش‌های برآورد معمول، بر اساس پذیره نرمال بودن توزیع جامعه ساخته می‌شوند، در حالی که در عمل موارد متعددی پیش می‌آیند که در آن‌ها توزیع جامعه غیرنرمال باشد.

در چنین مواردی ممکن است یافتن توزیع نمونه‌ای برآوردگرها ناممکن باشد.

در نتیجه فواصل اطمینانی که بر حسب توزیع نرمال جامعه ساخته می‌شوند، ممکن است سطح اطمینان تجربی آن‌ها با سطح اطمینان اسمی یکی نباشد.

با روش‌های مونت کارلو می‌توان سطح اطمینان این فواصل را برآورد کرد.

سطح اطمینان

فرض کنید (U, V) یک برآورد فاصله‌ای برای θ باشد. U و V آماره‌هایی هستند که توزیع نمونه‌ای آن‌ها به توزیع جامعه F_X وابسته است.

سطح اطمینان، احتمالی است که فاصله تصادفی (U, V) مقدار واقعی θ را در بر می‌گیرد. بنابراین محاسبه سطح اطمینان یک فاصله اطمینان، یک مساله انتگرال‌گیری است.

فرض کنید بخواهیم یک برآورد فاصله‌ای برای واریانس به دست آوریم.

مشهور است که چنین برآوردی به پذیره توزیع نرمال برای جامعه خیلی حساس است. یعنی تخطی از توزیع نرمال می‌تواند نتایج به شدت گمراه‌کننده‌ای در بر داشته باشد.

از نمونه‌گیری مونت کارلو می‌توان برای برآورد سطح اطمینان واقعی یک فاصله اطمینان با فرض توزیع نرمال جامعه وقتی که توزیع جامعه واقعا نرمال نیست، استفاده کرد.

فاصله اطمینان برای واریانس

ابتدا بر اساس توزیع واقعی نرمال مساله را بررسی می‌کنیم.

$$X_1, \ldots, X_n \sim N(\mu, \sigma^2)$$

$$V = \frac{(n-1)S^2}{\sigma^2} \sim \chi^2(n-1).$$

یک فاصله اطمینان یک طرفه در سطح $100(1-\alpha)\%$ عبارتست از $(0, \frac{(n-1)S^2}{\chi^2_\alpha})$ که در آن χ^2_α چندک αام توزیع $\chi^2(n-1)$ است.

اگر توزیع جامعه نرمال با واریانس σ^2 باشد، آنگاه احتمال آنکه فاصله مقدار واقعی σ^2 را پوشش دهد برابر $1-\alpha$ است.

فاصله اطمینان برای واریانس: ادامه

برآورد کران بالای فاصله برای $N(۰, \sigma^۲ = ۴)$ به صورت زیر قابل انجام است:

```
n <- 20
alpha <- .05
x <- rnorm(n, mean=0, sd=2)
UCL <- (n-1) * var(x) / qchisq(alpha, df=n-1)
```

در این مثال، وقتی که توزیع جامعه واقعا نرمال است، سطح اطمینان به طور دقیق قابل اندازه‌گیری است:

$$P(\frac{۱۹S^۲}{\chi^۲_{۰/۰۵}(۱۹)} > ۴) = P(\frac{(n-۱)S^۲}{\sigma^۲} > \chi^۲_{۰/۰۵}(n-۱)) = ۰/۹۵.$$

اگر تولید نمونه و محاسبه این فاصله به دفعات زیادی تکرار شود، تقریباً ٪۹۵ فاصله‌های عددی ساخته شده باید مقدار واقعی $\sigma^۲ = ۴$ را در بر داشته باشند.

برآورد سطح اطمینان تجربی

فرض کنید $X \sim F_X$ و θ پارامتر مورد نظر باشد. برای برآورد سطح اطمینان باید مراحل زیر اجرا شوند:

۱. برای هر تکرار مونت کارلو، $j = ۱, \ldots, m$:
   - نمونه $X_۱^{(j)}, \ldots, X_n^{(j)}$ را تولید کن
   - فاصله اطمینان C_j را محاسبه کن
   - متغیر نشانگر $y_j = I(\theta \in C_j)$ را محاسبه کن

۲. سطح اطمینان تجربی با کمیت $\bar{y} = \frac{۱}{m} \sum_{j=۱}^{m} y_j$ محاسبه می‌شود

برآوردگر $\bar{y}$ نسبت نمونه‌ای برای برآورد سطح اطمینان واقعی $۱ - \alpha^*$ است. بنابراین چون y_j ها متغیرهای برنولی هستند، $Var(\bar{y}) = \frac{(۱-\alpha^*)\alpha^*}{m}$. در نتیجه برآوردی برای خطای معیار برآورد نیز عبارتست از

$$\hat{se}(\bar{y}) = \sqrt{\frac{(۱ - \bar{y})\bar{y}}{m}}.$$

برآورد مونت کارلو در مثال واریانس توزیع نرمال

```
n <- 20
alpha <- .05
UCL <- replicate(1000, expr = {
       x <- rnorm(n, mean = 0, sd = 2)
       (n-1) * var(x) / qchisq(alpha, df = n-1)
       } )
# compute the mean to get the confidence level
> mean(UCL > 4)
[1] 0.95
```

تمرین. در مورد دستور *replicate* جزییات را همراه با مثال تشریح کنید

تخطی از پذیره نرمال

برآورد فاصله‌ای واریانس نسبت به پذیره نرمال بودن توزیع جامعه حساس است. به این معنی که اگر توزیع جامعه نرمال نباشد، سطح اطمینان واقعی با مقدار اسمی آن اختلاف دارد.

سطح اطمینان واقعی به توزیع آماره S^2 بستگی دارد.

$$P(\frac{(n-1)S^2}{\chi^2_\alpha} > \sigma^2) = P(S^2 > \frac{\sigma^2\chi^2_\alpha}{n-1}) = 1 - G(\frac{\sigma^2\chi^2_\alpha}{n-1}),$$

که در آن G تابع توزیع آماره S^2 است.
اگر توزیع واقعی جامعه نرمال نباشد، مساله برآورد سطح اطمینان به یک مساله انتگرال‌گیری منتهی می‌شود:

$$G(t) = P(S^2 \leq c_\alpha) = \int_0^{c_\alpha} g(x)dx,$$

به طوری که $g(\cdot)$ تابع چگالی (نامعلوم) S^2 و $c_\alpha = \frac{\sigma^2\chi^2_\alpha}{n-1}$.

سطح اطمینان تجربی

توجه داشته باشید که در این جا برای محاسبه مونت کارلو انتگرال بالا، شرط معلوم بودن ضابطه $g(\cdot)$ لازم نیست و تنها کافی است بتوان از این توزیع نمونه‌هایی تولید کرد

به عنوان مثال فرض کنید توزیع واقعی جامعه $\chi^2(2)$ باشد (واریانس جامعه ۴ است اما توزیع نرمال نیست).

```
n <- 20
alpha <- .05
UCL <- replicate(1000, expr = {
       x <- rchisq(n, df = 2)
       (n-1) * var(x) / qchisq(alpha, df = n-1)
       } )
> mean(UCL > 4)
[1] 0.777
```

آزمون فرضیه‌ها

$$H_0 : \theta \in \Theta_0 \quad vs \quad H_1 : \theta \in \Theta_1$$

$$\Theta_0 \cup \Theta_1 = \Theta, \quad \Theta_0 \cap \Theta_1 = \emptyset$$

سطح معنی‌داری یک آزمون را با α نشان می‌دهند که کران بالا برای احتمال رخداد خطای نوع اول است.

تابع توان یک آزمون، $\pi(\theta)$، احتمال رد فرضیه H_0 است. بنابراین

$$\alpha = \sup_{\theta \in \Theta_0} \pi(\theta).$$

اگر T آماره آزمون و T^* مقدار مشاهده شده این آماره باشد، آنگاه T^* معنی‌دار گفته می‌شود اگر تصمیم مبتنی بر آن منجر به رد فرضیه H_0 شود.

احتمال معنی‌داری یا همان pـ مقدار، کوچکترین مقدار ممکن α است به طوری که آماره آزمون مشاهده شده، معنی‌دار باشد.

استفاده از pـ مقدار باید با ملاحظه صورت گیرد!!!

نرخ خطای نوع اول تجربی

احتمال خطای نوع اول، احتمال شرطی رد فرضیه H_0 است زمانی که این فرضیه درست باشد.

بنابراین اگر آزمون تحت شرایط فرضیه صفر، H_0، به دفعات تکرار شود، نرخ خطای نوع اول مشاهده شده باید تقریباً حداکثر α باشد.

نرخ خطای نوع اول تجربی، در یک شبیه‌سازی مونت کارلو، نسبت آماره‌های آزمون معنی‌دار در نمونه‌های مونت کارلوی تولید شده است.

مراحل برآورد نرخ خطای نوع اول تجربی به صورت زیر است:

۱. برای هر تکرار مونت کارلو $j = 1, \ldots, m$:
   - نمونه $x_1^{(j)}, \ldots, x_n^{(j)}$ را تحت فرضیه صفر تولید کن
   - آماره آزمون T_j را محاسبه کن
   - اگر فرضیه H_0 رد می‌شود، قرار بده $I_j = 1$ در غیر این صورت قرار بده $I_j = 0$

۲. نسبت آزمون‌های معنی‌دار را با کمیت $\frac{1}{m}\sum_{j=1}^{m} I_j$ محاسبه کن

خطای استاندارد برآورد

پارامتر برآورد شده، یعنی نرخ خطای نوع اول تجربی، احتمالی است که با نسبت نمونه، $\hat{p}$، برآورد می‌شود. بنابراین برآوردی برای خطای استاندارد عبارتست از

$$\hat{se}(\hat{p}) = \sqrt{\frac{\hat{p}(۱ - \hat{p})}{m}} \leq \frac{۰/۵}{\sqrt{m}}.$$

مثال: آزمون t

$$X_۱, \ldots, X_{۲۰} \sim N(\mu, \sigma^۲),$$

$$H_۰ : \mu = ۵۰۰ \quad vs \quad H_۱ : \mu > ۵۰۰ \quad \& \quad \alpha = ۰/۰۵$$

برای آزمون این فرضیه‌ها از آماره t استفاده می‌شود:

$$T^* = \frac{\bar{X} - ۵۰۰}{S/\sqrt{۲۰}} \sim t(۱۹).$$

مقادیر بزرگ T^* از فرضیه جانشین $H_۱$ پشتیبانی می‌کنند.

در این مثال، از نمونه‌های مونت کارلو برای برآورد نرخ خطای نوع اول وقتی که $\sigma = ۱۰۰$ است، استفاده می‌کنیم.

برآورد خطای استاندارد برآورد نرخ خطای نوع اول باید نزدیک به $\sqrt{\frac{۰/۰۵ \times ۰/۹۵}{m}} \approx ۰/۰۰۲۲$ باشد.

مثال: ادامه

```
n <- 20
alpha <- .05
mu0 <- 500
sigma <- 100
m <- 10000  # number of replicates
p <- numeric(m)  # storage for p-values
for (j in 1:m) {
     x <- rnorm(n, mu0, sigma)
     ttest <- t.test(x, alternative = "greater", mu = mu0)
     p[j] <- ttest$p.value
}
p.hat <- mean(p < alpha)
se.hat <- sqrt(p.hat * (1 - p.hat) / m)
> print(c(p.hat, se.hat))
[1] 0.055700000 0.002293415
```

مثال: آزمون نرمال بودن بر اساس چولگی

یکی از آزمون‌های بررسی نرمال بودن یک نمونه، مبتنی بر پارامتر چولگی است:

$$\gamma_1 = \frac{\mathbb{E}[(X-\mu_X)^3]}{\sigma_X^3},$$

یک توزیع، متقارن است اگر برای آن $\gamma_1 = 0$. ضریب چولگی نمونه به صورت زیر به دست می‌آید:

$$b_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^3}{(\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2)^{3/2}}.$$

نشان داده شده است اگر توزیع X نرمال باشد، آنگاه توزیع مجانبی b_1 نیز نرمال با میانگین صفر و واریانس $\frac{6}{n}$ خواهد بود.

در آزمون فرضیه‌های

$$H_0 : \gamma_1 = 0 \quad vs \quad H_1 : \gamma_1 \neq 0,$$

به ازای مقادیر بزرگ $|b_1|$، فرضیه نرمال بودن رد می‌شود.

مثال: ادامه

در این آزمون، توزیع نمونه‌ای آماره چولگی بر اساس پذیره نرمال بودن جامعه به دست می‌آید.

این توزیع نمونه‌ای تقریبی، برای حجم نمونه‌های کم و متوسط خوب نیست و سرعت همگرایی توزیع چولگی به نرمال کند است.

در این مثال، نرخ خطای نوع اول آزمون مبتنی بر چولگی را در سطح $\alpha = ۰/۰۵$ و برای نمونه‌های $n = ۱۰, ۲۰, ۳۰, ۵۰, ۱۰۰, ۵۰۰$، بر اساس توزیع مجانبی آماره چولگی، ارزیابی می‌کنیم.

مقادیر بحرانی آزمون برای این حجم‌های نمونه به صورت زیر محاسبه می‌شوند:

```
n <- c(10, 20, 30, 50, 100, 500) #sample sizes
cv <- qnorm(.975, 0, sqrt(6/n))  #crit. values for each n
> n
[1]  10  20  30  50 100 500
> cv
[1] 1.5181816 1.0735165 0.8765225 0.6789514 0.4800912 0.2147033
```

مثال: ادامه

```
sk <- function(x) {
      # computes the sample skewness coeff.
      xbar <- mean(x)
      m3 <- mean((x - xbar)^3)
      m2 <- mean((x - xbar)^2)
      return( m3 / m2^1.5 )
}
####
p.reject <- numeric(length(n)) #to store sim. results
m <- 10000
for (i in 1:length(n)) {
     sktests <- numeric(m)       #test decisions
     for (j in 1:m) {
          x <- rnorm(n[i])
          # test decision is 1 (reject) or 0
          sktests[j] <- as.integer(abs(sk(x)) >= cv[i] )
          }
     p.reject[i] <- mean(sktests) # proportion rejected
}
> p.reject
[1] 0.0141 0.0287 0.0358 0.0382 0.0406 0.0477
```

نتایج نشان می‌دهد، استفاده از تقریب نرمال برای توزیع آماره چولگی برای نمونه‌های $n \leq ۵۰$ مناسب نیست و برای نمونه‌های بزرگ، حتی $n = ۵۰۰$، هم مورد شک و تردید است.

مثال: ادامه

برای نمونه‌های کوچک باید از واریانس دقیق آماره استفاده کرد:

$$Var(b_1) = \frac{6(n-2)}{(n+1)(n+3)}.$$

```
cv <- qnorm(.975, 0, sqrt(6*(n-2)/((n+1)*(n+3))))
> round(cv, 4)
[1] 1.1355 0.9268 0.7943 0.6398 0.4660 0.2134
> n
[1]  10  20  30  50 100 500
>     p.reject
[1] 0.0540 0.0548 0.0504 0.0495 0.0491 0.0535
```

این برآوردها نزدیکتر به سطح اسمی ۰/۰۵ هستند.

توان آزمون

توان یک آزمون در تصمیم‌گیری درست، با تابع توان بیان می‌شود که احتمال رد H_0 به درستی می‌باشد.

البته تابع توان برای کل فضای پارامتر تعریف می‌شود:

$$\pi : \Theta \to [0, 1], \quad \pi(\theta) = P_\theta(RH_0).$$

بنابراین برای هر $\theta_1 \in \Theta_1$ احتمال رخداد خطای نوع دوم $1 - \pi(\theta_1)$ است.

آزمونی ترجیح داده می‌شود که دارای احتمال خطای کمتری باشد. خطای نوع اول که در سطح α کنترل شده است. بنابراین مقایسه آزمون‌ها برای فرضیه‌های یکسان و در سطح برابر، در واقع مقایسه توان آن‌ها در زیرفضای Θ_1 است.

اگر توان یک آزمون را به طور تحلیلی نتوان محاسبه کرد (که خیلی هم معمول است)، می‌توان از روش مونت کارلو آن را تقریب زد.

برآورد مونت کارلوی توان آزمون

برای محاسبه توان آزمون در یک نقطه از فضای Θ_1 باید به صورت زیر عمل کرد:

۱. انتخاب یک مقدار مشخص از فضای پارامتر تحت فرضیه جانشین، $\theta_1 \in \Theta_1$

۲. برای هر تکرار مونت کارلو $j = 1, \ldots, m$:

الف) تولید نمونه $x_1^{(j)}, \ldots, x_n^{(j)}$ تحت فرضیه جانشین $\theta = \theta_1$

ب) محاسبه آماره آزمون T_j

ج) ثبت تصمیم آزمون، یعنی اگر H_0 رد شود قرار بده $I_j = 1$ و در غیر این صورت قرار بده $I_j = 0$

۳. محاسبه نسبت آزمون‌های معنی‌دار یعنی $\hat{\pi}(\theta_1) = \frac{1}{m} \sum_{j=1}^{m} I_j$

توان تجربی

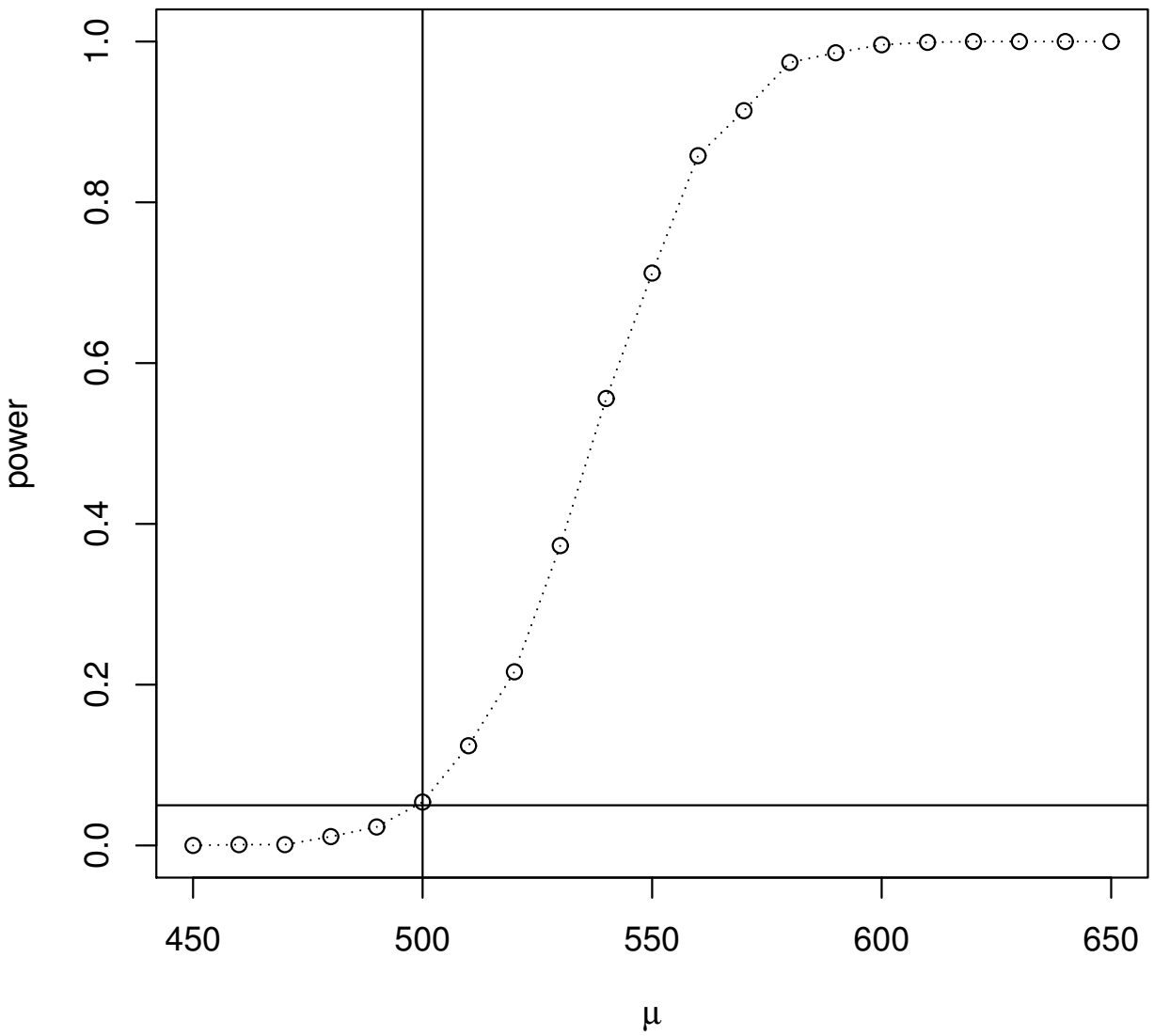
برای مثال آزمون t، توان آزمون را در چند نقطه از Θ_1، محاسبه می‌کنیم:

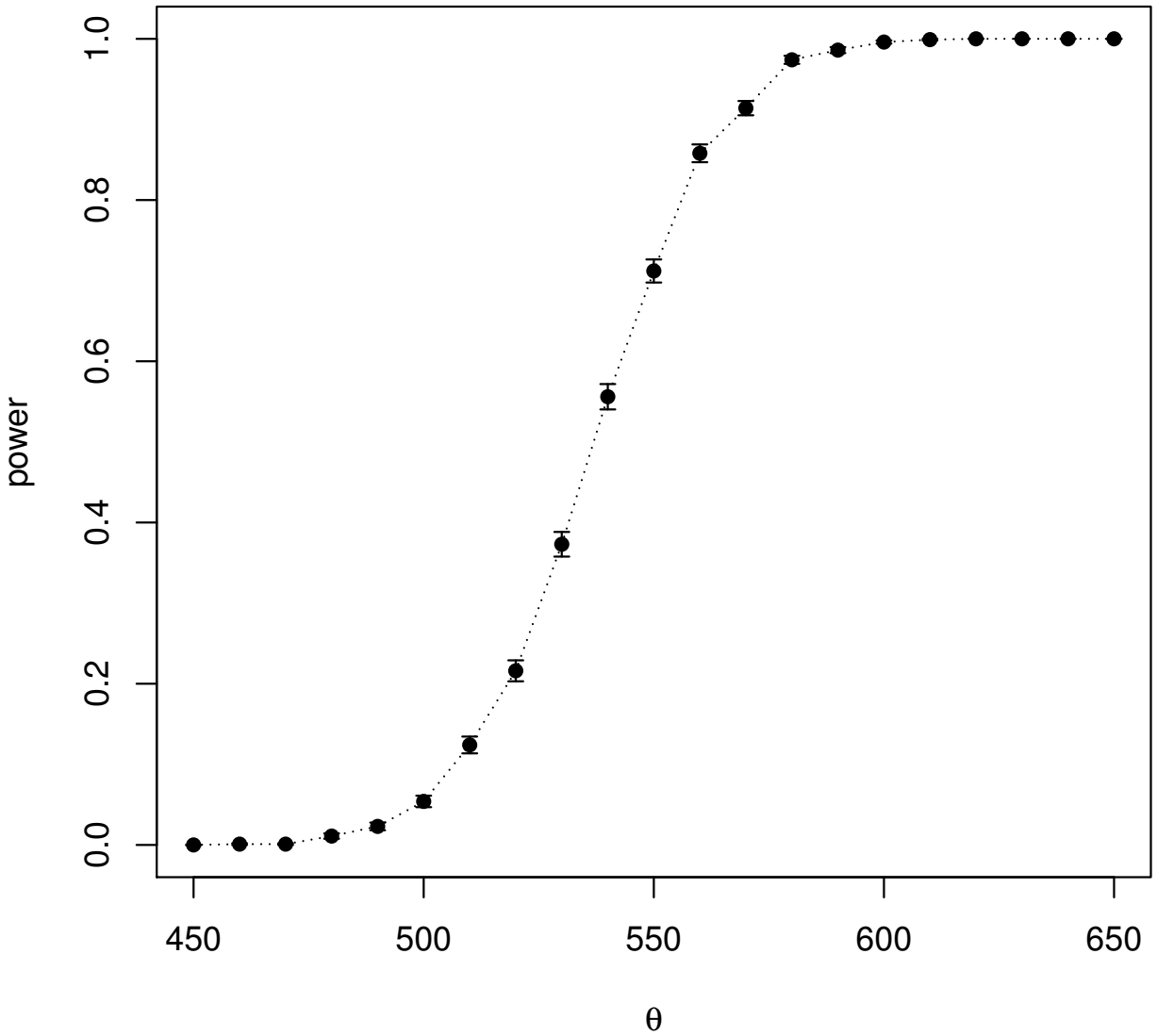
```
n <- 20
m <- 1000
mu0 <- 500
sigma <- 100
mu <- c(seq(450, 650, 10)) # alternatives
M <- length(mu)
power <- numeric(M)
for (i in 1:M) {
     mu1 <- mu[i]
     pvalues <- replicate(m, expr = {
             # simulate under alternative mu1
             x <- rnorm(n, mean = mu1, sd = sigma)
             ttest <- t.test(x,
             alternative = "greater", mu = mu0)
             ttest$p.value  } )
     power[i] <- mean(pvalues <= .05)
}
```

مثال: ادامه

برای رسم نمودار توان آزمون به همراه خطاهای متناظر در نقاطی که توان محاسبه شده است، یعنی $\hat{\pi}(\theta) \mp \hat{se}(\hat{\pi}(\theta))$، از دستور $errbar$ در بسته $Hmisc$ می‌توان استفاده کرد:

```
par(ask = TRUE)
library(Hmisc)  #for errbar
plot(mu, power, xlab = bquote(mu))
abline(v = mu0, lty = 1)
abline(h = .05, lty = 1)
lines(mu, power, lty=3)
# add standard errors
se <- sqrt(power * (1-power) / m)
errbar(mu, power, yplus = power+se, yminus = power-se,
xlab = bquote(theta))
lines(mu, power, lty=3)
detach(package:Hmisc)
par(ask = FALSE)
```


power
0.0
0.2
0.4
0.6
0.8
1.0
450
500
550
600
650
μ


power
0.0
0.2
0.4
0.6
0.8
1.0
450
500
550
600
650
θ

توان آزمون نرمال بودن بر اساس چولگی

برای توزیع نرمال آلوده

$$(۱ - p)N(۰, \sigma^۲ = ۱) + pN(۰, \sigma^۲ = ۱۰۰),$$

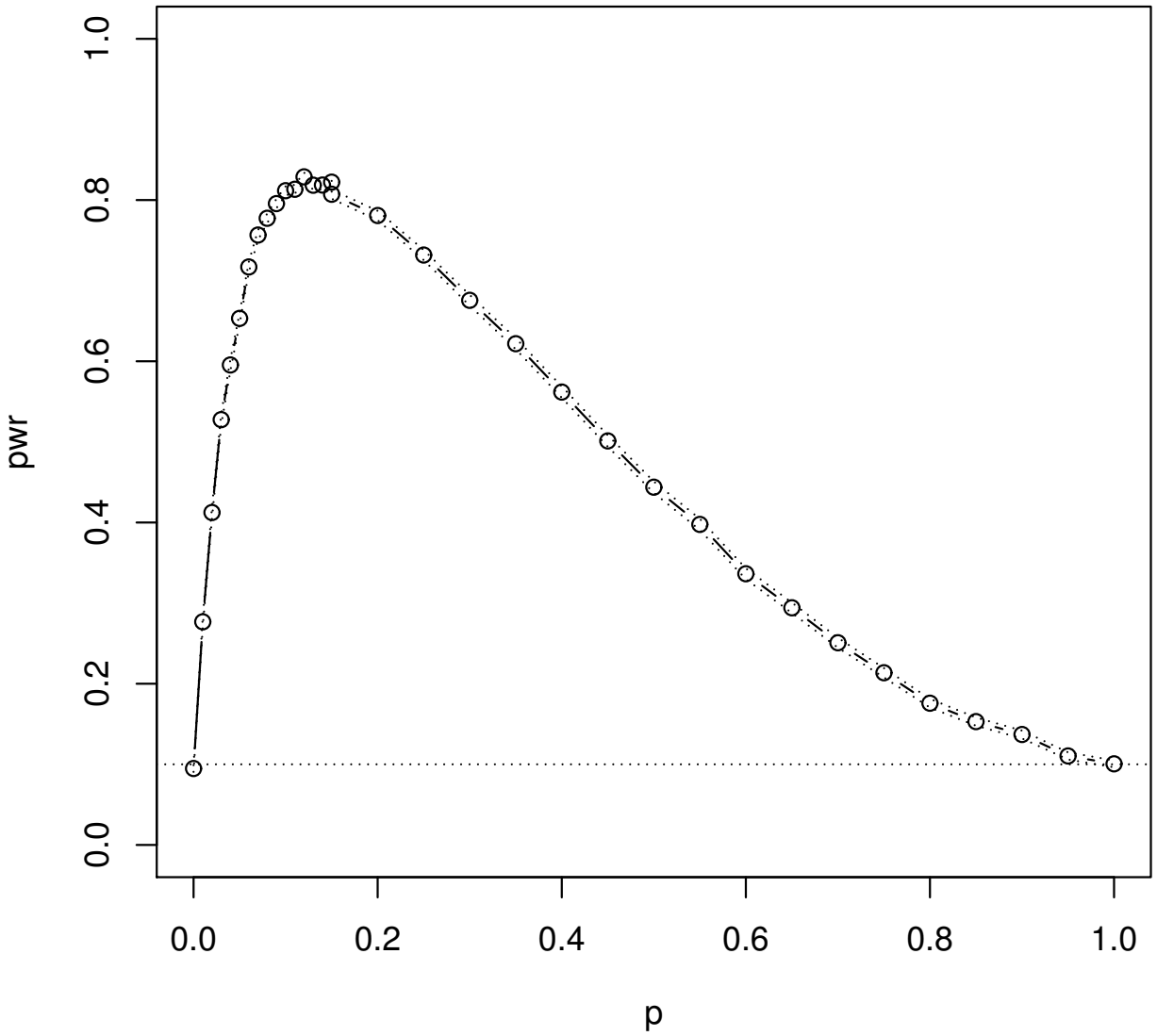
اگر $p = ۰$ یا $p = ۱$، توزیع نرمال است ولی برای هر $۰ < p < ۱$ توزیع نانرمال خواهد بود.

می‌توان توان آزمون سنجش نرمال بر اساس چولگی را برای دنباله‌ای از فرضیه‌های جانشین با تغییر p برآورد کرد و نمودار آن را رسم کرد.

فرض کنید $\alpha = ۰/۱$ و حجم نمونه $n = ۳۰$ انتخاب شود.

```
alpha <- .1
n <- 30
m <- 5000
p <- c(seq(0, .15, .01), seq(.15, 1, .05))
N <- length(p)
pwr <- numeric(N)
# critical value for the skewness test
cv <- qnorm(1-alpha/2, 0, sqrt(6*(n-2) / ((n+1)*(n+3))))
```

```
for (j in 1:N) {              # for each p
     e <- p[j]
     sktests <- numeric(m)
     for (i in 1:m) {          # for each replicate
          sigma <- sample(c(1, 10), replace = TRUE,
          size = n, prob = c(1-e, e))
          x <- rnorm(n, 0, sigma)
          sktests[i] <- as.integer(abs(sk(x)) >= cv)
          }
      pwr[j] <- mean(sktests)
}
# plot power vs p
plot(p, pwr, type = "b",
xlab = bquote(p), ylim = c(0,1))
abline(h = .1, lty = 3)
se <- sqrt(pwr * (1-pwr) / m)  # add standard errors
lines(p, pwr+se, lty = 3)
lines(p, pwr-se, lty = 3)
```


pwr
0.0
0.2
0.4
0.6
0.8
1.0
p
0.0
0.2
0.4
0.6
0.8
1.0

مقایسه توان آزمون‌ها

از شبیه‌سازی مونت کارلو، اغلب برای مقایسه عملکرد روش‌های مختلف استفاده می‌شود.

برای آزمون سنجش نرمال، عملکرد آزمون مبتنی بر چولگی را با دو آزمون معروف دیگر مقایسه می‌کنیم:

۱. آزمون شاپیرو ـ ویلک: این آزمون را در R می‌توان با تابع $shapiro.test$ انجام داد

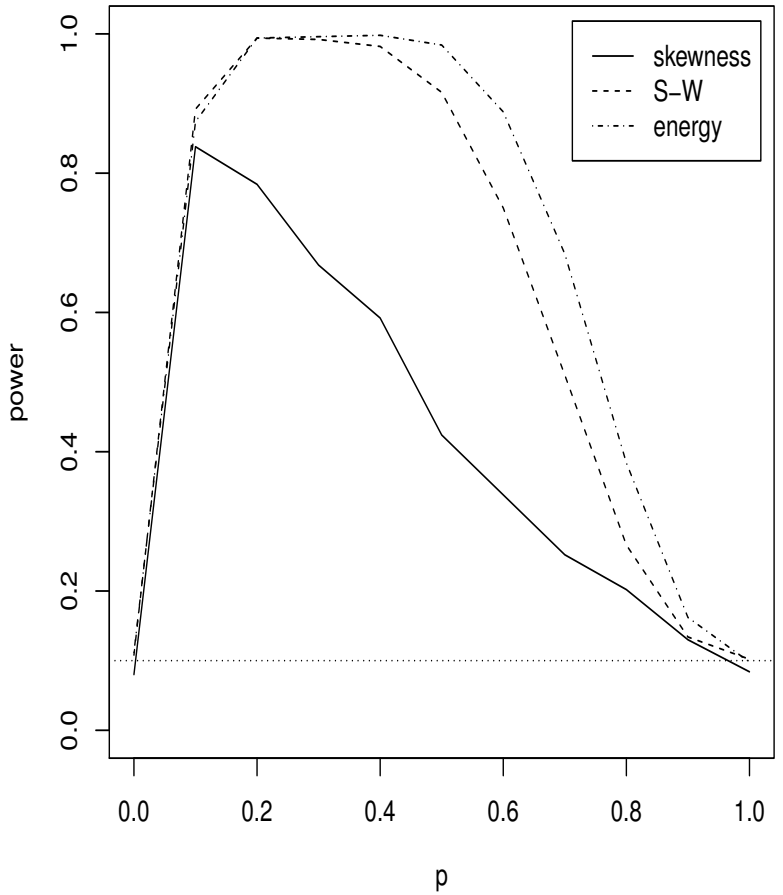
۲. آزمون انرژی: این آزمون برای سنجش نرمال چندمتغیره معرفی شده است و مبتنی بر یک معیار فاصله معروف به فاصله انرژی است. این آزمون برای یک متغیره خیلی شبیه به آزمون اندرسون ـ دارلینگ عمل می‌کند. این آزمون توسط دستور $mvnorm.etest$ که در بسته $energy$ موجود است، قابل اجراست.

برای اطلاعات بیشتر در مورد این آزمون‌ها به کتاب مراجعه کنید.

برای این مثال، $\alpha = ۰/۱$ و فرضیه جانشین بر اساس توزیع نرمال آلوده تنظیم می‌شود.

```
library(energy)
alpha <- .1
n <- 30; m <- 500  # try small m for a trial run
test1 <- test2 <- test3 <- numeric(m)
# critical value for the skewness test
cv <- qnorm(1-alpha/2, 0, sqrt(6*(n-2) / ((n+1)*(n+3))))
sim <- matrix(0, 11, 4)
# estimate power
for (i in 0:10) {
     p <- i * .1
     for (j in 1:m) {
          e <- p
          sigma <- sample(c(1, 10), replace = TRUE,
          size = n, prob = c(1-e, e))
          x <- rnorm(n, 0, sigma)
          test1[j] <- as.integer(abs(sk(x)) >= cv)
          test2[j] <- as.integer(
                      shapiro.test(x)$p.value <= alpha)
          test3[j] <- as.integer(
                      mvnorm.etest(x, R=200)$p.value <= alpha)
          }
     print(c(p, mean(test1), mean(test2), mean(test3)))
     sim[i+1, ] <- c(p, mean(test1), mean(test2), mean(test3))
}
[1] 0.000 0.080 0.108 0.112
[1] 0.100 0.838 0.892 0.874
[1] 0.200 0.784 0.994 0.994
[1] 0.300 0.668 0.992 0.996
[1] 0.400 0.592 0.982 0.998
[1] 0.500 0.424 0.916 0.984
[1] 0.600 0.338 0.750 0.888
[1] 0.700 0.252 0.510 0.684
[1] 0.800 0.202 0.266 0.384
[1] 0.900 0.130 0.134 0.162
[1] 1.000 0.084 0.102 0.098
```

```
# plot the empirical estimates of power
plot(sim[,1], sim[,2], ylim = c(0, 1), type = "l",
xlab = bquote(p), ylab = "power")
lines(sim[,1], sim[,3], lty = 2)
lines(sim[,1], sim[,4], lty = 4)
abline(h = alpha, lty = 3)
legend("topright", 1, c("skewness", "S-W", "energy"),
lty = c(1,2,4), inset = .02)
```



محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه دهم: روش‌های خودگردان (*Bootstrap Methods*)

حسین باغیشنی

دانشگاه شاهرود

۲ آذر ۱۳۹۳

استنباط آماری با مشاهدات ناقص

آمار، مطالعه روش‌های استخراج استنباط‌هایی از داده‌های ناقص و محدود شده است. مثلا مایلیم بدانیم:

- چطور یک نرون در مغز یک موش به کشیده شدن یکی از موهای صورتش، پاسخ می‌دهد

استنباط آماری با مشاهدات ناقص

آمار، مطالعه روش‌های استخراج استنباط‌هایی از داده‌های ناقص و محدود شده است. مثلا مایلیم بدانیم:

- چطور یک نرون در مغز یک موش به کشیده شدن یکی از موهای صورتش، پاسخ می‌دهد
- چند تا موش در جنگل ابر زندگی می‌کنند

استنباط آماری با مشاهدات ناقص

آمار، مطالعه روش‌های استخراج استنباط‌هایی از داده‌های ناقص و محدود شده است. مثلا مایلیم بدانیم:

- چطور یک نرون در مغز یک موش به کشیده شدن یکی از موهای صورتش، پاسخ می‌دهد
- چند تا موش در جنگل ابر زندگی می‌کنند
- در اردیبهشت، تا چه ارتفاعی آب رودخانه زاینده‌رود زیر پل خواجو بالا خواهد آمد

استنباط آماری با مشاهدات ناقص

آمار، مطالعه روش‌های استخراج استنباط‌هایی از داده‌های ناقص و محدود شده است. مثلا مایلیم بدانیم:

- چطور یک نرون در مغز یک موش به کشیده شدن یکی از موهای صورتش، پاسخ می‌دهد
- چند تا موش در جنگل ابر زندگی می‌کنند
- در اردیبهشت، تا چه ارتفاعی آب رودخانه زاینده‌رود زیر پل خواجو بالا خواهد آمد
- متوسط دمای هوا در شاهرود در طول سال، چقدر است
- ...

عدم قطعیت

برای همه این چیزها، مجموعه‌ای از مشاهدات داریم. اما می‌دانیم

عدم قطعیت

برای همه این چیزها، مجموعه‌ای از مشاهدات داریم. اما می‌دانیم

- داده‌ها ناقص هستند
- تکرار آزمایش یا مشاهدات، حتی اگر نهایت سعی خود را برای ثابت نگه داشتن شرایط به کار ببریم، همیشه کم و بیش نتایج متفاوتی خواهد داشت

عدم قطعیت

برای همه این چیزها، مجموعه‌ای از مشاهدات داریم. اما می‌دانیم

- داده‌ها ناقص هستند
- تکرار آزمایش یا مشاهدات، حتی اگر نهایت سعی خود را برای ثابت نگه داشتن شرایط به کار ببریم، همیشه کم و بیش نتایج متفاوتی خواهد داشت

بنابراین مضحک است اگر استنباط استخراج‌شده از داده‌ها را قطعی بدانیم.

مدل‌های تصادفی و تابعی‌ها

اگرچه تکرار یک آزمایش نتایج متفاوتی به دست می‌دهد، بعضی از نتایج نسبت به سایرین بیشتر رخ می‌دهند.

و فراونی‌های نسبی این پیشامدها، پایدار هستند.

بنابراین، مکانیسم تولید داده‌ها را می‌توان به وسیله توزیع‌های احتمالی و فرآیندهای تصادفی، مدل‌بندی کرد.

مدل‌های تصادفی و تابعی‌ها

اگرچه تکرار یک آزمایش نتایج متفاوتی به دست می‌دهد، بعضی از نتایج نسبت به سایرین بیشتر رخ می‌دهند.

و فراونی‌های نسبی این پیشامدها، پایدار هستند.

بنابراین، مکانیسم تولید داده‌ها را می‌توان به وسیله توزیع‌های احتمالی و فرآیندهای تصادفی، مدل‌بندی کرد.

کمیت‌هایی مانند مثال‌هایی که ذکر کردیم، به صورت توابعی از مدل تصادفی، یعنی توزیع احتمال زیربنایی، نمایش داده می‌شوند.

چون یک تابع از یک تابع را تابعی می‌نامند، و کمیت‌های مورد نظر توابعی از تابع توزیع احتمال واقعی هستند، آن‌ها را تابعی‌های آماری (پارامتر)، $\theta(F)$، نیز می‌نامند.

تابعی‌ها می‌توانند عدد حقیقی (مانند تعداد کل موش‌ها)، بردار یا کل یک منحنی (منحنی رگرسیون قد افراد بر روی وزن)، باشند.

استنباط آماری، برآورد کردن این تابعی‌ها یا آزمودن فرضیه‌هایی در مورد آن‌هاست.

مدل‌های تصادفی، عدم قطعیت، استنباط آماری

برآورد پارامترها و سایر استنباط‌ها، توابعی از مقادیر داده هستند.

این به آن معنی است که: آن‌ها عدم‌قطعیت همراه با فرآیند تصادفی زیربنایی را به ارث می‌برند.

اگر آزمایش تکرار شود، ما داده‌های متفاوتی خواهیم داشت اما با یک توزیع قطعی مشخص و اجرای یک روش استنباطی ثابت، نتایج استنباطی متفاوتی در بر خواهد داشت اما دوباره با یک توزیع قطعی مشخص.

مدل‌های تصادفی، عدم قطعیت، استنباط آماری

برآورد پارامترها و سایر استنباط‌ها، توابعی از مقادیر داده هستند.

این به آن معنی است که: آن‌ها عدم‌قطعیت همراه با فرآیند تصادفی زیربنایی را به ارث می‌برند.

اگر آزمایش تکرار شود، ما داده‌های متفاوتی خواهیم داشت اما با یک توزیع قطعی مشخص و اجرای یک روش استنباطی ثابت، نتایج استنباطی متفاوتی در بر خواهد داشت اما دوباره با یک توزیع قطعی مشخص.

آماردانان مایل به استفاده از این توزیع برای کمی‌سازی عدم قطعیت همراه با استنباط‌ها هستند:

مدل‌های تصادفی، عدم قطعیت، استنباط آماری

برآورد پارامترها و سایر استنباط‌ها، توابعی از مقادیر داده هستند.

این به آن معنی است که: آن‌ها عدم‌قطعیت همراه با فرآیند تصادفی زیربنایی را به ارث می‌برند.

اگر آزمایش تکرار شود، ما داده‌های متفاوتی خواهیم داشت اما با یک توزیع قطعی مشخص و اجرای یک روش استنباطی ثابت، نتایج استنباطی متفاوتی در بر خواهد داشت اما دوباره با یک توزیع قطعی مشخص.

آماردانان مایل به استفاده از این توزیع برای کمی‌سازی عدم قطعیت همراه با استنباط‌ها هستند:

- خطای معیار، پاسخی است به این سوال که تا چه حد برآورد پارامتر مورد نظر از یک تکرار آزمایش به دیگری، تغییر می‌کند؟
- یک ناحیه اطمینان برای پارامتر، پاسخ به این سوال است که همه مقادیری از پارامتر که این داده‌ها را، با حداقل یک احتمال مشخص، می‌توانند تولید کنند، کدامند؟

مدل‌های تصادفی، عدم قطعیت، توزیع‌های نمونه‌ای

برای داشتن چیزهایی مانند خطای استاندارد و فاصله اطمینان، نیازمند دانستن توزیع برآوردگرهای به دست آمده هستیم.

به توزیع برآوردگرها، توزیع‌های نمونه‌ای گویند.

توزیع‌های نمونه‌ای از توزیع داده‌ها پیروی می‌کنند، زیرا برآوردگرها توابعی از داده‌ها هستند.

مدل‌های تصادفی، عدم قطعیت، توزیع‌های نمونه‌ای

برای داشتن چیزهایی مانند خطای استاندارد و فاصله اطمینان، نیازمند دانستن توزیع برآوردگرهای به دست آمده هستیم.

به توزیع برآوردگرها، توزیع‌های نمونه‌ای گویند.

توزیع‌های نمونه‌ای از توزیع داده‌ها پیروی می‌کنند، زیرا برآوردگرها توابعی از داده‌ها هستند.

از دیدگاه ریاضی، با یک مساله خوش تعریف روبه‌رو هستیم. اما محاسبه داستان دیگری است!!!

مدل‌های تصادفی، عدم قطعیت، توزیع‌های نمونه‌ای

برای داشتن چیزهایی مانند خطای استاندارد و فاصله اطمینان، نیازمند دانستن توزیع برآوردگرهای به دست آمده هستیم.

به توزیع برآوردگرها، توزیع‌های نمونه‌ای گویند.

توزیع‌های نمونه‌ای از توزیع داده‌ها پیروی می‌کنند، زیرا برآوردگرها توابعی از داده‌ها هستند.

از دیدگاه ریاضی، با یک مساله خوش تعریف روبه‌رو هستیم. اما محاسبه داستان دیگری است!!!

معمولا، برآوردگرها توابع پیچیده‌ای از داده‌ها هستند و صحبت در مورد محاسبه شکل‌های بسته برای توزیع آن‌ها، ناامیدکننده است.

در این موارد، دو راهکار کلاسیک آماردانان عبارتند از:

- تمرکز بر روی موارد خاص و ساده
- استفاده از نظریه مجانبی توزیع‌ها

مدل‌های تصادفی، عدم قطعیت، توزیع‌های نمونه‌ای، محاسبات

تا حدود دهه ۶۰ میلادی، علم آمار در توسعه دو راهکار بالا متمرکز شده بود.

اما با انقلابی که دنیای انفورماتیک و کامپیوترها ایجاد کردند، باعث به کار آمدن مدل‌های آماری پیچیده‌تر (و البته واقعی‌تر) و خارج شدن از دنیای کوچک مدل‌های ساده و سرراست شد.

از طرف دیگر، ممکن است نظریه بزرگ‌نمونه برای مدل‌های پیچیده نیز، درمانی داشته باشد، اما همگرایی به توزیع‌های حدی ممکن است به طور غیرقابل قبولی کند باشد.

تا قبل از دهه ۷۰ میلادی، آمار با مساله کمی‌سازی عدم قطعیت استنباط‌ها بدون در نظر گرفتن پذیره‌های غیرمفید و آمار مجانبی، مواجه شد.

همه راه حل‌ها تبدیل شد به محاسبات آماری بیشتر و بیشتر.

یکی از موفق‌ترین راه حل‌ها که توسط بردلی افرون پیشنهاد شد، خودگردان‌سازی، *Bootstrapping*، است.

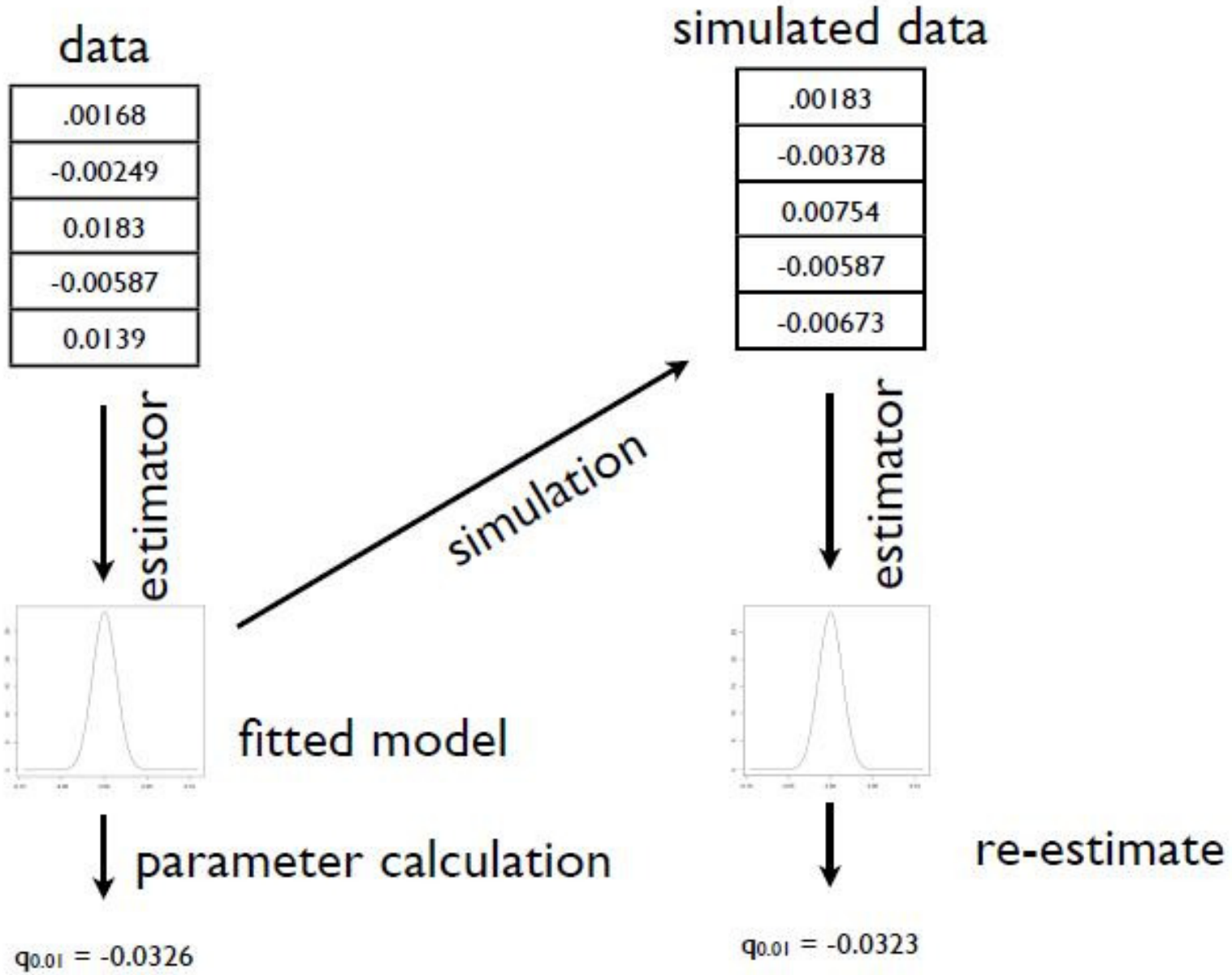
اصل خودگردان‌سازی

کلید بررسی عدم قطعیت در برآوردگرها، توزیع نمونه‌ای آن‌ها می‌باشد.

زیرکی افرون در این بود که گفت می‌توانیم تکرار آزمایش را شبیه‌سازی کنیم:

- مدلی را که حدس می‌زنیم مکانیسم واقعی تولید داده‌ها باشد را به داده‌ها برازش می‌دهیم
- اجرای آن مکانیسم، داده‌های شبیه‌سازی‌شده را تولید می‌کند که دارای توزیع مشابه داده‌های واقعی است
- محاسبه مقدار برآوردگر بر روی داده‌های شبیه‌سازی‌شده، یک تحقق از توزیع نمونه‌ای برآوردگر را نتیجه می‌دهد
- تکرار این فرآیند به تعداد فراوان، توزیع نمونه‌ای برآوردگر را تقریب می‌زند

چون از مدل برای محاسبه عدم قطعیت خودش استفاده می‌شود، افرون آن را خودگردان نامید.


data
.00168
-0.00249
0.0183
-0.00587
0.0139
estimator
simulation
simulated data
.00183
-0.00378
0.00754
-0.00587
-0.00673
estimator
fitted model
parameter calculation
re-estimate
$q_{0.01}$ = -0.0326
$q_{0.01}$ = -0.0323

نمادگذاری

داده‌های اصلی را با X نشان می‌دهیم. دقت کنید در اینجا منظور از X چارچوب داده است نه یک عدد تنها.

پارامتر برآوردشده به وسیله داده‌ها: $\hat{\theta}$

مجموعه داده‌های شبیه‌سازی‌شده از مدل برازش‌شده: $\tilde{X}_1, \ldots, \tilde{X}_B$

برآوردهای خودگردان پارامتر، حاصل از داده‌های شبیه‌سازی‌شده: $\tilde{\theta}_1, \ldots, \tilde{\theta}_B$

علاوه بر این فرض کنید، تابعی مورد نظر، $\theta(F)$، توسط آماره T به صورت $\hat{t} = T(X)$ برآورد شود و به طور مشابه برای داده‌های شبیه‌سازی‌شده:

$$\tilde{t}_1 = T(\tilde{X}_1), \ldots, \tilde{t}_B = T(\tilde{X}_B),$$

$$\tilde{t} = (\tilde{t}_1, \ldots, \tilde{t}_B).$$

آماره T ممکن است تابعی مستقیم از پارامترهای برآوردشده باشد، یا تابعی غیرمستقیم.

در این جا فرض می‌کنیم مدل برای مقداری از θ یک مدل درست است. این مقدار را با θ_0 نشان می‌دهیم. به طور مشابه فرض کنید مقدار درست تابعی، t_0 است.

واریانس و خطای استاندارد

ساده‌ترین معیار برای کمی‌سازی عدم قطعیت، واریانس یا خطای استاندارد است:

$$\begin{aligned} \hat{Var}(\hat{t}) &= Var(\tilde{t}) \\ \hat{se}(\hat{t}) &= sd(\tilde{t}) \end{aligned}$$

منطق این روابط اینست که $\tilde{X}$ شبیه‌سازی‌شده توزیع یکسانی با متغیر واقعی X دارد که داده‌های اصلی، x، از آن استخراج شده‌اند. بنابراین اجرای مشابه روش برآورد بر روی داده‌های شبیه‌سازی‌شده، توزیع نمونه‌ای را نتیجه خواهد داد.

دقت کنید در این جا فرض بر آن است که مدل در نظر گرفته شده درست است و $\hat{\theta}$ خیلی از مقدار واقعی θ. دور نیست.

```
rboot <- function(B, statistic, simulator, ...) {
         tboots <- replicate(B, statistic(simulator(...)))
         return(tboots)
}
bootstrap.se <- function(simulator, statistic, B, ...) {
                tboots <- rboot(B, statistic, simulator, ...)
                se <- sd(tboots)
                return(se)
}
```

تصحیح اریبی

می‌توان از روش خودگردان برای تصحیح اریبی یک برآوردگر اریب، استفاده کرد.

از آن‌جا که توزیع نمونه‌ای $\tilde{t}$ نزدیک به توزیع $\hat{t}$ است، و $\hat{t}$ نیز نزدیک به t_0، پس

$$\mathbb{E}(\hat{t}) - t_0 \approx \mathbb{E}(\tilde{t}) - \hat{t}.$$

سمت چپ تساوی، اریبی هست که به دنبال محاسبه آن هستیم و سمت راست مقداری است که از روی نمونه‌های خودگردان قابل دستیابی است.

دقت کنید که تساوی بالا مادامی معتبر است که توزیع نمونه‌ای $\hat{t} - t_0$ نزدیک به توزیع نمونه‌ای $\tilde{t} - \hat{t}$ باشد. این لزوم، شرط ضعیفتری نسبت به نزدیک بودن توزیع‌های نمونه‌ای $\tilde{t}$ به $\hat{t}$ و $\hat{t}$ به t_0 است.

یک شرط کافی (اما نه لازم) برای برقراری تساوی بالا، آن است که کمیت $\hat{t} - t_0$ محوری یا تقریبا محوری باشد.

```
bootstrap.bias <- function(simulator, statistic, B, t.hat, ...) {
                  tboots <- rboot(B, statistic, simulator, ...)
                  bias <- mean(tboots) - t.hat
                  return(bias)
}
```

فاصله اطمینان

$$Pr(t_{\cdot} \in C) = ۱ - \alpha,$$

در زمان محاسبه فاصله اطمینان، عدم شناخت دقیق توزیع نمونه‌ای به آن معنی است که سطح اطمینان واقعی (نرخ پوشش فاصله) مقدار دلخواه $۱ - \alpha$ نخواهد بود.

هر چه نرخ پوشش تجربی نزدیک‌تر به مقدار اسمی باشد، به معنی تقریب بهتر و در نتیجه فاصله اطمینان دقیق‌تر است.

فرض کنید C_l و C_u به ترتیب کران‌های پایین و بالای فاصله اطمینان باشند. برای فواصل اطمینان با دم‌های برابر داریم:

$$\begin{aligned} \frac{\alpha}{2} &= Pr(C_l \geq t_0) = Pr(C_l - \hat{t} \geq t_0 - \hat{t}) \\ &= Pr(\hat{t} - C_l \leq \hat{t} - t_0) \\ \frac{\alpha}{2} &= Pr(\hat{t} - C_u \geq \hat{t} - t_0) \end{aligned}$$

فاصله اطمینان خودگردانی پایه

روش خودگردانی، توزیع $\tilde{t} - \hat{t}$ را نتیجه می‌دهد که تقریبا با توزیع $\hat{t} - t$، یکی است. با محاسبه این توزیع و $\hat{t}$، می‌توان کران‌های C_l و C_u را به دست آورد:

$$\begin{aligned} C_l &= \hat{t} - \left(Q_{\tilde{t}}(1 - \frac{\alpha}{2}) - \hat{t} \right) \\ C_u &= \hat{t} - \left(Q_{\tilde{t}}(\frac{\alpha}{2}) - \hat{t} \right) \end{aligned}$$

که در آن $Q_{\tilde{t}}$ تابع چندک $\tilde{t}$ است. یعنی چندک‌های توزیع نمونه‌ای نمونه‌های شبیه‌سازی‌شده را به دست می‌دهد.

تمرین: دو رابطه بالا را اثبات کنید.

این فاصله، یک فاصله اطمینان خودگردانی پایه یا فاصله اطمینان مبتنی بر کمیت محوری است. به سادگی قابل محاسبه است و دقت آن هم قابل قبول است.

```
bootstrap.ci.basic <- function(simulator, statistic, B, t.hat,
                               alpha, ...) {
   tboots <- rboot(B,statistic, simulator, ...)
   ci.lower <- 2*t.hat - quantile(tboots,1-alpha/2)
   ci.upper <- 2*t.hat - quantile(tboots,alpha/2)
   return(list(ci.lower=ci.lower,ci.upper=ci.upper))
}
```

فواصل اطمینان خودگردانی استودنت‌شده

فاصله اطمینان خودگردانی پایه، مبتنی بر پذیره تقریبا یکسان بودن توزیع $\tilde{t} - \hat{t}$ با توزیع $\hat{t} - t_0$ است.

اما حتی زمانی که این پذیره نادرست باشد، توزیع

$$\tau = \frac{\hat{t} - t_0}{\hat{se}(\hat{t})}$$

نزدیک به توزیع

$$\tilde{\tau} = \frac{\tilde{t} - \hat{t}}{se(\tilde{t})}$$

خواهد بود. این کمیت‌ها شبیه آماره t در آزمون tـ استودنت هستند و چون آزمون t توسط استودنت کشف شد، به آن‌ها کمیت‌های استودنت‌شده می‌گویند.

فواصل اطمینان خودگردانی استودنت‌شده: ادامه

اگر τ و $\tilde{\tau}$ توزیع یکسانی داشته باشند، می‌توان فاصله اطمینان زیر را به دست آورد:

$$(\hat{t} - \hat{se}(\hat{t})\, Q_{\tilde{\tau}}(۱ - \frac{\alpha}{۲}), \hat{t} - \hat{se}(\hat{t})\, Q_{\tilde{\tau}}(\frac{\alpha}{۲}))$$

این فاصله با فاصله پایه یکی خواهد بود هرگاه $\hat{se}(\hat{t}) = se(\tilde{t})$ و در غیر این صورت متفاوت خواهند بود.

برای محاسبه $se(\tilde{t})$ نیاز به سطح دومی از خودگردان‌سازی داریم. به الگوریتم زیر دقت کنید:

فواصل اطمینان خودگردانی استودنت‌شده: الگوریتم

۱. مدل را با $\hat{\theta}$ برازش بده و $\hat{t}$ را محاسبه کن

۲. برای $i = ۱, \ldots, B_۱$:

الف) $\tilde{X}_i$ را از $\hat{\theta}$ تولید کن

ب) مقادیر $\tilde{\theta}_i$ و $\tilde{t}_i$ را محاسبه کن

ج) برای $j = ۱, \ldots, B_۲$:

۱. $X^{\dagger}_{ij}$ از $\tilde{\theta}_i$ تولید کن

۲. $t^{\dagger}_{ij}$ را محاسبه کن

د) $\tilde{\sigma}_i$ را برابر انحراف استاندارد $t^{\dagger}_{ij}$ها قرار بده

ه) برای تمام jها قرار بده $\tilde{\tau}_{ij} = \frac{t^{\dagger}_{ij} - \tilde{t}_i}{\tilde{\sigma}_i}$

۳. $\hat{se}(\hat{t})$ را برابر انحراف استاندارد $\tilde{t}_i$ها قرار بده

۴. چندک‌های $\alpha/۲$ و $۱ - \alpha/۲$ توزیع $\tilde{\tau}$ را محاسبه کن

۵. همه کمیت‌های مورد نظر را در فاصله اطمینان جایگذاری کن

فواصل اطمینان خودگردانی چندکی

مزیت فواصل استودنت‌شده نسبت به فواصل پایه، دقت بالاترشان است.

عیب آن‌ها نیز زمان‌بر بودن محاسبات است.

نوع دیگری از فواصل اطمینان خودگردانی، فواصل چندکی است که به راحتی به صورت زیر تشکیل می‌شود:

$$(Q_{\tilde{t}}(\alpha/۲), Q_{\tilde{t}}(۱ - \alpha/۲)).$$

این نوع فواصل اطمینان به سادگی قابل محاسبه‌اند، اما دقیق نیستند.

البته همه این فواصل اطمینان مطرح‌شده، صورت‌های مختلفی دارند که توسط افراد مختلف پیشنهاد شده‌اند.

آزمون فرضیه خودگردانی

برای آزمون فرضیه‌ها، دو توزیع نمونه‌ای متفاوت را محاسبه می‌کنیم:

- توزیع آماره آزمون تحت فرضیه صفر که برای محاسبه اندازه آزمون و سطح معنی‌داری است
- توزیع آماره آزمون تحت فرضیه جانشین که برای محاسبه توان آزمون است

هر دو توزیع با روش خودگردان‌سازی قابل محاسبه هستند.

در آزمون فرضیه، آماره مورد نظر، t، آماره آزمون می‌باشد.

```
boot.pvalue <- function(test,simulator,B,testhat, ...) {
   testboot <- rboot(B=B, statistic=test, simulator=simulator, ...)
   p <- (sum(test >= testhat)+1)/(B+1)
   return(p)
}
```

این کد، pـ مقدار خودگردانی را برای یک آزمون محاسبه می‌کند. دقت کنید که $testhat$ مقدار آماره آزمون برای داده‌های اصلی است و $test$ تابعی است که معرف آماره آزمون است.

برای محاسبه توان آزمون به روش مشابه می‌توان عمل کرد. فقط باید نمونه‌های خودگردانی تحت فرضیه جانشین تولید شوند و $testhat$ مقدار بحرانی آزمون باشد نه مقدار مشاهده‌شده آماره آزمون.

مثال: قانون نابرابری ثروت پارتو

توزیع پارتو (توزیع قانون توانی) یک مدل معمول برای داده‌های با دم‌های سنگین است. به این معنی که چگالی احتمال $f(x)$ زمانی که $x \longrightarrow \infty$ خیلی کند به صفر میل می‌کند. یا به عبارت دیگر، این توزیع به شدت به راست چوله است و میانگین آن خیلی بزرگتر از میانه است.

$$f(x) = \frac{\theta - ۱}{x_۰} \left(\frac{x}{x_۰} \right)^{-\theta}$$

که در آن $x_۰$ مقیاس می‌نیمم توزیع است.

تمرین. نشان دهید که $x_۰$ مد توزیع پارتو است.

اگر $x_۰$ معلوم باشد، آن گاه:

$$\hat{\theta} = ۱ + \frac{n}{\sum_{i=۱}^{n} \log \frac{x_i}{x_۰}},$$

که برآوردگری سازگار و کاراست.

مثال پارتو: ادامه

فایل $pareto.R$ شامل تعدادی تابع مرتبط با توزیع پارتو است. یکی از توابع آن $pareto.fit$ است که مدل بالا را به داده‌ها برازش می‌دهد.

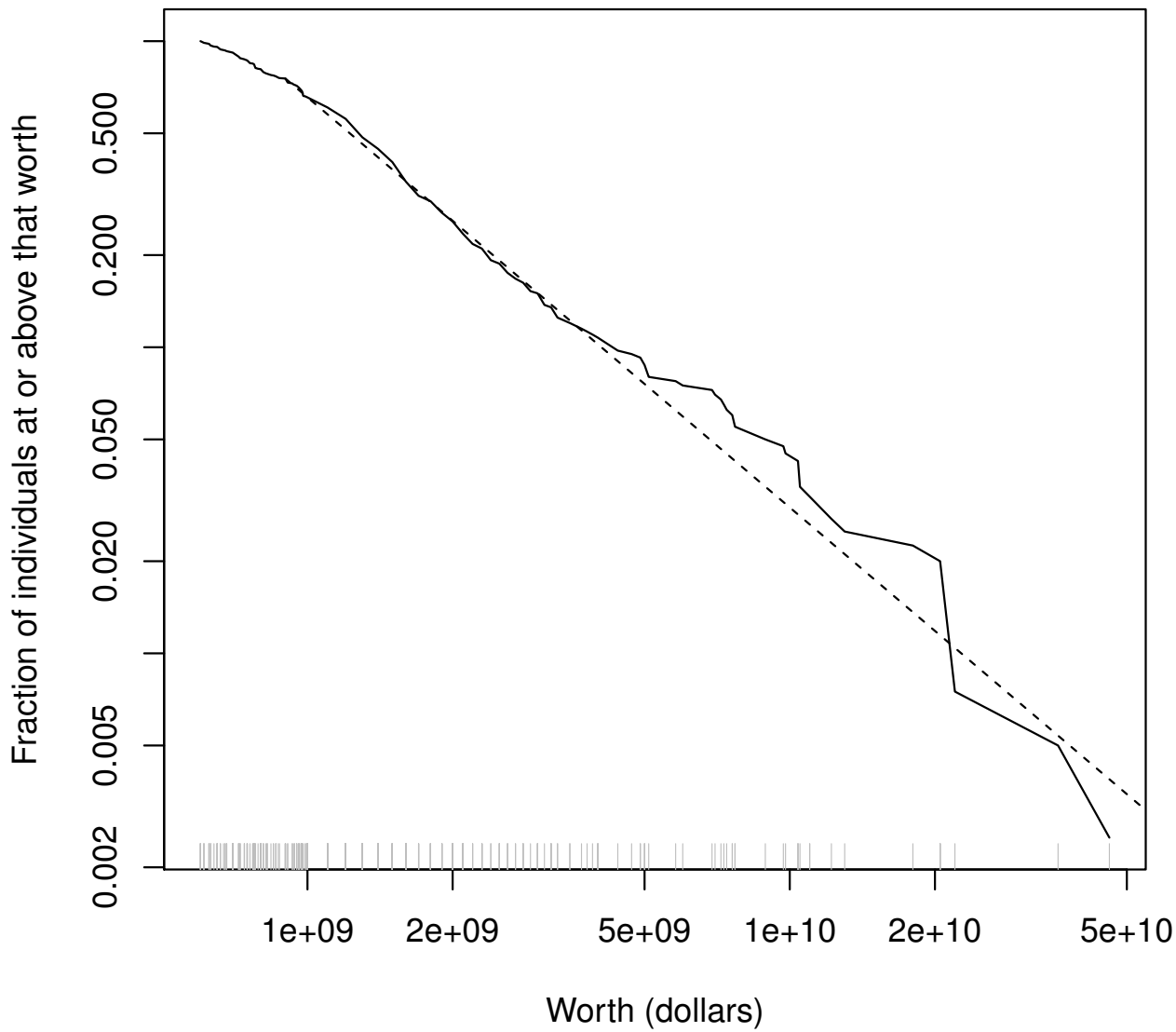
پارتو، وقتی که می‌خواست توزیع داده‌های ثروت را مدل‌بندی کند، به چگالی پارتو رسید.

تقریبا در کشورهای مختلف و زمان‌های متفاوت، دم بالایی توزیع درآمد و ثروت از یک قانون توانی پیروی می‌کند.

پارامتر θ در این توزیع با مقدار پولی که کم و بیش در دست ثروتمندترین مردم کشور هست، تغییر می‌کند.

شکل صفحه بعد، توزیع ثروت را برای ۴۰۰ پولدار اول آمریکا در سال ۲۰۰۳ نشان می‌دهد. با در نظر گرفتن $x_. = ۹ \times ۱۰^۸$، تعداد افرادی که در دم قرار می‌گیرند، ۳۰۲ نفر است و $\hat{\theta} = ۲/۳۴$:

```
source("pareto.R")
wealth <- scan("wealth.dat")
wealth.pareto <- pareto.fit(wealth,threshold=9e8)
> signif(wealth.pareto$exponent,3)
[1] 2.34
```


Fraction of individuals at or above that worth
0.002
0.005
0.020
0.050
0.200
0.500
1e+09
2e+09
5e+09
1e+10
2e+10
5e+10
Worth (dollars)

مثال پارتو: ادامه

چه میزان عدم قطعیت در این برآورد θ وجود دارد؟

با روش خودگردانی می‌توان به این سوال پاسخ داد:

```
rboot.pareto <- function(B,exponent,x0,n) {
   replicate(B,pareto.fit(rpareto(n,x0,exponent),x0)$exponent)
}
pareto.se <- function(B,exponent,x0,n) {
   return(sd(rboot.pareto(B,exponent,x0,n)))
}
pareto.bias <- function(B,exponent,x0,n) {
   return(mean(rboot.pareto(B,exponent,x0,n)) - exponent)
}
```

با $\hat{\theta} = ۲/۳۴$، $x_۰ = ۹ \times ۱۰^۸$، $n = ۳۰۲$ و $B = ۱۰^۴$، خطای معیار (عدم قطعیت برآورد) برابر ۰/۰۷۷ به دست می‌آید که با نتایج مجانبی هم هم‌خوانی دارد.

به طور مجانبی نیز (با توجه به سازگاری برآوردگر) اریبی به سمت صفر میل می‌کند. میزان اریبی خودگردان نیز برابر ۴×۱۰^{-۳} به دست آمده است که قابل صرفنظر است.

مثال پارتو: ادامه

می‌توان فاصله اطمینان خودگردانی پایه را نیز محاسبه کرد:

```
pareto.ci <- function(B,exponent,x0,n,alpha) {
   tboot <- rboot.pareto(B,exponent,x0,n)
   ci.lower <- 2*exponent - quantile(tboot,1-alpha/2)
   ci.upper <- 2*exponent - quantile(tboot,alpha/2)
   return(list(ci.lower=ci.lower, ci.upper=ci.upper))
}
```

با استفاده از همان مشخصات قبلی، فاصله ٪۹۵ پایه به صورت $(۲/۱۸, ۲/۴۸)$ به دست می‌آید.

روش خودگردان‌سازی ناپارامتری

خودگردان‌سازی، توزیع نمونه‌ای را همراه با سه منبع خطای برآورد، تقریب می‌زند:

۱. خطای شبیه‌سازی: استفاده از تعداد تکرار فراوان اما متناهی برای دستیابی به توزیع نمونه‌ای کامل. تعداد کافی تکرار و طراحی مناسب شبیه‌سازی، می‌تواند این خطا را به اندازه دلخواه کوچک کند.

۲. خطای آماری: توزیع نمونه‌ای پارامترهای برآوردشده خودگردانی تحت مدل برازش‌شده، دقیقا با توزیع نمونه‌ای پارامترهای برآوردشده تحت مدل واقعی مکانیسم تولید داده‌ها یکی نیست. توزیع نمونه‌ای با تغییر پارامترها، تغییر می‌کند و برآورد اولیه ما کاملا دقیق نیست. اما اغلب مشخص شده است که توزیع برآوردها حول مقدار واقعی پایاتر از توزیع خود برآوردهاست. بنابراین کم کردن برآورد اولیه از مقادیر خودگردان، باعث کاهش خطای آماری می‌شود.

۳. خطای مشخص‌سازی: داده‌ها از مدل‌هایی که ما در نظر می‌گیریم دقیقا پیروی نمی‌کنند. بنابراین شبیه‌سازی مدل هرگز با توزیع نمونه‌ای واقعی یکی نمی‌شود.

خطاها

خطای آماری:

$$F_n \longrightarrow \tilde{X} \longrightarrow F_n^*,$$

$$F_n \longrightarrow \tilde{t} \longrightarrow F_n^*$$

خطای مشخص‌سازی:

$$F \longrightarrow X \longrightarrow F_n$$

$$F \longrightarrow t \longrightarrow F_n$$

برای درک خطای آماری، به مثال ۱ ۰ ۷ کتاب توجه کنید.

روش خودگردان‌سازی ناپارامتری: ادامه

افرون، یک ایده معرکه دومی هم داشت، که مربوط به خطای مشخص‌سازی می‌شد.

وی پیشنهاد کرد برای فرار از این خطا، به جای شبیه‌سازی از مدل، از خود داده‌ها (یا به عبارتی توزیع تجربی داده‌ها) بازنمونه‌گیری کنیم.

در واقع خود داده‌ها (یا توزیع تجربی داده‌ها) آماره بسنده برای پارامتر است.

از دیدگاه دیگر، توزیع تجربی داده‌ها کم‌تعصب‌ترین برآورد ممکن از توزیع واقعی زیربنایی است. هر چیز دیگری اریبی و پیش‌داوری وارد می‌کند. ممکن است مدل انتخاب‌شده دقیق باشد، اما احتمال نامناسب بودن آن نیز کم نیست.

بسیاری از کمیت‌ها را می‌توان به طور مستقیم، و بدون وساطت یک مدل پارامتری، از روی توزیع تجربی برآورد کرد.

خودگردان‌سازی ناپارامتری افرون، مجموعه داده اصلی را به عنوان یک جامعه متناهی در نظر می‌گیرد و یک نمونه شبیه‌سازی‌شده جدید، که در آن هر مشاهده با احتمال برابر انتخاب می‌شود، از آن استخراج می‌کند. سپس برآورد را با نمونه جدید تکرار می‌کند.

در واقع، معمولا وقتی آماردانان اسم خودگردان را می‌آورند، منظورشان این نوع خودگردان است.

الگوریتم خودگردان‌سازی ناپارامتری

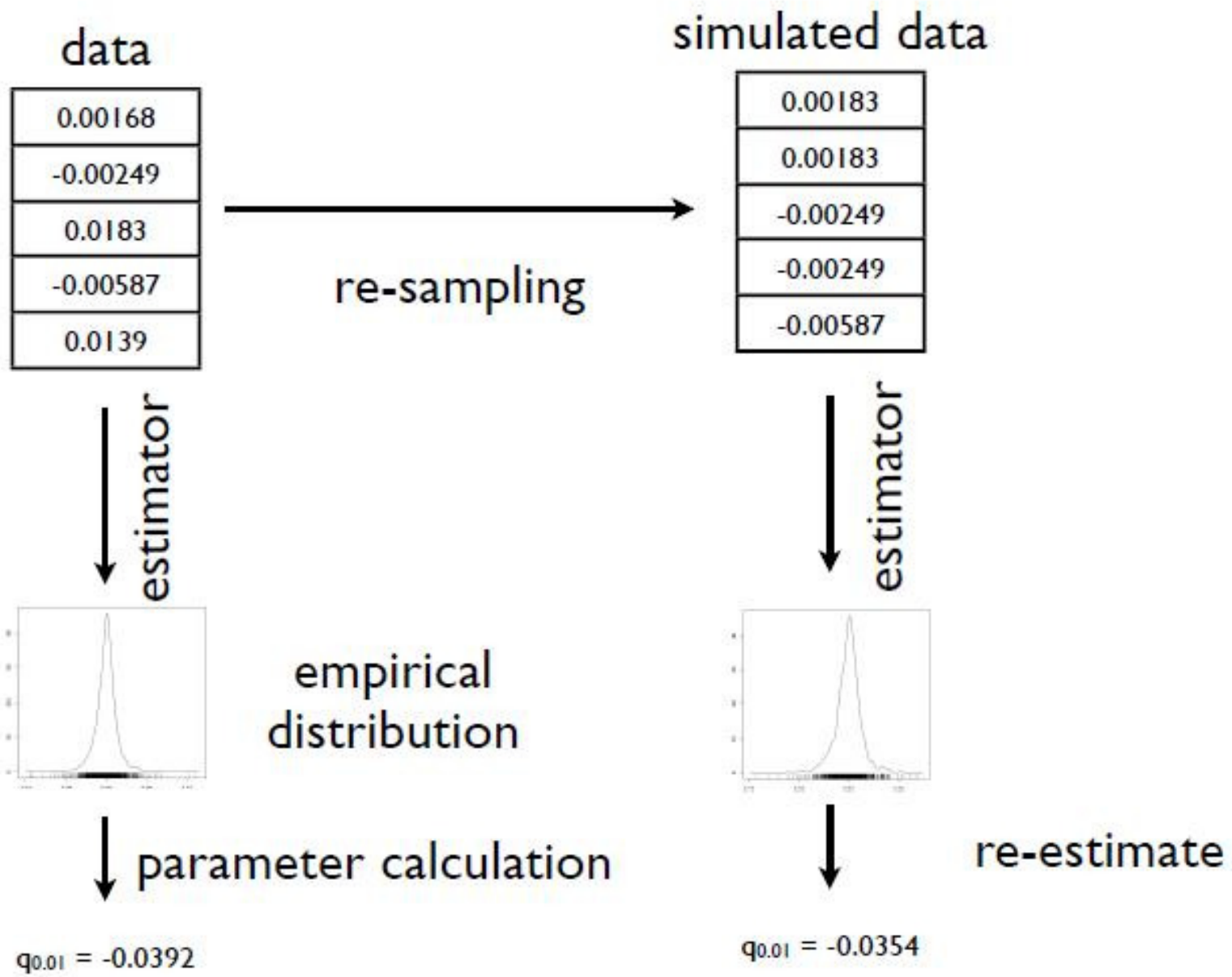
توزیع نمونه‌ای $\hat{t}$ را می‌توان به کمک روش خودگردان‌سازی به صورت زیر تقریب زد:

۱. برای هر تکرار خودگردان $b = ۱, \ldots, B$:

الف) نمونه $\tilde{x}^{(b)} = (\tilde{x}_۱, \ldots, \tilde{x}_n)$ را با نمونه‌گیری با جایگذاری از نمونه مشاهده شده $x_۱, \ldots, x_n$ تولید کن

ب) مقادیر $\tilde{\theta}^{(b)}$ و $\tilde{t}^{(b)}$ را محاسبه کن

۲. برآورد توزیع نمونه‌ای خودگردان، F_n^*، توزیع تجربی $\tilde{t}^{(B)}, \ldots, \tilde{t}^{(۱)}$ می‌باشد


data
0.00168
-0.00249
0.0183
-0.00587
0.0139
re-sampling
simulated data
0.00183
0.00183
-0.00249
-0.00249
-0.00587
estimator
estimator
empirical
distribution
parameter calculation
re-estimate
$q_{0.01}$ = -0.0392
$q_{0.01}$ = -0.0354

روش خودگردان‌سازی ناپارامتری: ادامه

هر آنچه که با خودگردان‌سازی پارامتری مطرح شد، با خودگردان‌سازی ناپارامتری نیز قابل طرح و انجام است.

تنها تفاوت در توزیع داده‌های شبیه‌سازی‌شده خودگردانی است که همان توزیع تجربی داده‌های اصلی است.

```
resample <- function(x) { sample(x,size=length(x),replace=TRUE) }
resamp.pareto <- function(B,data,x0) {
   replicate(B,pareto.fit(resample(data),threshold=x0)$exponent)
}
> sd(resamp.pareto(B,wealth,x0))
[1] 0.07762305
> mean(resamp.pareto(B,wealth,x0))-exponent
[1] 0.003534574
resamp.pareto.CI <- function(B,data,alpha,x0) {
   thetahat <- pareto.fit(data,threshold=x0)$exponent
   thetaboot <- resamp.pareto(B,data,x0)
   ci.lower <- thetahat - (quantile(thetaboot,1-alpha/2) - thetahat)
   ci.upper <- thetahat - (quantile(thetaboot,alpha/2) - thetahat)
   return(list(ci.lower=ci.lower,ci.upper=ci.upper))
}
> resamp.pareto.CI(B,wealth,0.05,x0)
$ci.lower
   97.5%
2.173528

$ci.upper
    2.5%
2.476391
```

خودگردان‌سازی پارامتری در مقابل ناپارامتری

اگر یک مدل مشخص (پارامتری) مناسب داشته باشیم، شبیه‌سازی از آن مدل نتایج دقیق‌تری را، با حجم نمونه ثابت n، نسبت به بازنمونه‌گیری از توزیع تجربی داده‌ها (ناپارامتری) به دست خواهد داد.

در واقع، در این حالت، برآورد پارامتری توزیع سریع‌تر از توزیع تجربی، به توزیع واقعی همگرا می‌شود.

در مقابل اگر مدل پارامتری به اشتباه مشخص شده باشد، به سرعت به یک توزیع نادرست همگرا می‌شود. در چنین مواردی استفاده از روش ناپارامتری، نتایج خیلی بهتری به دست خواهد داد.

معمولا، چون در اغلب کاربردها نسبت به پذیره‌های مدل‌های پارامتری تردید زیادی وجود دارد، ترجیح داده می‌شود از روش‌های بازنمونه‌گیری استفاده شود. مگر آن که بتوان قانع شد که یک مدل پارامتری، تقریب خیلی خوبی از واقعیت پدیده تصادفی خواهد بود.

مثال: مجموعه داده آزمون ورودی دانشکده حقوق

در بسته $bootstrap$ در R، که مربوط به مباحث و مثال‌های کتاب افرون و تیبشیرانی است، مجموعه داده‌ای به نام داده‌های دانشکده حقوق وجود دارد که شامل معدل نمره آزمون‌های ورودی ۸۲ دانشکده حقوق، $LSAT$، و معدل نمرات دبیرستان شرکت‌کنندگان در آزمون، GPA، است.

این داده‌ها در این بسته با نام $law۸۲$ قابل دسترسی است. یک نمونه ۱۵ تایی از این داده‌ها در صفحه ۱۸۵ کتاب آورده شده است.

هدف برآورد ضریب وابستگی بین این دو متغیر و محاسبه برآورد خودگردانی خطای استاندارد این برآورد می‌باشد.

در این مثال تابعی t، ضریب همبستگی بین دو متغیر $LSAT$ و GPA می‌باشد.

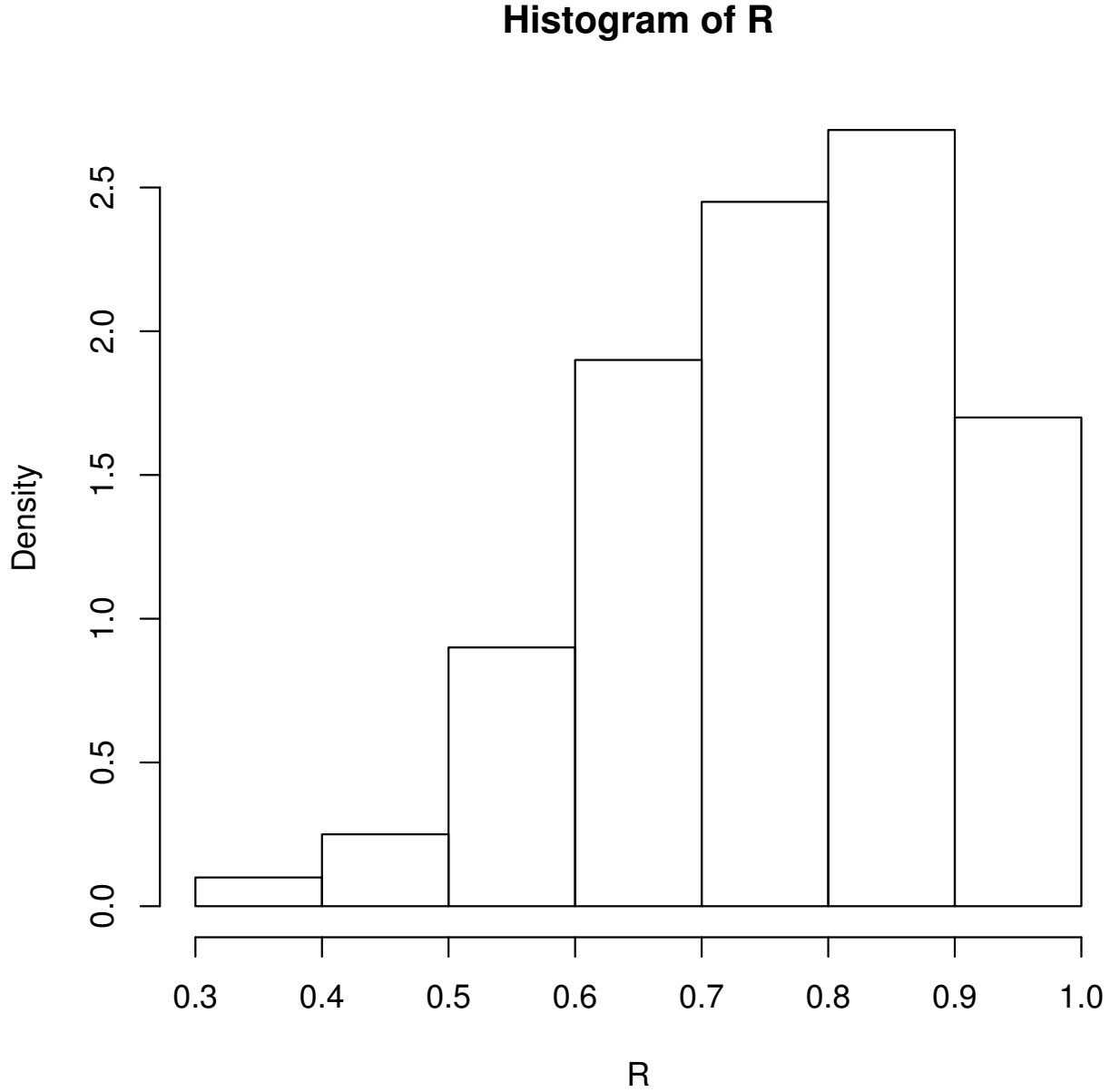
```
library(bootstrap)  # for the law data
> print(cor(law$LSAT, law$GPA)) # for a sample with size 15
[1] 0.7763745
> print(cor(law82$LSAT, law82$GPA))
[1] 0.7599979
```

ادامه مثال: مجموعه داده آزمون ورودی دانشکده حقوق

```
B <- 200  # number of replicates
n <- nrow(law) # sample size
R <- numeric(B) # storage for replicates
# bootstrap estimate of standard error of R
for (b in 1:B) {
    # randomly select the indices
    i <- sample(1:n, size = n, replace = TRUE)
    LSAT <- law$LSAT[i]       #i is a vector of indices
    GPA <- law$GPA[i]
    R[b] <- cor(LSAT, GPA)
}
# output
> print(se.R <- sd(R))
[1] 0.1348793
```

برآورد خطای استاندارد بر اساس نظریه نرمال برابر ۰/۱۱۵ است. (چرا؟)

تمرین: با تابع *boot* در بسته *boot* می‌توان محاسبات بالا را انجام داد. برای مثال آزمون ورودی دانشکده حقوق، محاسبات را تکرار کنید و با نتایج بالا مقایسه کنید.


Histogram of R
Density
0.0
0.5
1.0
1.5
2.0
2.5
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
R

ادامه مثال: مجموعه داده آزمون ورودی دانشکده حقوق

برای برآورد اریبی نیز می‌توان مشابه مثال‌های قبلی عمل کرد:

```
# sample estimate for n=15
theta.hat <- cor(law$LSAT, law$GPA)
# bootstrap estimate of bias
B <- 2000 # larger for estimating bias
n <- nrow(law)
theta.b <- numeric(B)
for (b in 1:B) {
     i <- sample(1:n, size = n, replace = TRUE)
     LSAT <- law$LSAT[i]
     GPA <- law$GPA[i]
     theta.b[b] <- cor(LSAT, GPA)
}
bias <- mean(theta.b - theta.hat)
> bias
[1] -0.002769211
```

برآورد اریبی برای یک برآوردگر نسبت

داده‌های $patch$ نیز در بسته $bootstrap$ قرار دارد که توضیح کامل آن در کتاب افرون و تیبشیرانی (۱۹۹۳) هست.

این داده‌ها شامل اندازه‌های یک هرمون خاص در گردش خون ۸ آزمودنی بعد از پانسمان پزشکی است. پارامتر مورد نظر عبارتست از:

$$\theta = \frac{E(new) - E(old)}{E(old) - E(placebo)}$$

که در آن $E(old)$ و $E(new)$ پانسمان‌های قدیمی و جدید هستند. برای این داده‌ها، اگر $|t| \leq ۰/۲۰$، نشان‌دهنده برابری عملکرد پزشکی پانسمان‌های قدیم و جدید است.

تابعی مورد نظر آماره $t = \frac{\bar{Y}}{\bar{Z}}$ است. هدف برآورد خودگردانی اریبی آماره نسبت برابری عملکرد است.

```
data(patch, package = "bootstrap")
> patch
  subject placebo oldpatch newpatch     z     y
1       1    9243    17649    16449  8406 -1200
2       2    9671    12013    14614  2342  2601
3       3   11792    19979    17274  8187 -2705
4       4   13357    21816    23798  8459  1982
5       5    9055    13850    12560  4795 -1290
6       6    6290     9806    10157  3516   351
7       7   12412    17208    16570  4796  -638
8       8   18806    29044    26325 10238 -2719
```

برآورد اریبی برای یک برآوردگر نسبت: ادامه

```
n <- nrow(patch) # in bootstrap package
B <- 2000
theta.b <- numeric(B)
theta.hat <- mean(patch$y) / mean(patch$z)
# bootstrap
for (b in 1:B) {
     i <- sample(1:n, size = n, replace = TRUE)
     y <- patch$y[i]
     z <- patch$z[i]
     theta.b[b] <- mean(y) / mean(z)
}
bias <- mean(theta.b) - theta.hat
se <- sd(theta.b)
> print(list(est=theta.hat, bias = bias,se = se, cv = bias/se))
$est
[1] -0.0713061

$bias
[1] 0.004647646

$se
[1] 0.09888232

$cv
[1] 0.04700179
```

معمولا اگر $\frac{|bias|}{se} \leq$ ۰/۲۵، نیازی به تصحیح اریبی نیست. در این مثال این نسبت کمتر از ۰/۰۵ است.

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه یازدهم: جک‌نایف و اعتبارسنجی متقابل

حسین باغیشنی

دانشگاه شاهرود

۱۴ آذر ۱۳۹۳

جک‌نایف: یک روش بازنمونه‌گیری دیگر

روش جک‌نایف مانند خودگردان‌سازی، یک روش بازنمونه‌گیری است. در واقع به صورت ناپارامتری عمل می‌کند.

این روش خیلی زودتر از خودگردان‌سازی و بعضی دیگر از روش‌های بازنمونه‌گیری معرفی شد.

در واقع این روش در سال ۱۹۴۹ توسط کوییلا برای برآورد اریبی و سپس توسط توکی (۱۹۵۸) برای برآورد خطای استاندارد، پیشنهاد شد.

روش جک‌نایف شبیه به نوعی اعتبارسنجی متقابل LOO ($Leave\ One\ Out$) عمل می‌کند که در مورد آن صحبت خواهیم کرد.

فرض کنید $x = (x_1, \ldots, x_n)$ نمونه مشاهده شده باشد. iامین نمونه جک‌نایف را با $x_{(i)}$ نشان می‌دهیم، به طوری که همان نمونه x است با این تفاوت که iامین مشاهده نمونه حذف شده باشد. یعنی:

$$x_{(i)} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

برآورد اریبی به روش جک‌نایف

فرض کنید تابعی مورد نظر (برای استنباط) $\theta = t(F)$ باشد.

اگر $\hat{\theta} = T_n(x)$، آنگاه برآورد iامین جک‌نایف، برای $i = ۱, \ldots, n$، عبارتست از $\hat{\theta}_{(i)} = T_{n-۱}(x_{(i)})$.

برآوردگر $\hat{\theta}$ همان برآوردگر جایگذاری‌شده بر اساس توزیع تجربی، یعنی $\hat{\theta} = t(F_n)$، است.

تعریف: یک برآوردگر جایگذاری مانند $\hat{\theta}$ را هموار گویند، هرگاه تغییرات کوچک در داده‌ها، تغییرات ناچیزی در $\hat{\theta}$ ایجاد کند.

اگر $\hat{\theta}$ یک برآوردگر هموار باشد، تعریف می‌کنیم $\hat{\theta}_{(i)} = t(F_{n-۱}(x_{(i)}))$ و برآورد اریبی جک‌نایف عبارتست از:

$$\hat{b}_{jack} = (n-۱)(\bar{\theta}_{(\cdot)} - \hat{\theta}),$$

که در آن

$$\bar{\theta}_{(\cdot)} = \frac{۱}{n}\sum_{i=۱}^{n} \hat{\theta}_{(i)}.$$

برآورد اریبی

اگر $\hat{\theta}$ ناریب باشد، آن‌گاه

$$\mathbb{E}(\bar{\theta}_{(\cdot)}) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(\hat{\theta}_{(i)}) = \theta.$$

اما معمولا برای برآوردگرها داریم:

$$\mathbb{E}(\hat{\theta}) = \theta + \frac{a}{n} + \frac{b}{n^2} + O(n^{-3}).$$

در نتیجه

$$\mathbb{E}(\bar{\theta}_{(\cdot)} - \hat{\theta}) = \frac{a}{n(n-1)} + O(n^{-3}).$$

بنابراین برآوردگر جک‌نایف با اریبی تصحیح‌شده، $\hat{\theta}_{jack} = \hat{\theta} - b_{jack}$، یک برآوردگر ناریب برای θ، تا مرتبه ۲، است (چرا؟)

برآورد اریبی: واریانس نمونه

برای درک بهتر روابط قبلی، فرض کنید تابعی θ واریانس جامعه باشد.

بنابراین برآوردگر جایگذاری واریانس بر حسب نمونه‌ای به حجم n عبارتست از $\hat{\theta} = ۱/n \sum_{i=۱}^{n} (x_i - \bar{x})^۲$.

این برآوردگر اریب است:

$$b(\hat{\theta}) = \mathbb{E}(\hat{\theta} - \sigma^۲) = \frac{n-۱}{n}\sigma^۲ - \sigma^۲ = -\frac{\sigma^۲}{n}.$$

در نتیجه اریبی در هر نمونه جک‌نایف $-\frac{\sigma^۲}{n-۱}$ است. پس برای هر $i = ۱, \ldots, n$ داریم:

$$\begin{aligned}\mathbb{E}(\hat{\theta}_{(i)} - \hat{\theta}) &= \mathbb{E}(\hat{\theta}_{(i)} - \theta) - \mathbb{E}(\hat{\theta} - \theta) = b(\hat{\theta}_{(i)}) - b(\hat{\theta}) \\ &= -\frac{\sigma^2}{n-1} - (-\frac{\sigma^2}{n}) = -\frac{\sigma^2}{n(n-1)} = \frac{b(\hat{\theta})}{n-1}.\end{aligned}$$

در این‌جا برآورد با اریبی تصحیح‌شده همان برآوردگر ناریب واریانس است:
$\hat{\theta}_{jack} = S^۲ = \frac{۱}{n-۱} \sum_{i=۱}^{n} (x_i - \bar{x})^۲$.

مثال: داده‌های نوع پانسمان

در مبحث روش خودگردان‌سازی، مثالی برای داده‌های پانسمان بهداشتی آورده شد. برای آن مثال، می‌خواهیم با روش جک‌نایف، اریبی را برآورد کنیم.

```
data(patch, package = "bootstrap")
n <- nrow(patch)
y <- patch$y
z <- patch$z
theta.hat <- mean(y) / mean(z)
> print (theta.hat)
[1] -0.0713061
# compute the jackknife replicates, leave-one-out estimates
theta.jack <- numeric(n)
for (i in 1:n){
  theta.jack[i] <- mean(y[-i]) / mean(z[-i])
}
  bias <- (n - 1) * (mean(theta.jack) - theta.hat)
> print(bias) # jackknife estimate of bias
[1] 0.008002488
```

شبه‌مقادیر جک‌نایف

راه دیگری برای معرفی برآوردگر جک‌نایف، بر حسب شبه‌مقادیر جک‌نایف است:

$$\tilde{\theta}_i = n\hat{\theta} - (n-1)\hat{\theta}_{(i)}, \quad i = 1, \ldots, n.$$

با معرفی این شبه‌مقادیر، میانگین آن‌ها همان برآوردگر با اریبی تصحیح‌شده می‌باشد:

$$\frac{1}{n}\sum_{i=1}^{n}\tilde{\theta}_i = n\hat{\theta} - (n-1)\bar{\theta}_{(.)} = \hat{\theta} - b_{jack}.$$

ایده پشتیبان شبه‌مقادیر، این است که به ما اجازه می‌دهند برآوردگر با اریبی تصحیح‌شده جک‌نایف را به شکل ساده میانگین n مقدار مستقل نشان دهیم.

ویژگی‌های شبه‌مقادیر

- شبه‌مقادیر $\{\tilde{\theta}_i\}$ به طور کلی مستقل نیستند. البته برای حالت خاصی که آماره خطی است، یعنی $\hat{\theta} = n^{-1} \sum_i a(x_i)$، داریم $\tilde{\theta}_i = a(x_i)$. به عنوان مثال برای میانگین $\tilde{\theta}_i = x_i$.
- بنابراین یک ایده منطقی، در نظر گرفتن $\tilde{\theta}_i$ها به عنوان تقریب‌های خطی مشاهدات iid و میانگین گرفتن از آن‌ها به عنوان برآوردگر با اریبی تصحیح‌شده جک‌نایف $\hat{\theta}_{jack}$ است.
- با قرار دادن $\tilde{S}^2$ به عنوان واریانس نمونه شبه‌مقادیر، برآوردگری برای واریانس $\hat{\theta}$، $\sigma^2(\hat{\theta})$، به صورت زیر خواهد بود:

$$v_{jack} = \frac{\tilde{S}^2}{n}.$$

برآورد جک‌نایف خطای استاندارد

برای برآورد خطای استاندارد جک‌نایف علاوه بر فرمول مبتنی بر شبه‌مقادیر، می‌توان از عبارت زیر هم، زمانی که برآوردگر $\hat{\theta}$ هموار است، استفاده کرد:

$$\hat{se}_{jack} = \left(\frac{n-1}{n} \sum_{i=1}^{n} (\hat{\theta}_{(i)} - \bar{\theta}_{(\cdot)})^2 \right)^{1/2}.$$

تمرین: برای محاسبه خطای استاندارد جک‌نایف، چرا از ضریب $\frac{n-1}{n}$ در فرمول بالا استفاده شده است؟

برای مثال، از برآوردهای جک‌نایف در مثال قبلی برای داده‌های پانسمان بهداشتی بیماران، استفاده می‌کنیم:

```
se <- sqrt((n-1) *
      mean((theta.jack - mean(theta.jack))^2))
> print(se)
[1] 0.1055278
```

فواصل اطمینان جک‌نایف

شبه‌مقادیر جک‌نایف کمک می‌کند تا فواصل اطمینانی به شکل زیر بتوان ساخت:

$$\hat{\theta}_{jack} \mp t_{1-\frac{\alpha}{2};n-1}\, se_{jack}$$

تمرین: یک تابع کلی در R به اسم $jackknife$ بنویسید که روش جک‌نایف را اجرا کند. تابع باید دارای دو ورودی باشد: x که بیانگر داده‌هاست و $theta$ که بیانگر یک تابعی است به طوری که وقتی روی x عمل می‌کند، برآوردهایی را تولید کند. این تابع باید لیستی شامل مولفه‌های زیر را برگرداند:

- $bias$: برآورد جک‌نایف اریبی
- se: برآورد جک‌نایف خطای استاندارد
- $values$: برآوردهای جک‌نایف $\{\hat{\theta}_{(i)}\}$
- $pseudo.values$: شبه‌مقادیر جک‌نایف $\{\tilde{\theta}_i\}$
- $lower.ci$: کران پایین فاصله اطمینان جک‌نایف
- $upper.ci$: کران بالای فاصله اطمینان جک‌نایف

آیا v_{jack} برآوردگر خوبی است؟

واضح است که v_{jack} به عنوان برآوردی از $\sigma^۲(\bar{x})$، خوب است زیرا معادل است با برآوردگر (نااریب) معمول آن.

به‌طور مشابه (با منطقی مشابه)، این برآوردگر برای هر آماره خطی نیز مناسب و خوب است.

حتی اگر g تابعی به‌طور پیوسته مشتق‌پذیر در میانگین جامعه، μ، باشد، نشان داده شده است که v_{jack} برآوردگری سازگار برای واریانس $g(\bar{x})$ است:

$$\frac{v_{jack}}{\sigma^۲(g(\bar{x}))} \xrightarrow{P} ۱$$

اما مواردی وجود دارند که v_{jack} برآوردگر مناسبی برای واریانس یک برآوردگر نیست.

به‌ویژه نشان داده شده است، زمانی که برآوردگر تابعی هموار از داده‌ها نیست، v_{jack} ضعیف عمل می‌کند.

یک مثال ساده از یک برآوردگر ناهموار، میانه است.

اجرای جک‌نایف بر روی میانه

فرض کنید نمونه $\{۱, ۲, \ldots, ۹, ۱۰\}$ مشاهده شده باشد.

برآوردهای جک‌نایف چه طور خواهند بود؟

برآوردهای جک‌نایف عبارتند از ۵ تا ۵ و ۵ تا ۶

برای هر مجموعه داده با n زوج، همیشه دو مقدار یکتا برای $\hat{\theta}_{(i)}$، هر کدام با $n/۲$ تکرار، مشاهده خواهد شد.

بنابراین به نظر نمی‌رسد استفاده از این برآوردهای جک‌نایف ایده خوبی برای برآورد واریانس میانه باشد و البته هم نیست.

نشان داده شده است v_{jack} برآوردگری ناسازگار برای $\sigma^۲(\hat{\theta})$ است:

$$\frac{v_{jack}}{\sigma^۲(\hat{\theta})} \xrightarrow{d} \left(\frac{\chi^۲_۲}{۲}\right)^۲.$$

مثال: خطای استاندارد میانه

در این مثال، خطای استاندارد میانه یک نمونه ۱۰ تایی از اعداد صحیح $\{۱, \ldots, ۱۰۰\}$ محاسبه می‌شود:

```
set.seed(123) # for the specific example given
n <- 10
x <- sample(1:100, size = n)
# jackknife estimate of se
M <- numeric(n)
for (i in 1:n) { # leave one out
     y <- x[-i]
     M[i] <- median(y)
}
Mbar <- mean(M)
> print(sqrt((n-1)/n * sum((M - Mbar)^2)))
[1] 1.5
# bootstrap estimate of se
Mb <- replicate(1000, expr = {
      y <- sample(x, size = n, replace = TRUE)
      median(y) })
> print(sd(Mb))
[1] 13.69387
> print(x)
 [1] 29 79 41 86 91  5 50 83 51 42
> print(M)
 [1] 51 50 51 50 50 51 51 50 50 51
> print(Mb)
```

همانطور که ملاحظه می‌شود برآورد خودگردان و جک‌نایف خیلی با هم اختلاف دارند. در واقع روش جک‌نایف به دلیل ناهموار بودن میانه، موفق نیست.

جک‌نایف d ـ حذفی

یک نسخه دیگر از جک‌نایف که پیشنهاد شده است، جک‌نایف d ـ حذفی $(delete - d\ jackknife)$ نامیده می‌شود.

همانطور که از نام آن پیداست، به جای هر بار حذف یک مشاهده برای محاسبه مجموعه $\{\hat{\theta}_{(i)}\}$، هر بار d مشاهده حذف می‌شود.

این رهیافت دارای مزایایی است:
از جمله نشان داده شده است اگر d به طور مناسب انتخاب شود (صفحه ۱۹۴ کتاب را ببینید)، آنگاه برآوردگر جک‌نایف d ـ حذفی واریانس برای میانه سازگار خواهد بود.

البته عیبی که این روش دارد، محاسبات خیلی زمان‌بر آن است. زیرا به جای محاسبه n برآورد در جک‌نایف معمولی، تعداد $\binom{n}{d}$ برآورد d ـ حذفی نیاز دارد که خیلی بزرگتر از n می‌تواند باشد.

اعتبارسنجی متقابل، چیزی که هر آماردانی باید بداند

اعتبارسنجی متقابل، *Cross Validation*، (CV) روشی است برای ارزیابی توان (پیشگویی) یک مدل آماری

هر آماردانی می‌داند که معیارهای (آماره‌های) برازش مدل، راهنماهای خوبی برای پاسخ به این سوال که یک مدل با چه دقتی پیشگویی خواهد کرد، نیستند.

به عنوان مثال، R^2 بزرگ برای یک مدل لزوماً به معنی خوب بودن آن مدل نیست:

- به سادگی می‌توان با افزودن پارامترهای بیشتر به یک مدل (به عبارتی بیش‌برازش مدل)، مقدار R^2 را نیز افزایش داد.
- در یک رگرسیون چندجمله‌ای، با افزودن جملات با مرتبه بالاتر می‌توان مدلی با برازش بهتر به داده‌ها به دست آورد.

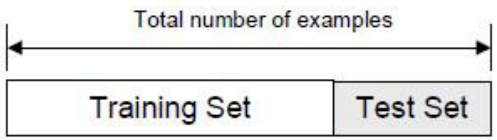
اما پیشگویی‌های مدل بر روی داده‌های جدید در یک مدل بیش‌برازش‌شده بدتر خواهد بود.

بیش‌برازش = عملکرد ضعیف در پیشگویی

مجموعه‌های آموزشی و آزمون

یک روش برای اندازه‌گیری قدرت پیشگویی یک مدل، آزمودن آن بر روی مجموعه‌ای از داده‌هاست که در برازش مدل مورد استفاده قرار نگرفته باشند.

متخصصان یادگیری ماشینی و داده‌کاوها به چنین مجموعه‌ای، مجموعه آزمون و به مجموعه داده‌ای که برای برازش مدل استفاده شده است، مجموعه آموزشی می‌گویند.



به عنوان مثال، دقت پیشگویی یک مدل را می‌توان با MSE آن که بر روی مجموعه آزمون محاسبه شده است، اندازه‌گیری کرد.

به‌طور کلی، از آن‌جا که داده‌های آزمون در برازش مدل استفاده نشده‌اند، این MSE نسبت به MSE محاسبه‌شده از مجموعه آموزشی، بزرگتر خواهد بود.

مجموعه‌های آموزشی و آزمون: عیب‌ها

استفاده از این رهیافت دو عیب اساسی دارد:

۱. در مواردی که حجم داده‌ها کم است، ممکن است در نظر گرفتن بخشی از آن‌ها به عنوان مجموعه آزمون مناسب نباشد.

۲. چون این رهیافت یک آزمایش آموزش ـ و ـ آزمون تکی است، اگر تقسیم کردن داده‌ها به دو مجموعه آموزش و آزمون به‌طور مناسب انجام نشده باشد، استنباط‌ها می‌توانند گمراه‌کننده باشند.

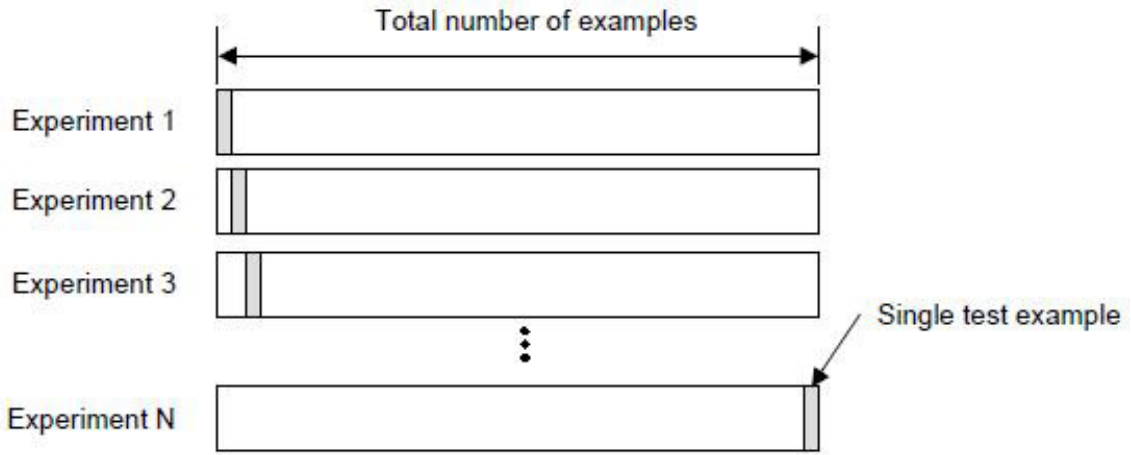
این محدودیت‌ها با در نظر گرفتن صورت‌های پیچیده‌تری از مجموعه‌های آموزشی و آزمون مرتفع می‌شوند:

- Leave-one-out cross-validation (LOOCV)
- Leave-k-out cross-validation
- K-Fold cross-validation
- ...

فرض کنید N مشاهده مستقل $y_1, \ldots, y_N$ را داریم.

در روش $LOOCV$، اندازه‌های دقت به صورت زیر به دست می‌آیند:

۱. فرض کنید مشاهده iام مجموعه آزمون را تشکیل دهد و مدل را با بقیه مشاهدات برازش دهیم. سپس خطای $e_i^* = y_i - \hat{y}_i$ را برای مشاهده حذف‌شده محاسبه می‌کنیم. به این خطا گاهی خطای پیشگویی نیز می‌گویند.



۲. مرحله ۱ را برای $i = 1, \ldots, N$ تکرار می‌کنیم.

۳. MSE را برحسب $e_1^*, \ldots, e_N^*$ محاسبه می‌کنیم که به آن کمترین توان‌های دوم خطای پیشگویی، $MSPE$، می‌گویند.

$LOOCV$: ادامه

این روش، استفاده خیلی کاراتری از داده‌های موجود است. زیرا در هر مرحله تنها یک مشاهده حذف می‌شود.

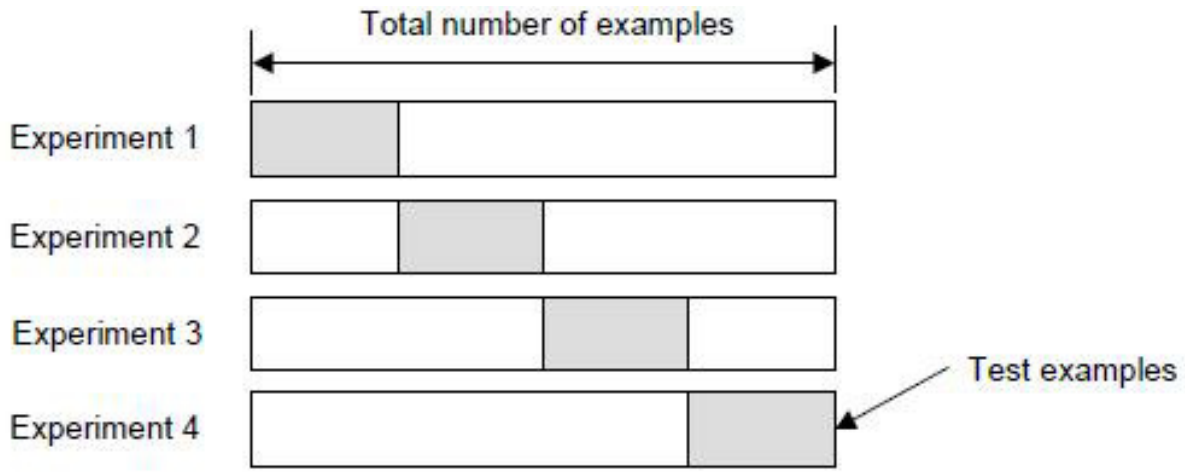
البته این روش، جز برای مدل‌های خطی، می‌تواند خیلی زمان‌بر باشد.

معیارهای دیگری مانند MAE به‌طور مشابه قابل محاسبه‌اند. یک معیار مرتبط آماره $PRESS$ است که برابر $N \times MSPE$ می‌باشد.

اعتبارسنجی متقابل با حذف هر بار k مشاهده

یکی از صورت‌های اعتبارسنجی متقابل، شامل تشکیل مجموعه‌های آزمون با حجم k مشاهده در هر مرحله است.

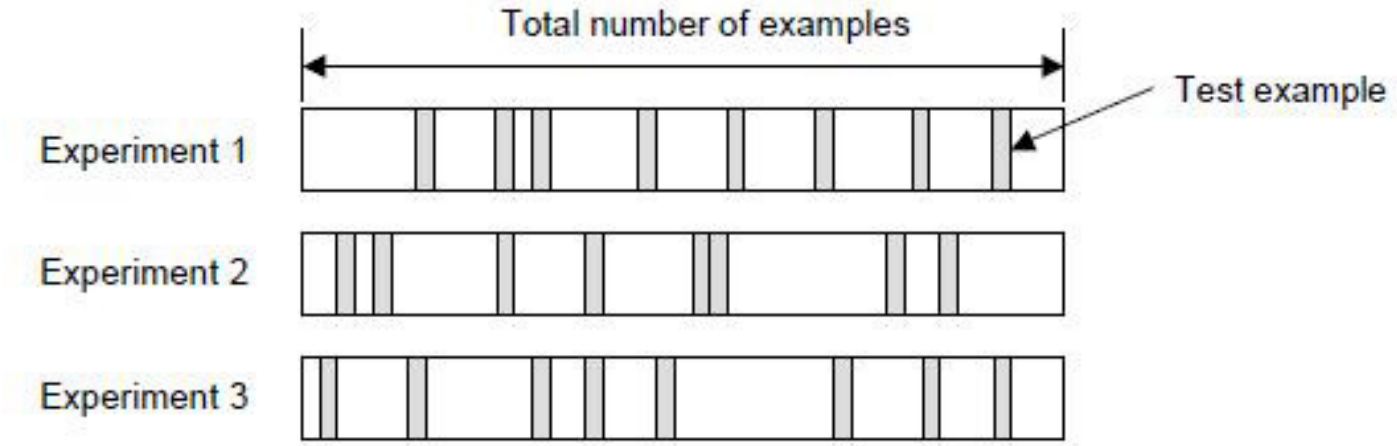
یکی از مزیت‌های این رهیافت، محاسبات کمتر آن است. اما در مورد نحوه انتخاب k باید دقت و توجه کافی داشت.



یک انتخاب معمول $k = ۱۰$ است.

اعتبارسنجی متقابل با بازنمونه‌گیری تصادفی

یکی دیگر از روش‌های اعتبارسنجی متقابل، اعتبارسنجی متقابل k *fold* است که در آن نمونه اصلی به طور تصادفی به k زیرنمونه تقسیم می‌شود و در هر مرحله، یکی از آن‌ها به عنوان مجموعه آزمون در نظر گرفته می‌شود.



یک نسخه متداول دیگر *bootstrap* + ۰.۶۳۲ است که توسط افرون و تیبشیرانی (۱۹۹۷) معرفی شد و ویژگی‌های بهتری دارد، اما اجرای آن پیچیده است.

Efron, B. & Tibshirani, R. (1997), *Improvements on Cross-Validation: The .632+ Bootstrap Method*, JASA, 92, 548-560

کاربردها

به‌طور کلی با می‌نیمم کردن $MSPE$ (یا آماره‌های مشابه)، می‌توان مدل آماری با قدرت پیشگویی بهتر را انتخاب کرد.

از اعتبارسنجی متقابل در قسمت‌های مختلفی از استنباط آماری، به‌ویژه انتخاب مدل، استفاده می‌شود:

- سنجش نیکویی برازش مدل‌ها و انتخاب بهترین مدل
- ارزیابی پایداری برآوردهای پارامترها
- سنجش دقت رده‌بندی الگوریتم‌های رده‌بندی
- انتخاب پارامترهای میزان‌ساز، $tuning\ parameters$، مدل‌های آماری مختلف مانند: درجه آزادی یک هموارساز ناپارامتری، پارامتر k در روش kـ نزدیکترین همسایگی، kNN، پارامترهای هسته در روش SVM.

به طور عملی، انتخاب مدل بر اساس CV خیلی بهتر از انتخاب مدل مبتنی بر آزمون‌های آماری است و تقریباً یک اندازه نااریب از MSE واقعی برای داده‌های جدید ارایه می‌کند.

مثال

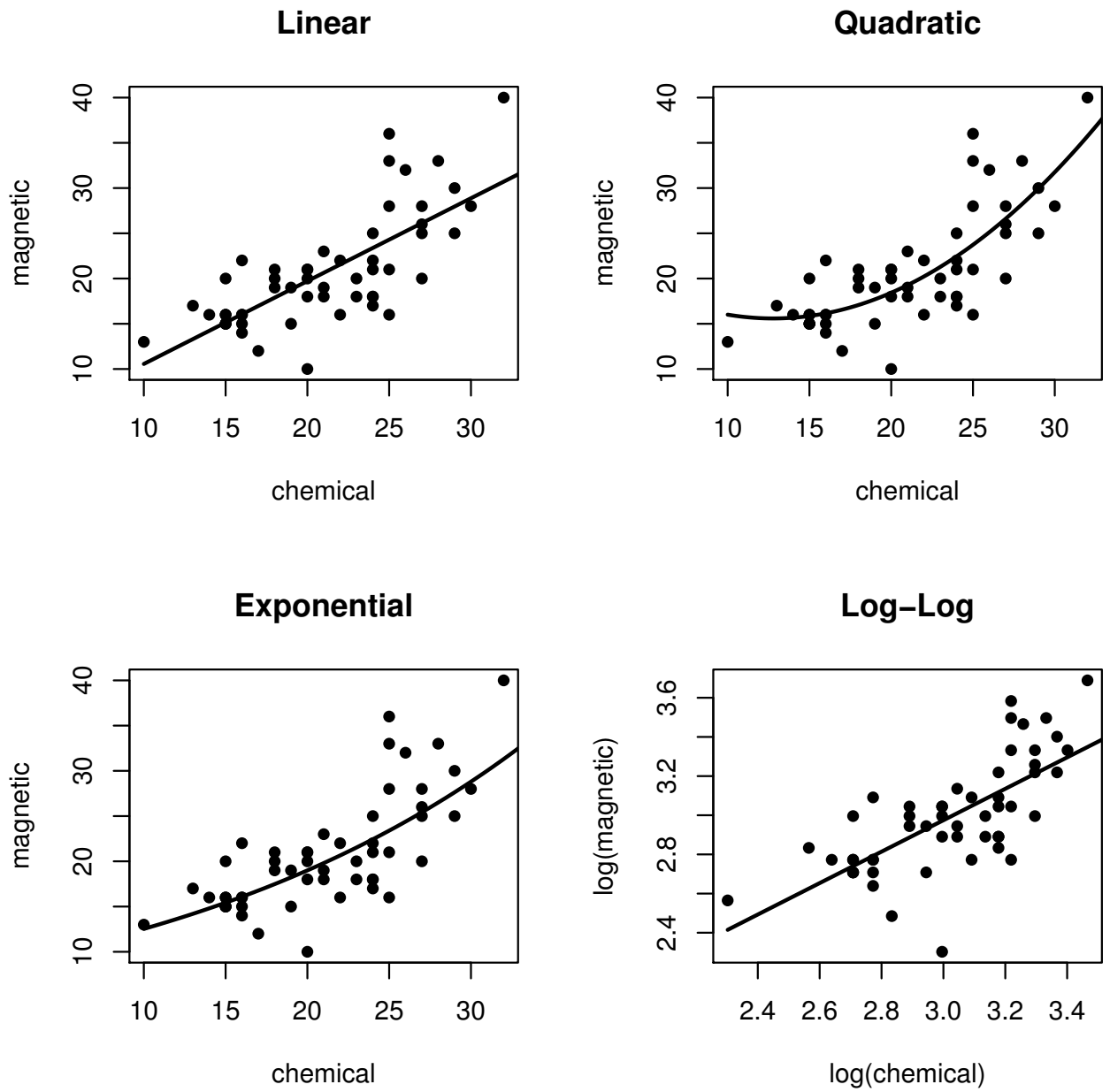
مجموعه داده $ironslag$ که در بسته $DAAG$ موجودند، شامل ۵۳ اندازه‌گیری میزان آهن به دو روش شیمیایی و مغناطیسی است.

با توجه به شکل پراکنش داده‌ها، این احساس وجود دارد که ممکن است رابطه بین این دو متغیر، خطی نباشد.

برای این داده‌ها سه رابطه دیگر: درجه دو، نمایی و لگاریتمی در نظر می‌گیریم و مدل‌های مربوط به آن‌ها را به صورت زیر معرفی می‌کنیم:

$$\begin{aligned}
Linear: \quad Y &= \beta_0 + \beta_1 X + \epsilon \\
Quadratic: \quad Y &= \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon \\
Exponential: \quad \log(Y) &= \beta_0 + \beta_1 X + \epsilon \\
Logarithmic: \quad \log(Y) &= \beta_0 + \beta_1 \log(X) + \epsilon
\end{aligned}$$

روش‌های متفاوتی برای انتخاب بهترین مدل، بر حسب هدف، وجود دارند. در این مثال بر روش مبتنی بر خطای پیشگویی که با CV قابل برآورد است، متمرکز می‌شویم.


Linear
magnetic
chemical
10
15
20
25
30
10
20
30
40
Quadratic
magnetic
chemical
10
15
20
25
30
10
20
30
40
Exponential
magnetic
chemical
10
15
20
25
30
10
20
30
40
Log−Log
log(magnetic)
log(chemical)
2.4
2.6
2.8
3.0
3.2
3.4
2.4
2.8
3.2
3.6

محاسبه خطای پیشگویی

بر اساس $LOOCV$ خطای پیشگویی را می‌توان به صورت زیر برآورد کرد:

۱. برای $i = ۱, \ldots, n$ مشاهده (x_i, y_i) را مجموعه آزمون در نظر بگیرید و از بقیه داده‌ها برای برازش مدل استفاده کنید:

الف) مدل را با $n - ۱$ مشاهده (x_j, y_j)، که در آن $j \neq i$، در مجموعه آموزشی برازش دهید.

ب) مقادیر پیشگویی $\hat{y}_i$ را برای مجموعه آزمون به دست آورید. مثلا برای مدل خطی به صورت $\hat{y}_i = \hat{\beta}_۰ + \hat{\beta}_۱ x_i$ خواهد بود.

ج) خطای پیشگویی $e_i^* = y_i - \hat{y}_i$ را محاسبه کنید.

۲. MSE خطاهای پیشگویی را به صورت $MSPE = \frac{۱}{n} \sum_{i=۱}^{n} e_i^{*۲}$ محاسبه کنید.

مدلی که دارای کمترین $MSPE$ باشد، به عنوان بهترین مدل انتخاب می‌شود.

انتخاب بهترین مدل

```
n <- length(magnetic)   #in DAAG ironslag
e1 <- e2 <- e3 <- e4 <- numeric(n)
# fit models on leave-one-out samples
for (k in 1:n) {
     y <- magnetic[-k]
     x <- chemical[-k]
##
     J1 <- lm(y ~ x)
     yhat1 <- J1$coef[1] + J1$coef[2] * chemical[k]
     e1[k] <- magnetic[k] - yhat1
##
     J2 <- lm(y ~ x + I(x^2))
     yhat2 <- J2$coef[1] + J2$coef[2] * chemical[k] +
              J2$coef[3] * chemical[k]^2
     e2[k] <- magnetic[k] - yhat2
##
     J3 <- lm(log(y) ~ x)
     logyhat3 <- J3$coef[1] + J3$coef[2] * chemical[k]
     yhat3 <- exp(logyhat3)
     e3[k] <- magnetic[k] - yhat3
##
     J4 <- lm(log(y) ~ log(x))
     logyhat4 <- J4$coef[1] + J4$coef[2] * log(chemical[k])
     yhat4 <- exp(logyhat4)
     e4[k] <- magnetic[k] - yhat4
}
> c(mean(e1^2), mean(e2^2), mean(e3^2), mean(e4^2))
[1] 19.55644 17.85248 18.44188 20.45424
```

آیا همیشه می‌توان از CV استفاده کرد؟

دقت کنید که اعتبارسنجی متقابل همیشه قابل به کارگیری نیست.

به عنوان مثال، در یک مدل رگرسیونی، اگر دو یا بیشتر از دو مشاهده با مقادیر دقیقاً یکسان برای همه متغیرهای تبیینی و متغیر پاسخ y وجود داشته باشد، آن‌گاه خارج کردن یک مشاهده ($LOOCV$) موثر نخواهد بود.

برآورد سازگار مدل با CV

یک معیار انتخاب مدل را سازگار گویند، هرگاه اگر، در رده‌ای از مدل‌ها، مدل درست وجود داشته باشد، بتواند با افزایش n آن را شناسایی کند.

در یک مقاله معروف، شائو (۱۹۹۷) نشان داد $LOOCV$ به برآوردی سازگار منتهی نمی‌شود.

Shao, J. (1997), *An Asymptotic Theory for Linear Model Selection*, Statistica Sinica, 7, 221-264

به عبارت دیگر $LOOCV$، معیار سازگاری برای انتخاب مدل نیست. یعنی $LOOCV$ همواره قادر به پیدا کردن مدل درست نیست (البته به گفته مشهور باکس باید توجه داشت):

همه مدل‌ها نادرستند، اما برخی از آن‌ها مفیدند

به عبارتی، پذیرفتن وجود مدل درست در رده مدل‌های مورد نظر، چندان واقع‌بینانه نیست!!

برآورد سازگار مدل با CV : ادامه

در مقابل، انواع مشخصی از اعتبارسنجی متقابل با حذف k مشاهده (به‌طور دقیق‌تر زمانی که k همراه با n افزایش می‌یابد)، سازگار خواهند بود.

در عمل، یک مشکل این معیار آن است که تغییر کوچک در داده‌ها می‌تواند باعث تغییر زیاد در مدل انتخاب شده شود.

محققین مختلفی دریافته‌اند که اعتبارسنجی متقابل $k\ fold$، از این منظر، دارای عملکرد بهتری است.

اعتبارسنجی متقابل برای مدل‌های خطی

به‌طور کلی CV می‌تواند از نظر محاسباتی بسیار زمان‌بر و پرهزینه باشد.

اما برای مدل‌های خطی، محاسبه $LOOCV$ بسیار سریع و ساده است.

مدل خطی را در نظر بگیرید:

$$Y = X\beta + \epsilon.$$

می‌دانیم

$$\hat{\beta} = (X'X)^{-1}X'Y$$

و مقادیر برازش‌شده به صورت زیر محاسبه می‌شوند:

$$\hat{Y} = X\hat{\beta} = X(X'X)^{-1}X'Y = HY$$

چون برای محاسبه $\hat{Y}$ از H استفاده می‌شود، به آن ماتریس hat می‌گویند.

اعتبارسنجی متقابل برای مدل‌های خطی: ادامه

اگر مقادیر روی قطر اصلی H را با $h_1, \ldots, h_n$ نشان دهیم، آماره اعتبارسنجی متقابل (همان MSE باقی‌مانده‌های پیشگویی e_i^*) به صورت زیر قابل محاسبه است:

$$MSPE = \frac{1}{n} \sum_{i=1}^{n} \left[\frac{e_i}{1 - h_i} \right]^2 ,$$

که در آن، e_i باقی‌مانده‌های حاصل از برازش مدل بر روی همه n مشاهده است.

بنابراین برای مدل خطی لازم نیست n مدل جدا برازش داده شود.

این نتیجه بسیار جالب، این امکان را میسر می‌سازد که برای محاسبه $MSPE$ تنها یک بار مدل بر روی کل داده‌ها برازش داده شود.

رابطه با سایر معیارها

آماره‌های CV و معیارهای مرتبط با آن به شدت در آمار طرفدار دارند و مورد استفاده قرار می‌گیرند.

وجود رابطه سایر معیارهای انتخاب مدل با آماره‌های CV، به‌طور واضح، نشان داده نشده است. در این‌جا به چند مورد شناخته‌شده اشاره می‌کنیم:

- جک‌نایف: همان طور که در اسلایدهای قبلی مطرح شد، برآوردگر جک‌نایف مشابه $LOOCV$ محاسبه می‌شود، با این تفاوت که به جای محاسبه باقی‌مانده‌های e_i^* در هر تکرار، تابعی مورد نظر θ محاسبه می‌شود.
- AIC: معیار اطلاع آکاییک به صورت $AIC = -2\log\mathcal{L} + 2p$ تعریف می‌شود که در آن، $\mathcal{L}$ مقدار ماکسیمم تابع درستنمایی و p تعداد پارامترهای آزاد مدل است. به‌طور مجانبی، می‌نیمم کردن AIC معادل می‌نیمم کردن آماره CV است. این نتیجه برای هر مدلی، نه فقط مدل خطی، برقرار است. این ویژگی باعث شده است زمانی که هدف پیشگویی است، استفاده از AIC توصیه شود.

رابطه با سایر معیارها: ادامه

- BIC: معیار اطلاع بیزی به صورت $BIC = -۲ \log \mathcal{L} + p \log(n)$ تعریف می‌شود. به دلیل جریمه سنگین‌تر این معیار نسبت به AIC، مدلی که توسط BIC انتخاب می‌شود یا همان مدل منتخب بر اساس AIC است یا مدلی با تعداد پارامتر کمتر. شائو (۱۹۹۷) نشان داد، برای مدل‌های خطی، به‌طور مجانبی می‌نیمم کردن BIC معادل اعتبارسنجی متقابل با حذف ν مشاهده است، به‌طوری که

$$\nu = n[۱ - ۱/(\log(n) - ۱)].$$

یک منبع جامع و عالی برای روش‌های اعتبارسنجی متقابل، آرلوت و سلیسه (۲۰۱۰) است.

Arlot, S. & Celisse, A. (2010), *A Survey of Cross-Validation Procedures for Model Selection*, Statistics Surveys, 4, 40-79

اعتبارسنجی متقابل برای سری‌های زمانی

وقتی که داده‌ها مستقل نیستند، اجرای CV خیلی سختتر می‌شود، زیرا کنار گذاشتن یک مشاهده تمام اطلاعات مرتبط با آن را، به دلیل وابستگی با سایر مشاهدات، حذف نمی‌کند.

برای حالت خاصی که وابستگی بین داده‌ها به دلیل ماهیت وابسته به زمان بودن آن‌هاست، (یعنی در سری‌های زمانی)، یک آماره CV به صورت زیر به دست می‌آید:

۱. مدل را به داده‌های $y_1, \ldots, y_t$ برازش داده و $\hat{y}_{t+1}$ را مقدار پیش‌بینی برای مشاهده بعدی قرار می‌دهیم. سپس خطای $e^*_{t+1} = y_{t+1} - \hat{y}_{t+1}$ را به دست می‌آوریم.

۲. مرحله ۱ را برای $t = m, \ldots, n-1$ تکرار می‌کنیم، که در آن m می‌نیمم تعداد مشاهداتی است که برای برازش مدل نیاز داریم.

۳. مقدار MSE را برای $e^*_{m+1}, \ldots, e^*_n$ محاسبه می‌کنیم.

اعتبارسنجی متقابل برای داده‌های وابسته با ساختارهای وابستگی پیچیده، مانند مدل‌های آمیخته یا فضایی، می‌تواند خیلی مشکل و پرهزینه باشد.

اجرای CV در R

در R بسته‌ها و توابع متفاوتی وجود دارند که در رده‌های مشخصی از مدل‌های آماری، اعتبارسنجی متقابل را اجرا می‌کنند. در این جا به سه تا از آن‌ها اشاره می‌کنیم:

- تابع $validate$ در بسته $Design$ برای مدل‌های خطی و لجستیک، روش CV را اجرا می‌کند. این تابع، اعتبارسنجی مبتنی بر روش خودگردان‌سازی و همچنین $bootstrap$ + ۶۳۲. را نیز به عنوان انتخاب‌هایی فراهم آورده است.
- بسته $DAAG$: این بسته دارای سه تابع $cv.lm$، $CVlm$ و $CVbinary$ است که روش اعتبارسنجی متقابل با بازنمونه‌گیری تصادفی را در مدل‌های به ترتیب رگرسیون ساده، رگرسیون چندگانه و لجستیک، اجرا می‌کند.
- بسته $boot$: تابع $cv.glm$ روش اعتبارسنجی متقابل با بازنمونه‌گیری تصادفی k تایی را برای مدل‌های خطی تعمیم‌یافته (GLM) از جمله دوجمله‌ای، گاوسی، پواسون، گاما و غیره، اجرا می‌کند. اگر k برابر تعداد مشاهدات مشخص شود، آن‌گاه روش $LOOCV$ خواهد بود که پیش‌فرض تابع نیز همین است.

محاسبات آماری پیشرفته

جلسه دوازدهم: روش‌های نمونه‌گیری $MCMC$

حسین باغیشنی، دانشگاه صنعتی شاهرود

۱۹ آذر ۱۳۹۱

مقدمه

روش‌های نمونه‌گیری مونت کارلوی زنجیر مارکوفی *Markov chain Monte Carlo* ($MCMC$)، یک چارچوب کلی از مجموعه روش‌هایی است که برای انتگرال‌گیری به روش مونت کارلو، توسط متروپولیس و همکاران (۱۹۵۳) و هستینگز (۱۹۷۰) معرفی شدند.

مساله برآورد انتگرال $\int g(x)dx$ به روش مونت کارلو را به یاد بیاورید.

برای این منظور ابتدا مساله انتگرال‌گیری را به یک مساله محاسبه امید ریاضی بر حسب یک تابع چگالی، مانند $f(\cdot)$، تبدیل می‌کردیم. یعنی:

$$\mathcal{I} = \int g(x)dx = \int h(x)f(x)dx = \mathbb{E}_f(h(X)).$$

آنگاه برآورد مونت کارلوی $\mathcal{I}$، برابر میانگین یک نمونه به دست آمده از توزیع f، وقتی که حجم نمونه بزرگ باشد (قانون اعداد بزرگ)، خواهد بود.

بنابراین مساله انتگرال‌گیری به مساله یافتن راهی برای تولید نمونه از چگالی هدف $f(\cdot)$، تبدیل می‌شود.

رهیافت $MCMC$ برای نمونه‌گیری از $f(\cdot)$:

تشکیل یک زنجیر مارکوف با توزیع مانای f: اجرای الگوریتم به تعداد کافی بزرگ، به طوری که زنجیر (به طور تقریبی) به توزیع مانای خود همگرا شود.

در واقع برای محاسبه انتگرال $\mathcal{I}$ لازم نیست از یک نمونه (مستقل) مستقیما تولید شده از توزیع f، استفاده شود. می‌توان، بدون تولید مستقیم از f، یک نمونه (وابسته) $X_1, \ldots, X_n$ را با استفاده از یک زنجیر مارکوف ارگودیک با توزیع مانای f، به دست آورد.

اکنون سوالی که مطرح می‌شود، چگونگی ساخت چنین زنجیری است که روش‌های $MCMC$ برای پاسخ به این سوال ارایه شده‌اند.

تعریف ۷.۱ صفحه ۲۶۸ کتاب رابرت و کسلا (۲۰۰۴)، روش $MCMC$ برای تولید از توزیع f را روش تولید یک زنجیر مارکوف ارگودیک با توزیع مانای f، معرفی می‌کند.

مروری کوتاه بر زنجیرهای مارکوف

یک زنجیر مارکوف $\{X^{(t)}\}$ دنباله‌ای از متغیرهای تصادفی وابسته

$$X^{(۰)}, X^{(۱)}, \ldots, X^{(t)}, \ldots$$

است، به طوری که توزیع احتمال $X^{(t)}$ به شرط مفروض بودن متغیرهای گذشته، فقط به $X^{(t-۱)}$ وابسته است.

به این توزیع احتمال شرطی، هسته انتقال ($Transition\ Kernel$) یا هسته مارکوف K می‌گویند:

$$X^{(t+۱)}|X^{(۰)}, X^{(۱)}, \ldots, X^{(t)} \sim K(X^{(t)}, X^{(t+۱)}).$$

به عنوان مثال، یک زنجیر مارکوف قدم زدن تصادفی ساده به صورت $X^{(t+۱)} = X^{(t)} + \epsilon_t$ قابل نمایش است، که در آن $\epsilon_t \sim N(۰, ۱)$ و مستقل از $X^{(t)}$ است. بنابراین هسته مارکوف متناظر است با یک چگالی $N(X^{(t)}, ۱)$.

ویژگی‌های زنجیرهای مارکوف: تحویل‌ناپذیری

زنجیرهای مارکوف $MCMC$ معمولا از یک ویژگی پایداری قوی برخوردارند. این ویژگی، وجود یک توزیع احتمال ماناست. یعنی توزیع احتمال f وجود دارد، به طوری که اگر $X^{(t)} \sim f$، آنگاه $X^{(t+1)} \sim f$.

در نتیجه، به طور رسمی، هسته و توزیع مانا در معادله زیر صدق می‌کنند:

$$\int_{\chi} K(x, y) f(x) dx = f(y).$$

وجود یک توزیع مانا، شرط اولیه‌ای را که در نظریه زنجیرهای مارکوف معروف به تحویل‌ناپذیری ($Irreducibility$) است، بر K تحمیل می‌کند. که اگر زنجیر دارای این شرط نباشد، زنجیر مفیدی نخواهد بود.

تحویل‌ناپذیری به این معنی است که هسته K اجازه حرکت بر روی تمام وضعیت‌های فضای حالت زنجیر را دارد.

یا به عبارت ساده‌تر، صرفنظر از مقدار اولیه $X^{(0)}$، دنباله $\{X^{(t)}\}$ با احتمال مثبت به هر ناحیه از فضای حالت می‌تواند سر بزند.

ویژگی‌های زنجیر مارکوف: بازگشتی

وجود یک توزیع مانا، نتایج اساسی بر روی رفتار زنجیر $\{X^{(t)}\}$ دارد. یکی از آن نتیجه‌ها این است که اغلب زنجیرهایی که در الگوریتم‌های $MCMC$ دخیل هستند، بازگشتی می‌باشند.

که اگر زنجیر این ویژگی را نداشته باشد، مفید نخواهد بود.

بازگشتی بودن به این معنی است که زنجیر به هر مجموعه دلخواه غیر قابل اغماض، به دفعات نامتناهی باز خواهد گشت.

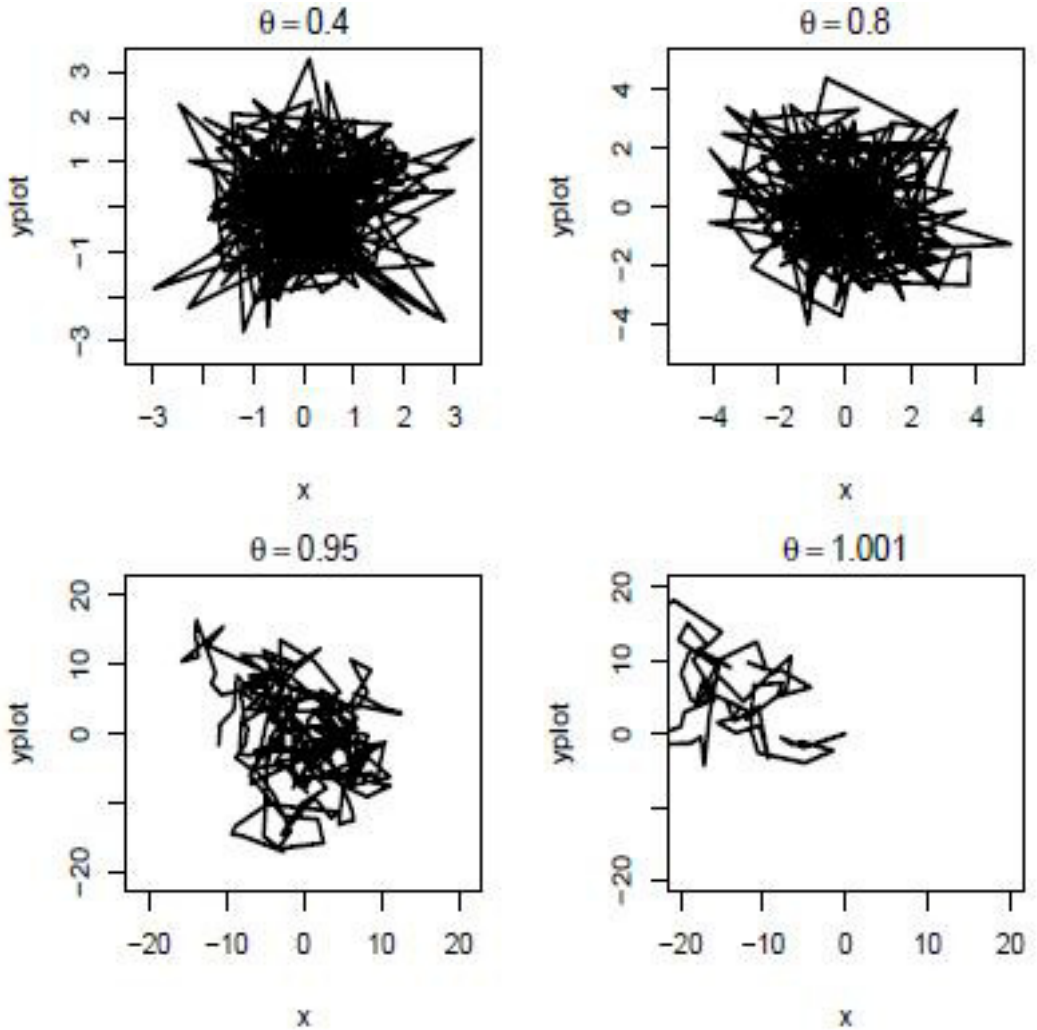
برای تشریح ویژگی بازگشتی، فرآیند $AR(1)$ را در نظر بگیرید:

$$X_t = \theta X_{t-1} + \epsilon_t; \quad \theta \in \Re$$

و $\epsilon_t \sim N(0, \sigma^2)$. اگر ϵ_tها مستقل باشند، X_t، به شرط مفروض بودن X_{t-1}، از $X_{t-2}, X_{t-3}, \ldots$ مستقل است.

توزیع مانای این زنجیر مارکوف عبارتست از $N(0, \frac{\sigma^2}{1-\theta^2})$ که باید شرط $|\theta| < 1$ برقرار باشد.

مثال: فرآیند $AR(1)$



تمرین

رنجیر مارکوف $AR(۱)$ را در نظر بگیرید که در آن $\epsilon_t \sim N(۰, ۱)$.

با شبیه‌سازی $X^{(۰)} \sim N(۰, ۱)$، هیستوگرام یک نمونه از $X^{(t)}$ را برای $t \leq ۱۰^۴$ و $\theta = ۰/۹$ رسم کنید.

بررسی کنید آیا توزیع مانای $N(۰, \frac{۱}{۱-\theta^۲})$ به این هیستوگرام برازش خوبی دارد؟

ویژگی‌های زنجیر مارکوف: ارگودیک بودن

در زنجیرهای بازگشتی، توزیع مانا یک توزیع حدی نیز هست.

به این معنی که توزیع حدی $X^{(t)}$ به ازای هر مقدار اولیه $X^{(0)}$، f است.

این ویژگی، ارگودیک بودن نیز نامیده می‌شود.

به عبارت دیگر ویژگی ارگودیک بودن زنجیر، معادل است با استقلال نسبت به شرایط اولیه.

برای زنجیرهای ارگودیک و برای توابع انتگرال‌پذیر h:

$$\frac{1}{T}\sum_{t=1}^{T} h(X^{(t)}) \longrightarrow \mathbb{E}_f[h(X)]$$

و به این معنی است که قانون قوی اعداد بزرگ برای رهیافت $MCMC$ نیز قابل کاربرد است.

که به آن قضیه ارگودیک نیز می‌گویند.

همگرایی

بیشتر از این در مورد جزییات زنجیرهای مارکوف بحث نخواهم کرد. کسانی که علاقه‌مند به جزییات بیشتر هستند، می‌توانند به فصل ۶ کتاب رابرت و کسلا (۲۰۰۴) مراجعه کنند.

با این حال مایلم حالتی را که همگرایی زنجیر به یک توزیع مانا هرگز اتفاق نمی‌افتد و مهم هم هست، مطرح کنم.

و آن حالت در چارچوب بیزی زمانی است که توزیع پسین، سره ($Proper$) نیست. زیرا در این حالت، زنجیر نمی‌تواند بازگشتی باشد.

استفاده از توزیع‌های پیشین ناسره در مدل‌های پیچیده آماری خیلی معمول است:

- گاهی موارد توزیع پسین سره می‌شود و الگوریتم $MCMC$ بازگشتی خواهد بود
- گاهی موارد نیز توزیع پسین سره نمی‌شود و الگوریتم $MCMC$ کار نخواهد کرد.

مسایل انتگرال‌گیری در استنباط بیزی

اگرچه الگوریتم‌های $MCMC$ روش‌های نمونه‌گیری عمومی هستند، اما اغلب کاربردهای آن‌ها در مسایل مربوط به استنباط بیزی دیده می‌شود.

در یک مدل بیزی، هر دوی مشاهدات و پارامترها متغیر تصادفی محسوب می‌شوند. برای یک نمونه n تایی، توزیع توام (x, θ) عبارتست از:

$$f_{x,\theta} = f_{x|\theta}(x_1, \ldots, x_n|\theta) f_\theta(\theta).$$

بنابراین می‌توان توزیع پیشین θ را با شرطی کردن بر روی اطلاعات موجود در داده‌ها، با استفاده از قاعده بیز، به‌روز کرد (توزیع پسین):

$$f_{\theta|x}(\theta|x) = \frac{f_{x|\theta}(x) f_\theta(\theta)}{\int f_{x|\theta}(x) f_\theta(\theta)\, d\theta}.$$

مسایل انتگرال‌گیری در استنباط بیزی: ادامه

بنابراین، امید ریاضی شرطی تابعی مانند $g(\theta)$ نسبت به توزیع پسین عبارتست از:

$$\mathbb{E}[g(\theta|x)] = \int g(\theta) f_{\theta|x}(\theta)\, d\theta = \frac{\int g(\theta) f_{x|\theta}(x) f_\theta(\theta)\, d\theta}{\int f_{x|\theta}(x) f_\theta(\theta)\, d\theta}.$$

این امید ریاضی حتی زمانی که چگالی پسین $f_{\theta|x}$ بدون ثابت نرمال‌سازش معلوم باشد، قابل تقریب است. این گزاره مهمی است، زیرا در اغلب موارد محاسبه ثابت نرمال‌ساز برای چگالی‌های پسین، کار دشوار و طاقت‌فرسایی است.

اما مساله اصلی، رام نبودن چنین انتگرالی و مشکل محاسبه عددی آن برای ابعاد بالاست.

$MCMC$ روشی را برای این گونه مسایل انتگرال‌گیری فراهم می‌آورد.

تاکید دوباره

ایده:
تولید یک زنجیر ارگودیک $X^{(t)}$، با مقدار اولیه $x^{(0)}$، با استفاده از یک هسته انتقال که دارای توزیع مانای f است.

- همگرایی (در توزیع) $X^{(t)}$ به متغیری با توزیع f را تضمین می‌کند.
- برای یک مقدار به اندازه کافی بزرگ T_0، $X^{(T_0)}$ را می‌توان به عنوان تحققی از توزیع f در نظر گرفت.
- یک نمونه وابسته $X^{(T_0)}, X^{(T_0+1)}, \ldots$ تولید می‌کند که از توزیع f تولید شده است به طوری که این نمونه برای اهداف برآورد کافی خواهد بود.

مساله: چطور می‌توان زنجیر مارکوفی با یک توزیع مانای مفروض ساخت؟

تولید زنجیر مارکوف با توزیع مانای مشخص

روش کار با الگوریتم‌های $MCMC$ واضح و سر راست است:

- با فرض داشتن چگالی هدف f:
  - یک هسته مارکوف K با توزیع مانای f می‌سازیم
  - سپس زنجیر مارکوف $X^{(t)} \to X \sim f$ را تولید می‌کنیم
  - در نهایت انتگرال مورد نظر با استفاده از قضیه ارگودیک قابل محاسبه است
- الگوریتم متروپولیس ـ هستینگز، مثالی از این نوع روش‌هاست
  - با شرط داشتن f، از توزیع نامزد $q(y|x)$ تولید نمونه می‌کنیم
  - تنها چیزی که برای تصمیم‌گیری در مورد پذیرش یا عدم پذیرش نمونه نیاز داریم، آن است که نسبت $\frac{f(y)}{q(y|x)}$ به جز یک ثابت، معلوم باشد

به چگالی (شرطی) $q(y|x)$، چگالی ابزاری یا پیشنهادی ($Proposal$) می‌گویند. این چگالی هر چیزی می‌تواند باشد. تنها نیاز نظری برای انتخاب $q(\cdot|x)$، به جز شرط بالا، آن است که تکیه‌گاه آن به قدری وسیع باشد که کل تکیه‌گاه f را بتواند پوشش دهد.

الگوریتم متروپولیس ـ هستینگز

الگوریتم متروپولیس ـ هستینگز ($Metropolis - Hastings$) را می‌توان به صورت زیر بیان کرد:

با شرط داشتن $x^{(t)}$،

۱. $Y_t \sim q(y|x^{(t)})$ را تولید کن

۲. قرار بده

$$X^{(t+1)} = \begin{cases} Y_t & with\ Porbability\ \ \rho(x^{(t)}, Y_t) \\ x^{(t)} & with\ Probability\ \ 1 - \rho(x^{(t)}, Y_t) \end{cases}$$

که در آن

$$\rho(x, y) = \min\left\{1, \frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}\right\}$$

به $\rho(x, y)$ احتمال پذیرش متروپولیس ـ هستینگز می‌گویند.

اجرای الگوریتم در R

اجرای الگوریتم در R به سادگی قابل انجام است.

با فرض داشتن تابعی برای شبیه‌سازی از $q(y|x)$، مانند $geneq(x)$، اگر $x[t]$ مقدار $X^{(t)}$ را مشخص کند، یک کد کلی برای اجرای الگوریتم بالا به صورت زیر خواهد بود:

```
y = geneq(x[t])
if (runif(1)<f(y)*q(y,x[t])/(f(x[t])*q(x[t],y))){
    x[t+1]=y
   } else {
    x[t+1]=x[t]
}
```

همگرایی الگوریتم متروپولیس ـ هستینگز

نرخ پذیرش الگوریتم، میانگین احتمال پذیرش بر روی تکرارهاست. یعنی:

$$\bar{\rho} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} \rho(X^{(t)}, Y_t) = \int \rho(x,y) f(x) q(y|x)\, dy dx.$$

از این کمیت برای ارزیابی عملکرد الگوریتم، می‌توان استفاده کرد.

الگوریتم متروپولیس ـ هستینگز در شرطی موسوم به شرط تعادل دقیق
(*Detailed Balance Condition*)

$$f(x)K(y|x) = f(y)K(x|y)$$

صدق می‌کند. با انتگرال‌گیری از طرفین تساوی بالا نسبت به x، می‌توان نتیجه گرفت، توزیع مانای زنجیر $\{X^{(t)}\}$ برابر f است.

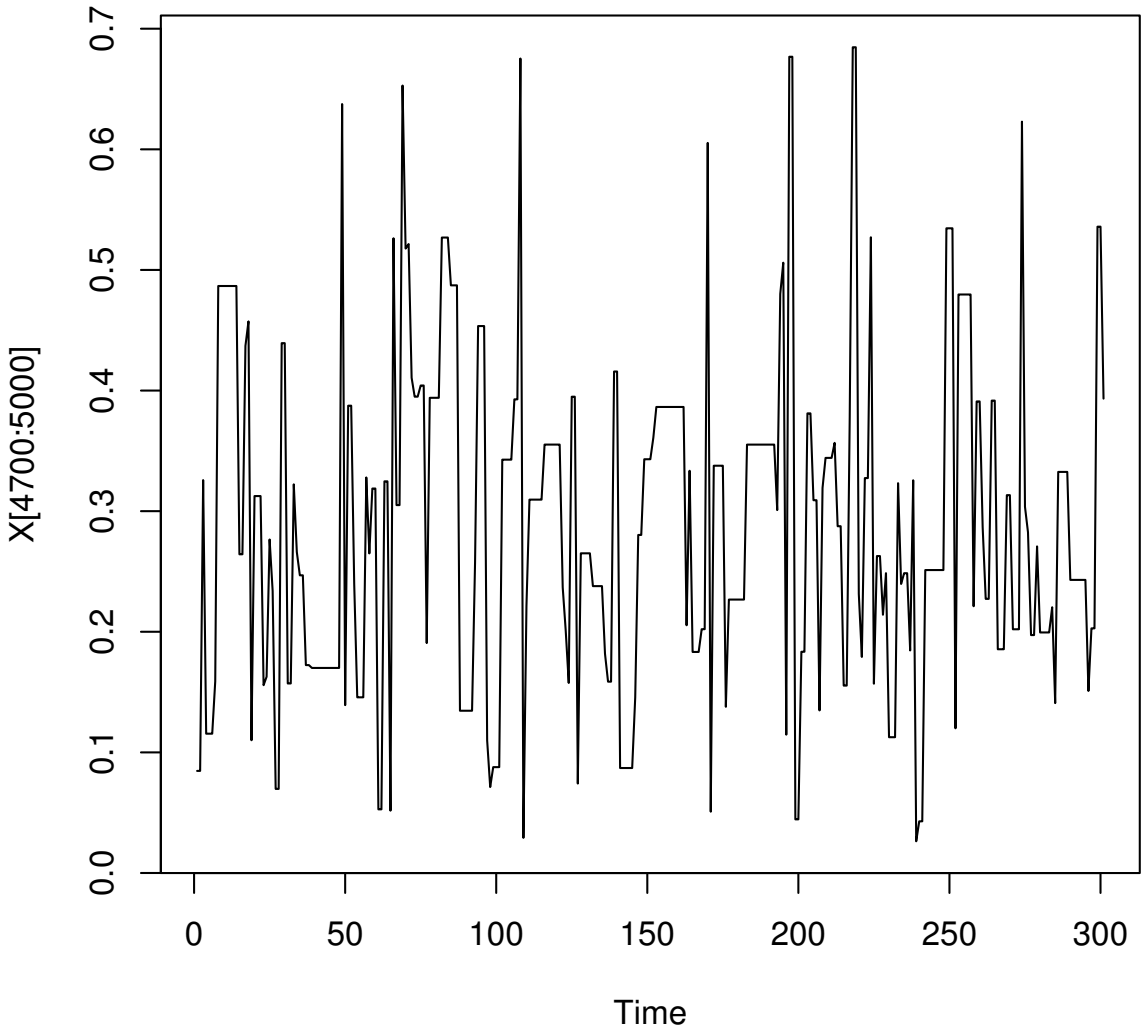
یک مثال ساده

در عمل، عملکرد الگوریتم MH قویا به انتخاب توزیع پیشنهادی q وابسته است. اما در این جا برای مقایسه با نمونه‌گیری مستقیم از یک توزیع، به مثال ساده‌ای توجه کنید.

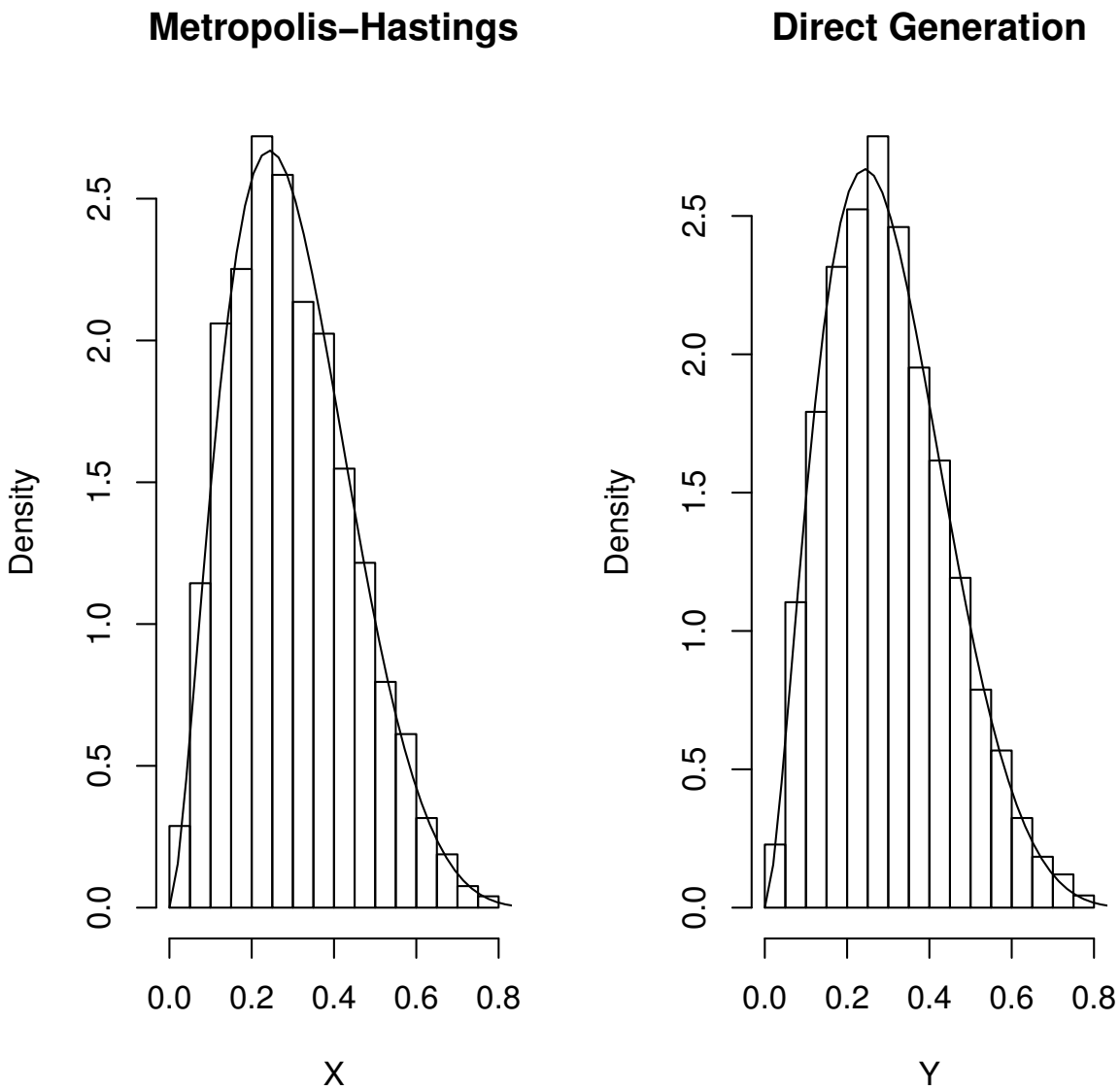
فرض کنید توزیع هدف $Beta(۲/۷, ۶/۳)$ و توزیع پیشنهادی $U(۰, ۱)$ باشد. یک نمونه MH را می‌توان با کد زیر تولید کرد:

```
a=2.7; b=6.3; c=2.669 # initial values
n=5000
X=rep(runif(1),n) # initialize the chain
for (i in 2:n){
    Y=runif(1)
    rho=dbeta(Y,a,b)/dbeta(X[i-1],a,b)
    X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<rho)
}
```

یک مثال ساده: ادامه



یک مثال ساده: ادامه



یک مثال ساده: ادامه

دقت کنید که اگرچه دو روش نمونه‌گیری به نظر یکسان هستند، اما نمونه $MCMC$ وابستگی دارند، در حالی که نمونه مستقیم iid هستند.

این به آن معنی است که در نمونه‌گیری $MCMC$، از کیفیت نمونه کاسته می‌شود یا به عبارت دیگر برای دستیابی به دقت یکسان با نمونه‌گیری مستقیم نیازمند حجم نمونه بزرگتری هستیم.

این مساله از طریق مفهومی به نام حجم نمونه موثر (*Effective Sample Size*) برای زنجیرهای مارکوف، مطرح می‌شود.

چند نکته

- برای توزیع‌های پیشنهادی متقارن، یعنی وقتی که $q(x|y) = q(y|x)$،

$$\rho(x_t, y_t) = \min\{1, \frac{f(y_t)}{f(x_t)}\}.$$

و بنابراین احتمال پذیرش مستقل از q خواهد بود.

- الگوریتم همیشه مقادیر y_t را که برای آن‌ها

$$\frac{f(y_t)}{q(y_t|x^{(t)})} > \frac{f(x^{(t)})}{q(x^{(t)}|y_t)},$$

می‌پذیرد. البته اگر جهت نامساوی عوض شود، ممکن است باز هم پذیرفته شود. اما اگر اختلاف زیاد باشد، مقادیر پیشنهادی اغلب موارد رد خواهند شد. این ویژگی نشان می‌دهد که چگونه انتخاب q می‌تواند بر روی عملکرد الگوریتم MH تاثیر بگذارد.

چند نکته: ادامه

- اگر تکیه‌گاه q در مقایسه با تکیه‌گاه f خیلی کوچک باشد، زنجیر مارکوف تولید شده در جستجوی دامنه f دچار مشکل خواهد شد و در نتیجه خیلی کند همگرا خواهد شد.
- الگوریتم MH فقط به نسبت‌های $\frac{f(y_t)}{f(x^{(t)})}$ و $\frac{q(x^{(t)}|y_t)}{q(y_t|x^{(t)})}$ وابسته است. در نتیجه الگوریتم مستقل از ثابت‌های نرمال‌ساز می‌باشد.
- زنجیر $\{x^{(t)}\}$ ممکن است چندین بار پشت سر هم مقادیر یکسانی داشته باشد، حتی اگر f یک چگالی (نسبت به اندازه لبگ) باشد.
- دنباله $\{y_t\}$ معمولا تشکیل یک زنجیر مارکوف نمی‌دهد. به این معنی که اگر مقادیر پذیرش‌شده را از زنجیر استخراج کنیم، تمام نتایج همگرایی به توزیع مانا خدشه‌دار خواهد شد و دیگر برقرار نخواهند بود.

الگوریتم MH مستقل

در حالتی که توزیع پیشنهادی q مستقل از $X^{(t)}$ باشد، الگوریتم حاصل با این انتخاب را الگوریتم MH مستقل گویند. در این حالت $q(y|x) = g(y)$.

با شرط داشتن $x^{(t)}$،

1. $Y_t \sim g(y)$ را تولید کن
2. قرار بده

$$X^{(t+1)} = \begin{cases} Y_t & with\ Porbability\ \ \min\left\{1, \frac{f(Y_t)g(x^{(t)})}{f(x^{(t)})g(Y_t)}\right\} \\ x^{(t)} & otherwise \end{cases}$$

ویژگی‌های الگوریتم MH مستقل

- این الگوریتم تعمیمی از روش رد و پذیرش است. چون در هر دو روش، توزیع ابزاری یکی است، مقادیر پیشنهادی Y_t مشابه هم هستند اما مقادیر پذیرش شده یکی نیستند.
- با اینکه این دو روش شباهت‌هایی دارند و می‌توان در یک مساله خاص از هر دو استفاده کرد و نتایج را مقایسه کرد، اما اگر در یک مساله بتوان از الگوریتم رد و پذیرش استفاده کرد، به ندرت از روش MH استفاده خواهد شد.
- نمونه‌های روش رد و پذیرش مستقل هستند، اما در مورد روش MH این طور نیست.
- در روش رد و پذیرش، نیازمند محاسبه کران بالای $M \geq \sup_x \frac{f(x)}{g(x)}$ هستیم. اما الگوریتم MH نیاز به محاسبه چنین کمیتی ندارد. این یک جنبه خوب روش MH است زمانی که محاسبه M زمان‌بر است یا M موجود ناکاراست و باعث تلف شدن بخش عمده‌ای از نمونه‌های پیشنهادی تولید شده می‌شود.

در واقع روش MH روش رد و پذیرش برای افراد تنبل است!

مثال

مجموعه داده $cars$ در هسته R، شامل سرعت و فاصله ترمز یک نمونه از اتومبیل‌هاست.

به این داده‌ها یک مدل درجه دو برازش می‌دهیم. یعنی مدل به صورت

$$y_{ij} = \beta_۰ + \beta_۱ x_i + \beta_۲ x_i^۲ + \epsilon_{ij},\ i = ۱, \ldots, k,\ j = ۱, \ldots, n_i$$

است، که در آن فرض می‌کنیم $\epsilon_{ij} \sim N(۰, \sigma^۲)$ و مستقل هستند.

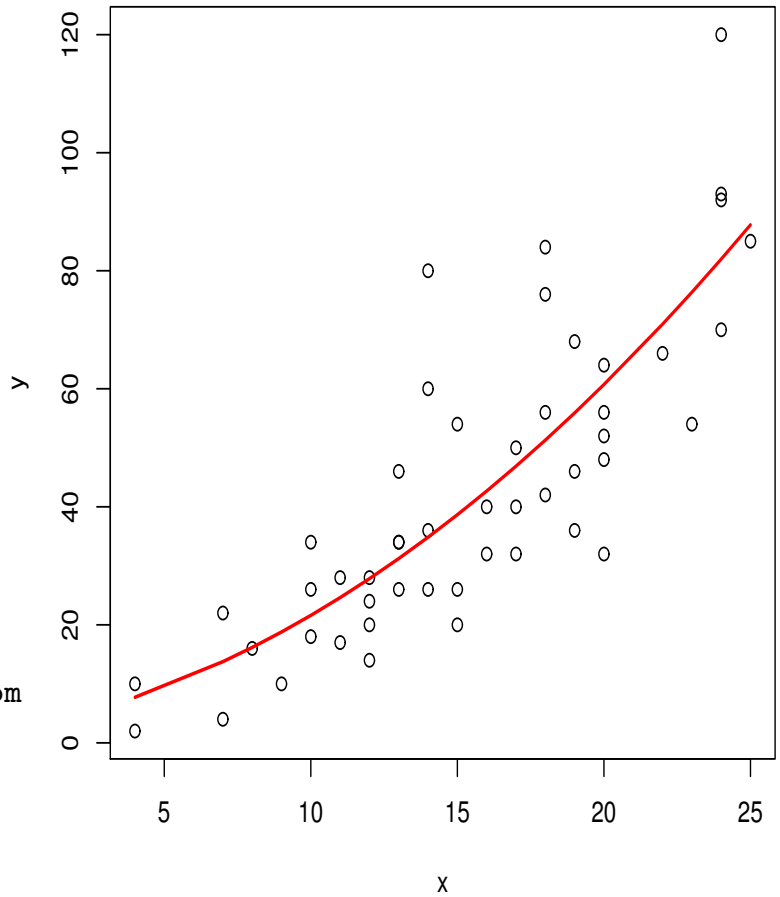
تابع درستنمایی متناسب است با

$$(\frac{۱}{\sigma^۲})^{N/۲} \exp\left\{-\frac{۱}{۲\sigma^۲}\sum_{ij}(y_{ij} - \beta_۰ - \beta_۱ x_i - \beta_۲ x_i^۲)^۲\right\}$$

که در آن $N = \sum_i n_i$ مجموع کل مشاهدات است.

برازش کمترین توان‌های دوم

```
data(cars)
x=cars$speed
y=cars$dist
x2=x^2
#
fit=lm(y~x+x2)
#
plot(y~x)
lines(x,predict(fit),col="red",lwd=2)
> summary(fit)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.47014   14.81716   0.167    0.868
x            0.91329    2.03422   0.449    0.656
x2           0.09996    0.06597   1.515    0.136
Residual standard error: 15.18 on 47 degrees of freedom
```



الگوریتم MH

می‌توان به تابع درستنمایی به عنوان توزیع پسین پارامترهای $\beta_0, \beta_1, \beta_2, \sigma^2$ نگاه کرد (به عنوان مثال با در نظر گرفتن توزیع‌های پیشین یکنواخت).

و به عنوان یک مثال برای نمایش نحوه اجرای الگوریتم، با استفاده از الگوریتم MH از این توزیع نمونه تولید کرد (دقت کنید چون توزیع نرمال است، به راحتی می‌توان به طور مستقیم از آن تولید نمونه کرد).

برای شروع، مقادیر اولیه، یعنی $X^{(0)} = (\beta_0^{(0)}, \beta_1^{(0)}, \beta_2^{(0)}, \sigma^{2(0)})$، برآوردهای کمترین توان‌های دوم در نظر می‌گیریم.

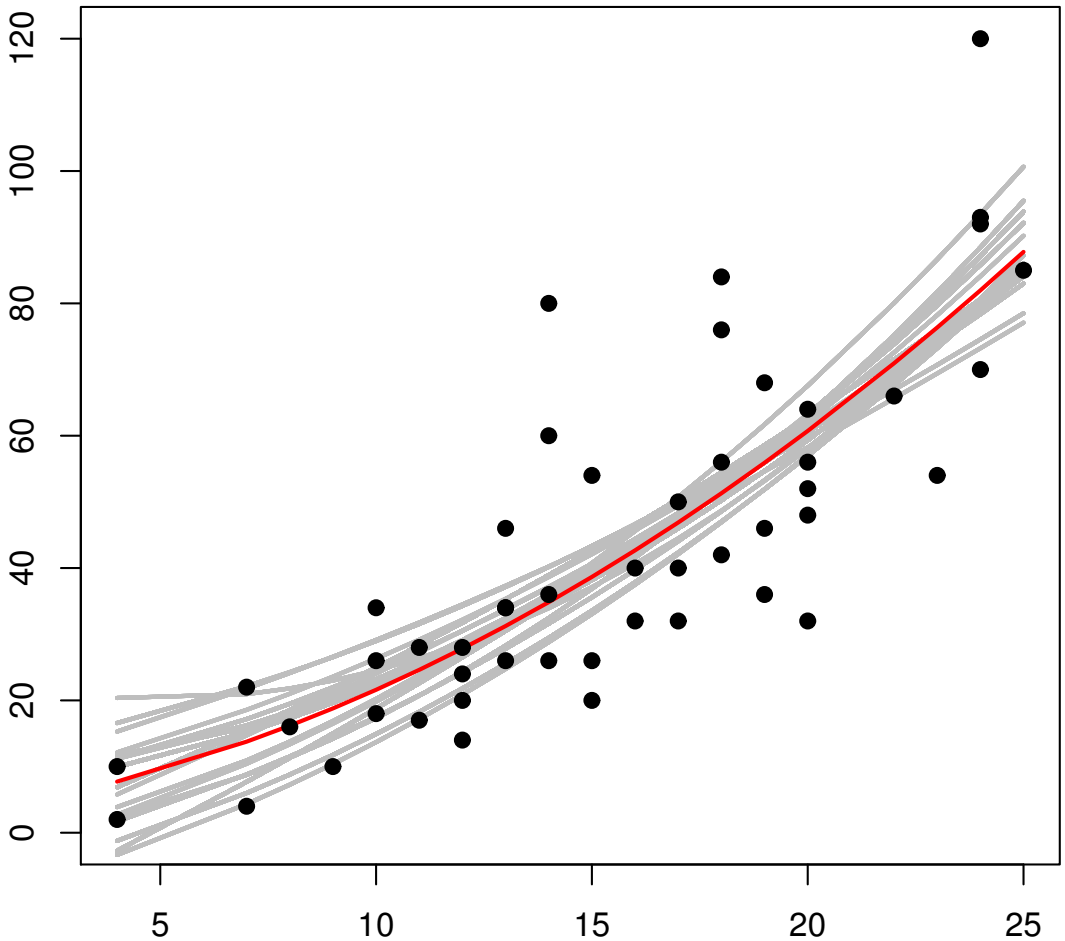
توزیع‌های پیشنهادی را نیز به صورت زیر در نظر می‌گیریم که همگی در MLE مرکزی شده‌اند:

$$\beta_0 \sim N(2/47, 15^2), \quad \beta_1 \sim N(0/91, 2^2), \quad \beta_2 \sim N(0/100, 0/06^2)$$

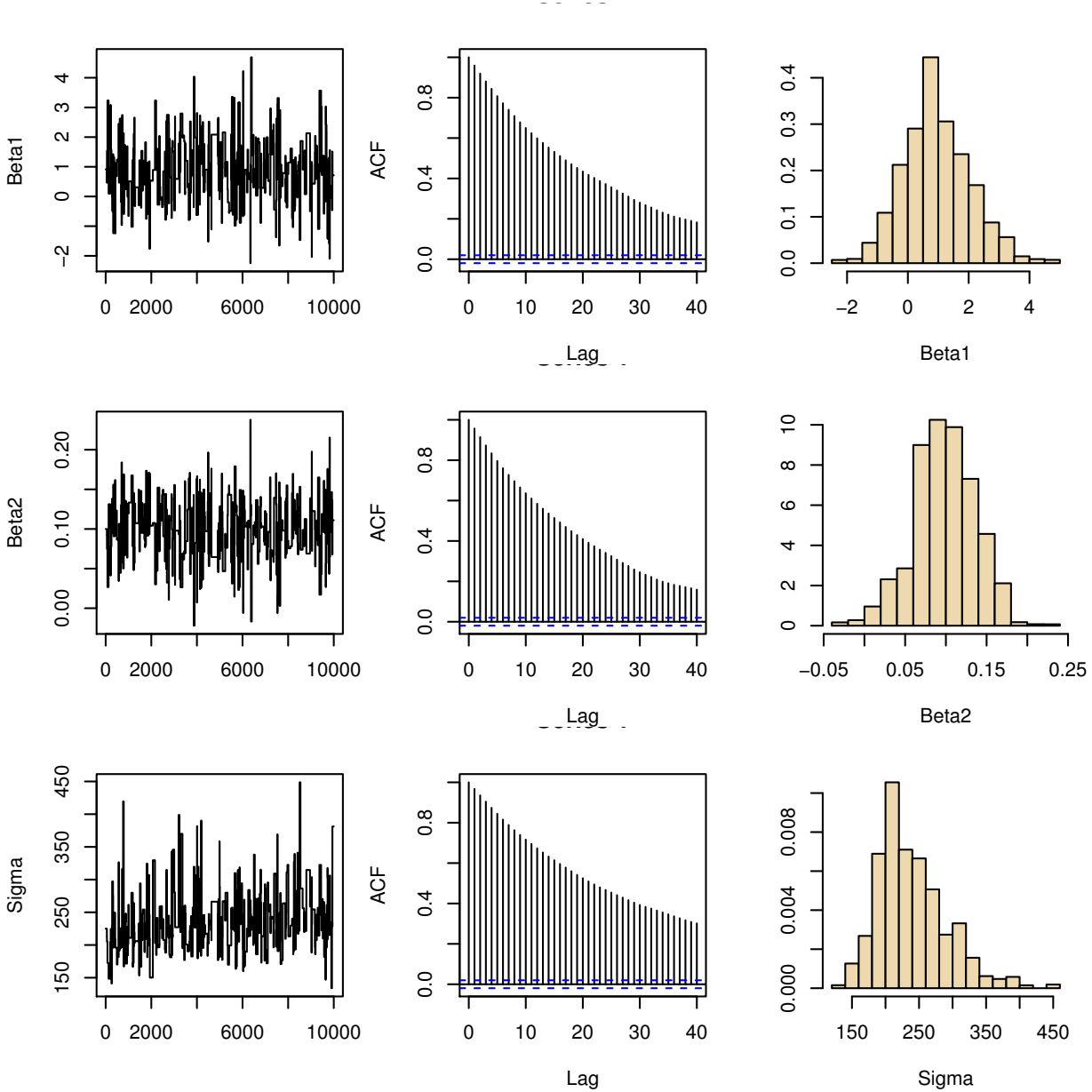
$$\sigma^{-2} \sim Gamm(N/2, (N-3)(15/18)^2)$$

```
Barking = function (nsim = 10^3){
    data = cars; x = data[, 1]; y = data[, 2]; n = length(x); x2 = x^2
    MSR = 225;  SSR = 10809
    bhat = coefficients(lm(y ~ x + x2))
    loglike = function(a, b, c, s2) { -(n/2) * log(s2) - sum((y - a - b * x - c * x2)^2)/(2 * s2) }
    dcand = function(a, b, c, s2) {
        dnorm(a, bhat[1], sd = 15, log = TRUE) + dnorm(b, bhat[2],
            sd = 2, log = TRUE)
        +dnorm(c, bhat[3], sd = 0.06, log = TRUE) - (n/2) * log(s2) -
            SSR/(2 * s2)
    }
    b1hat = array(bhat[1], dim = c(nsim, 1))
    b2hat = array(bhat[2], dim = c(nsim, 1))
    b3hat = array(bhat[3], dim = c(nsim, 1))
    s2hat = array(MSR, dim = c(nsim, 1))
    for (i in 2:nsim) {
        bcand = c(rnorm(1, mean = bhat[1], sd = 15), rnorm(1,
            mean = bhat[2], sd = 2), rnorm(1, mean = bhat[3],
            sd = 0.06), 1/rgamma(1, n/2, rate = SSR/2))
        test = min(exp(loglike(bcand[1], bcand[2], bcand[3],
            bcand[4]) - loglike(b1hat[i - 1], b2hat[i - 1], b3hat[i -
            1], s2hat[i - 1]) + dcand(b1hat[i - 1], b2hat[i -
            1], b3hat[i - 1], s2hat[i - 1]) - dcand(bcand[1],
            bcand[2], bcand[3], bcand[4])), 1)
        rho <- (runif(1) < test)
        b1hat[i] = bcand[1] * rho + b1hat[i - 1] * (1 - rho)
        b2hat[i] = bcand[2] * rho + b2hat[i - 1] * (1 - rho)
        b3hat[i] = bcand[3] * rho + b3hat[i - 1] * (1 - rho)
        s2hat[i] = bcand[4] * rho + s2hat[i - 1] * (1 - rho)
    }
    return(list('Beta0'=b1hat, 'Beta1'=b2hat,'Beta2'=b3hat,'Sigma'=s2hat))
}
```

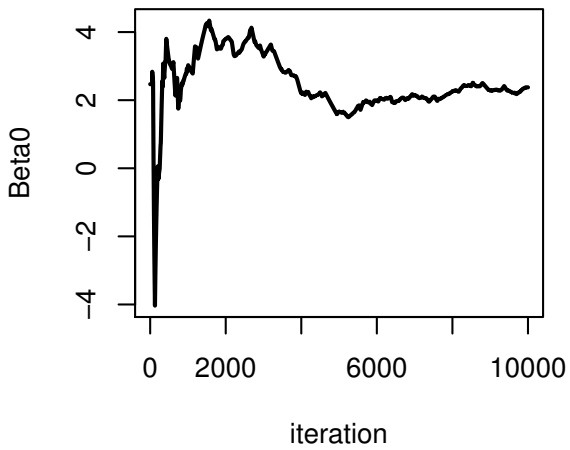
الگوریتم MH : نتیجه برازش

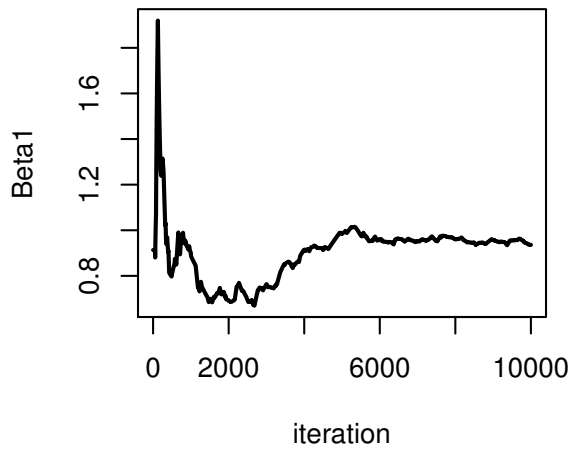


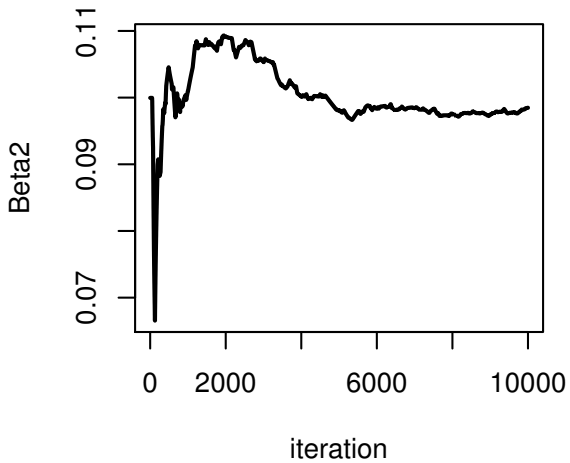
الگوریتم MH : آمیختگی زنجیر

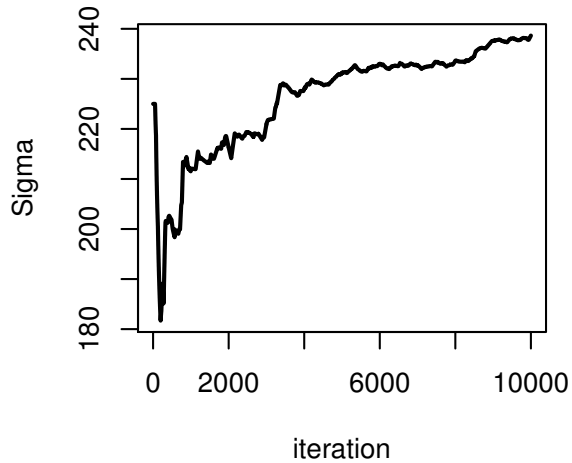


الگوریتم MH: بررسی همگرایی زنجیر









الگوریتم MH قدم زدن تصادفی: مقدمه

اجرای الگوریتم MH مستقل گاهی مشکل‌ساز می‌شود:

- ساخت چگالی پیشنهادی ممکن است پیچیده باشد
- توزیع‌های پیشنهادی در این نوع الگوریتم، اطلاعات موضعی را نادیده می‌گیرند

یک راه جانشین، گردآوری اطلاعات مرحله به مرحله است:

- یعنی کاوش همسایگی موقعیت جاری زنجیر
- این ایده، برای تولید مقدار بعدی زنجیر، نمونه تولید شده قبلی را به حساب می‌آورد. به این معنی که در همسایگی موقعیت جاری زنجیر، کاوش موضعی بیشتری را انجام می‌دهد.

الگوریتم MH قدم زدن تصادفی

اجرای ایده کاوش موضعی توسط کاوش در همسایگی مقدار جاری زنجیر، شبیه‌سازی Y_t از رابطه

$$Y_t = X^{(t)} + \epsilon_t,$$

است، که در آن ϵ_t یک نوفه تصادفی مستقل از $X^{(t)}$ با تابع چگالی g است. به عنوان مثال، انتخاب توزیع یکنواخت برای نوفه به این معنی است که

$$Y_t \sim U(X^{(t)} - \delta, X^{(t)} + \delta)$$

یا توزیع نرمال به معنی

$$Y_t \sim N(X^{(t)}, \tau^2)$$

است. در این حالت توزیع پیشنهادی عبارتست از

$$q(y|x) = g(y - x).$$

اگر چگالی g حول صفر متقارن باشد، که معمولا این طور است، زنجیر مارکوف متناظر با آن، یک فرآیند قدم زدن تصادفی است.

الگوریتم

با شرط داشتن $x^{(t)}$،

۱. $Y_t \sim g(y - x^{(t)})$ را تولید کن

۲. قرار بده

$$X^{(t+1)} = \begin{cases} Y_t & with\ Porbability\ \ \min\left\{1, \frac{f(Y_t)}{f(x^{(t)})}\right\} \\ x^{(t)} & otherwise \end{cases}$$

ویژگی‌های الگوریتم

- به دلیل وجود مرحله پذیرش متروپولیس‌ـ هستینگز، زنجیر $\{X^{(t)}\}$ یک زنجیر قدم زدن تصادفی نیست.
- احتمال پذیرش به g وابسته نیست. به این معنی که، برای یک جفت مفروض $(x^{(t)}, y_t)$، احتمال پذیرش چه y_t از توزیع نرمال تولید شود چه از توزیع کوشی، یکسان خواهد بود.
- اما انتخاب gهای مختلف منجر به دامنه‌های مختلف مقادیر برای Y_t ها و در نتیجه نرخ‌های پذیرش متفاوت خواهد شد. بنابراین نمی‌توان گفت انتخاب g بر رفتار الگوریتم تاثیری ندارد.

کالبیدن پارامتر مقیاس

پارامتر مقیاس فرآیند قدم زدن تصادفی، که وابسته به واریانس نوفه تصادفی است، باید به گونه‌ای انتخاب شود (کالبیده شود) تا کاوش به خوبی انجام شود.

این کالبیدن برای رسیدن به یک تقریب مناسب از توزیع هدف در تعداد تکرار قابل قبول، خیلی مهم است.

- اگر واریانس جمله نوفه خیلی بزرگ باشد، اکثر نمونه‌های پیشنهادی رد می‌شوند و الگوریتم شدیدا ناکاراست
- وقتی که این واریانس خیلی کوچک است، اکثر نمونه‌های پیشنهادی پذیرفته شده و زنجیری تولید می‌شود که خیلی شبیه به یک فرآیند قدم زدن تصادفی است و باز هم ناکاراست.

یک راه برای انتخاب پارامتر مقیاس، بررسی نرخ‌های پذیرش است. اگر نرخ پذیرش الگوریتم در دامنه [۰/۱۵, ۰/۵] قرار گیرد، مناسب است.

مثال: تولید توزیع t

هدف: تولید نمونه از توزیع t با ν درجه آزادی، با استفاده از توزیع پیشنهادی $N(X^{t)}, \sigma^{۲})$ و کالبیدن مقیاس زنجیر با انتخاب مقادیر متفاوت برای σ

در این حالت داریم:

$$\frac{f(y_t)}{f(x^{(t)}} = \frac{(۱ + \frac{y_t^{۲}}{\nu})^{-(\nu+۱)/۲}}{(۱ + \frac{x^{(t)۲}}{\nu})^{-(\nu+۱)/۲}}$$

مثال: کد R

```
rw.Metropolis <- function(nu, sigma, x0, n) {
x <- numeric(n)
x[1] <- x0
u <- runif(n)
k <- 0
for (i in 2:n) {
     y <- rnorm(1, x[i-1], sigma)
     if (u[i] <= (dt(y, nu) / dt(x[i-1], nu)))
     x[i] <- y  else {
              x[i] <- x[i-1]
              k <- k + 1
              }
     }
     k=(1-k/n)
     return(list(x=x, k=k))
}
```

مثال: نرخ پذیریش

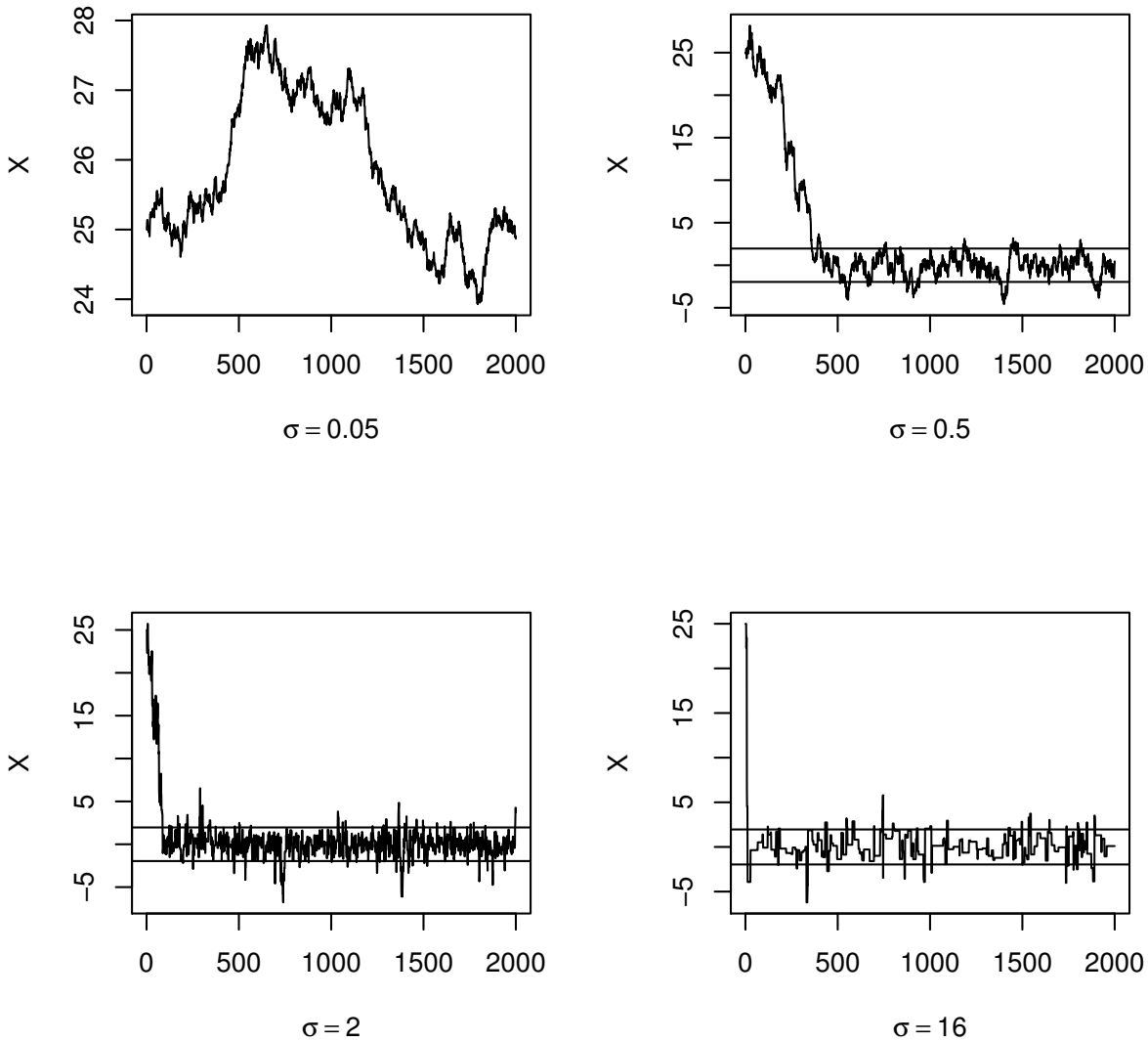
```
nu <- 4 # degrees of freedom
n <- 2000
sigma <- c(.05, .5, 2,  16)
x0 <- 25
##
rw1 <- rw.Metropolis(nu, sigma[1], x0, n)
rw2 <- rw.Metropolis(nu, sigma[2], x0, n)
rw3 <- rw.Metropolis(nu, sigma[3], x0, n)
rw4 <- rw.Metropolis(nu, sigma[4], x0, n)
# number of candidate points rejected
> print(c(rw1$k, rw2$k, rw3$k, rw4$k))
[1] 0.9955 0.8605 0.5460 0.0985
```

تنها زنجیر سوم مقداری نزدیک به دامنه [۰/۱۵, ۰/۵] دارد.

مثال: کد R برای تولید نمودارهای اثر و چندک‌ها

```
par(mfrow=c(2,2)) # display 4 graphs together
refline <- qt(c(.025, .975), df=n)
rw <- cbind(rw1$x, rw2$x, rw3$x,  rw4$x)
for (j in 1:4) {
     plot(rw[,j], type="l",
     xlab=bquote(sigma == .(round(sigma[j],3))),
     ylab="X", ylim=range(rw[,j]))
     abline(h=refline)
}
## Computing quantiles of target distribution and chains
a <- c(.05, seq(.1, .9, .1), .95)
Q <- qt(a, n)
rw <- cbind(rw1$x, rw2$x, rw3$x, rw4$x)
mc <- rw[501:N, ]
Qrw <- apply(mc, 2, function(x) quantile(x, a))
print(round(cbind(Q, Qrw), 3))
```

مثال: نمودارهای اثر



مثال: مقایسه چندک‌ها با مقادیر واقعی آن‌ها از توزیع t

	Q	V2	V3	V4	V5
5%	-1.65	24.30	-2.82	-2.20	-1.77
10%	-1.28	24.43	-2.05	-1.51	-1.38
20%	-0.84	24.85	-1.28	-0.89	-0.99
30%	-0.52	25.02	-0.83	-0.59	-0.64
40%	-0.25	25.22	-0.45	-0.27	-0.25
50%	0.00	25.83	-0.13	-0.05	0.09
60%	0.25	26.70	0.17	0.21	0.24
70%	0.52	26.89	0.47	0.51	0.92
80%	0.84	27.10	0.89	0.87	1.31
90%	1.28	27.42	1.38	1.30	1.79
95%	1.65	27.58	1.79	1.70	2.14

مثال: آمیخته نرمال

هدف: نمونه‌گیری از یک توزیع نرمال آمیخته دو متغیره دو مولفه‌ای با مشخصات زیر است:

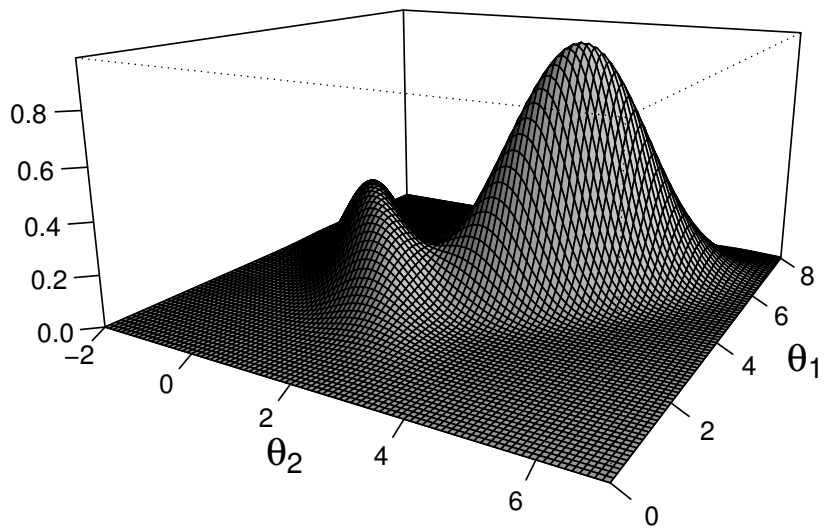
$$\mu_۱ = (۴, ۵)^T,\ \mu_۲ = (۰/۷, ۳/۵)^T,$$

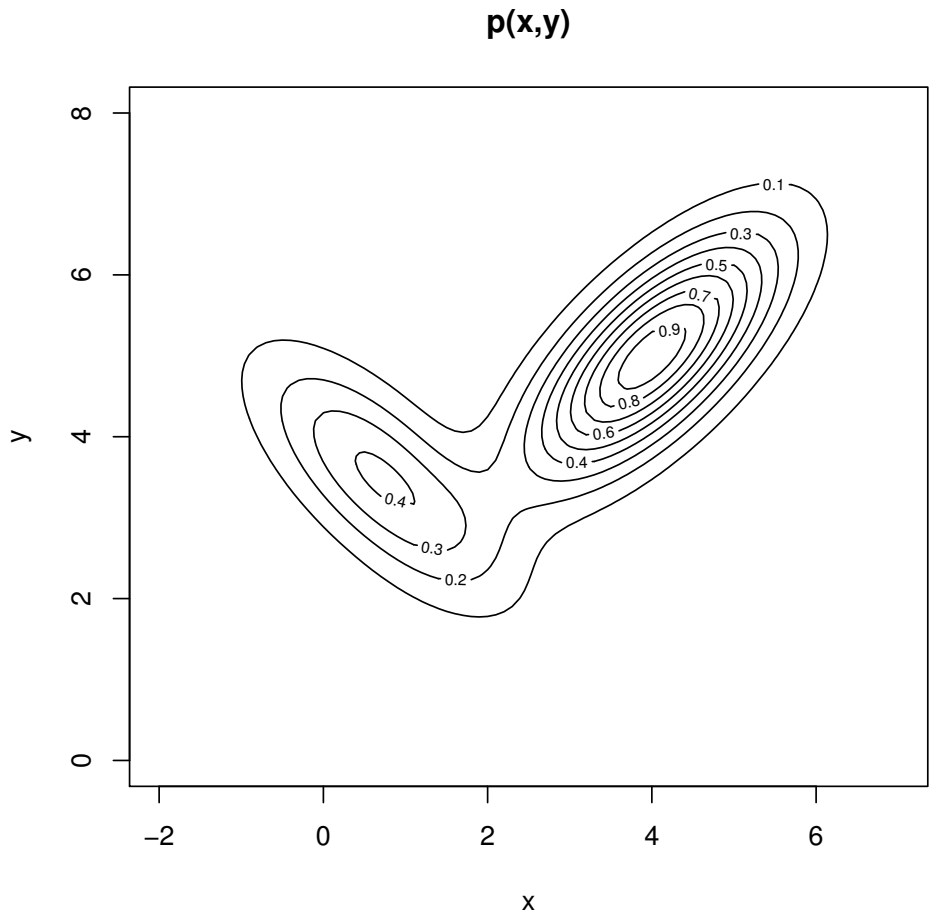
$$\Sigma_۱ = \begin{pmatrix} ۱ & ۰/۷ \\ ۰/۷ & ۱ \end{pmatrix},\ \Sigma_۲ = \begin{pmatrix} ۱ & -۰/۷ \\ -۰/۷ & ۱ \end{pmatrix}$$

توزیع پیشنهادی را نیز نرمال دومتغیره با بردار میانگین صفر و ماتریس قطری $\sigma \times I_۲$ در نظر می‌گیریم، که در آن سه مقدار متفاوت برای پارامتر مقیاس σ تعریف می‌کنیم:
$\sigma = (۰/۰۱, ۱, ۱۰۰)$.

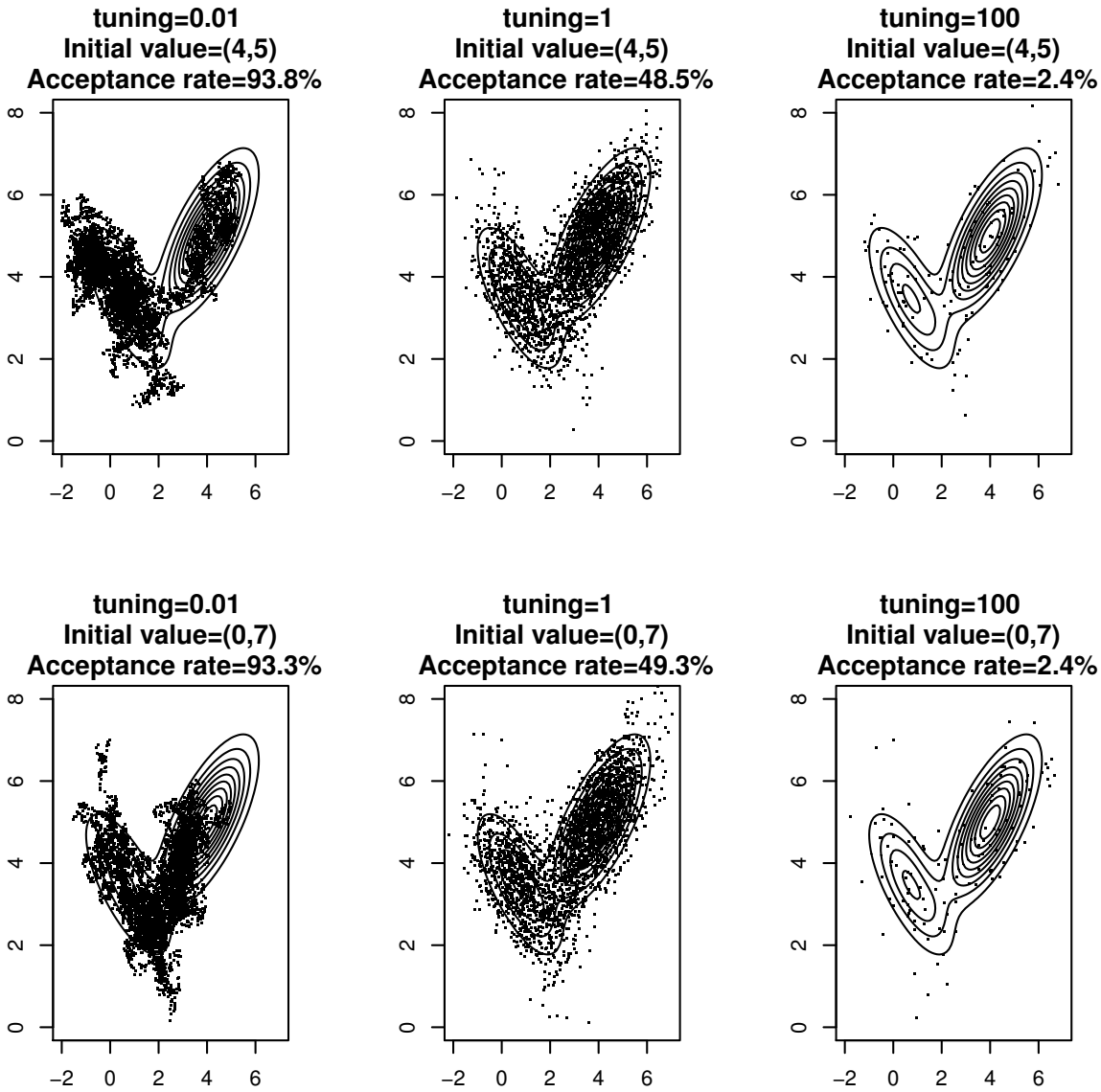
کد برنامه R از طریق صفحه شخصی من، قسمت تابلو اعلانات قابل دسترسی است.

مثال: آمیخته نرمال

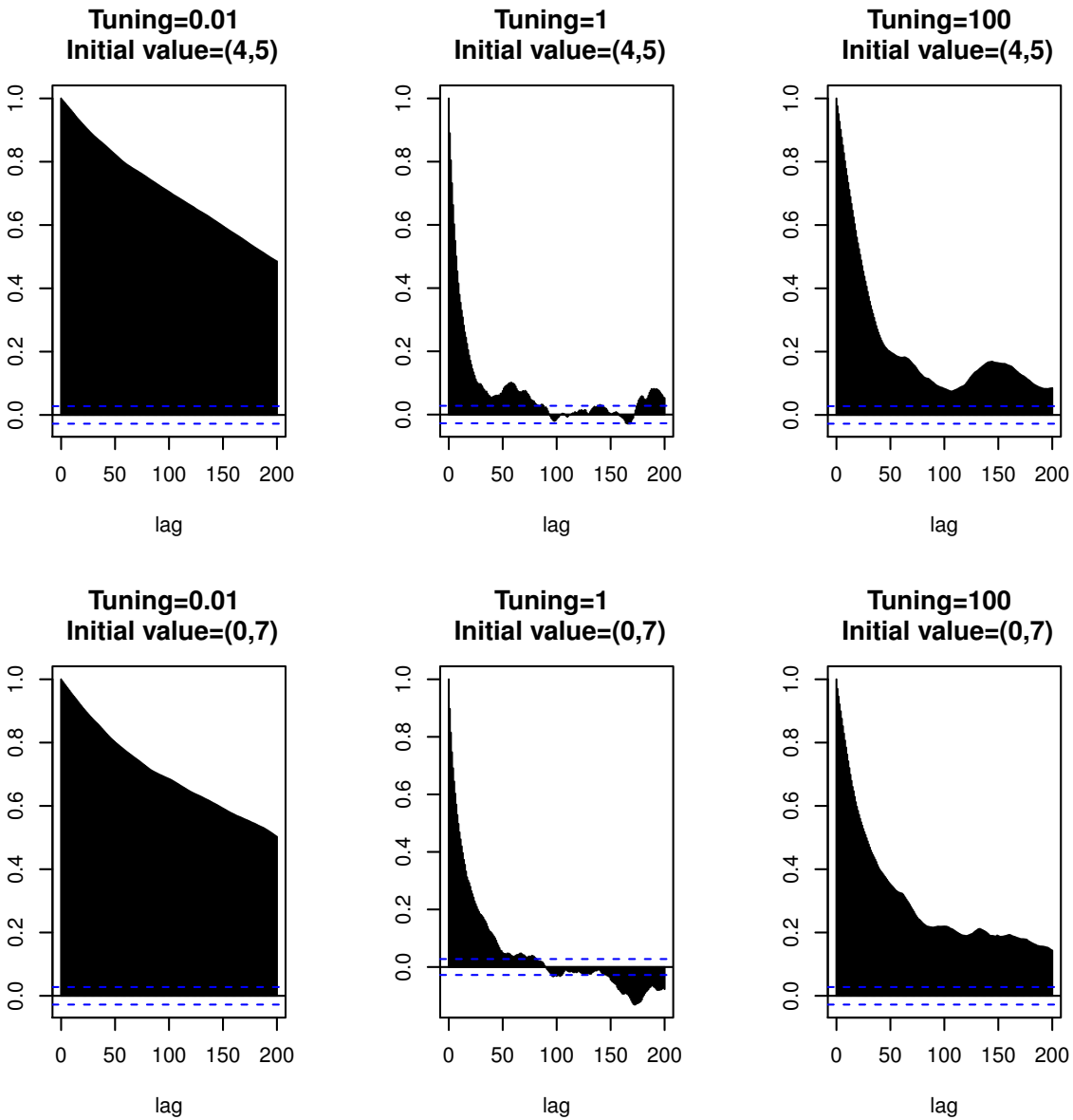




مثال: آمیخته نرمال



مثال: آمیخته نرمال



الگوریتم نمونه‌گیر گیبز: مقدمه

نام نمونه‌گیری گیبز از مقاله معروف گمان و گمان (۱۹۸۴) گرفته شده است.

اما استفاده شایع از این الگوریتم با کار گلفاند و اسمیت (۱۹۹۰) شروع شد، که از این الگوریتم برای حل مسایلی، در استنباط بیزی، که قبلا بدون حل مانده بودند، استفاده کردند.

الگوریتم‌های نمونه‌گیر گیبز، مسایل پیچیده (با بعد بالا)، مانند یک تابع چگالی هدف با بعد بالا، را به دنباله‌ای از مسایل ساده‌تر تقسیم می‌کنند

- ممکن است این مسایل پیچیده به وسیله الگوریتم MH قابل حل نباشند.

ممکن است دنباله مسایل ساده‌تر زمان زیادی برای همگرا شدن نیاز داشته باشند

- با این وجود این الگوریتم یک الگوریتم مفید و جالب توجه است.

الگوریتم نمونه‌گیر گیبز: مقدمه

الگوریتم گیبز، با استفاده از یک توزیع توام یک زنجیر مارکوف تشکیل می‌دهد:

- اگر دو متغیر تصادفی X و Y دارای توزیع توام $f(x, y)$ باشند
- به طوری که چگالی‌های شرطی متناظرشان $f_{X|Y}$ و $f_{Y|X}$ باشند
- الگوریتم یک زنجیر مارکوف (X_t, Y_t) را بر اساس مراحل زیر، می‌سازد:
  با شرط مفروض بودن $X_0 = x_0$، برای $t = 1, 2, \ldots,$ تولید کن:
  1. $Y_t \sim f_{Y|X}(\cdot|x_{t-1})$
  2. $X_t \sim f_{X|Y}(\cdot|y_t)$

چنانچه شبیه‌سازی از هر دو چگالی شرطی راحت باشد، اجرای الگوریتم بسیار روشن و ساده است.

توزیع مانای این زنجیر، $f(x, y)$ است و همگرایی زنجیر، به جز مواردی که تکیه‌گاه توزیع‌های شرطی بهم وصل نباشند، تضمین‌شده است.

مثال: نرمال دومتغیره

در اینجا مثال ساده‌ای را در نظر می‌گیریم: مدل نرمال دومتغیره

$$(X, Y) \sim N_۲ \left(۰, \begin{pmatrix} ۱ & \rho \\ \rho & ۱ \end{pmatrix} \right)$$

که برای آن نمونه‌گیر گیبز عبارتست از:

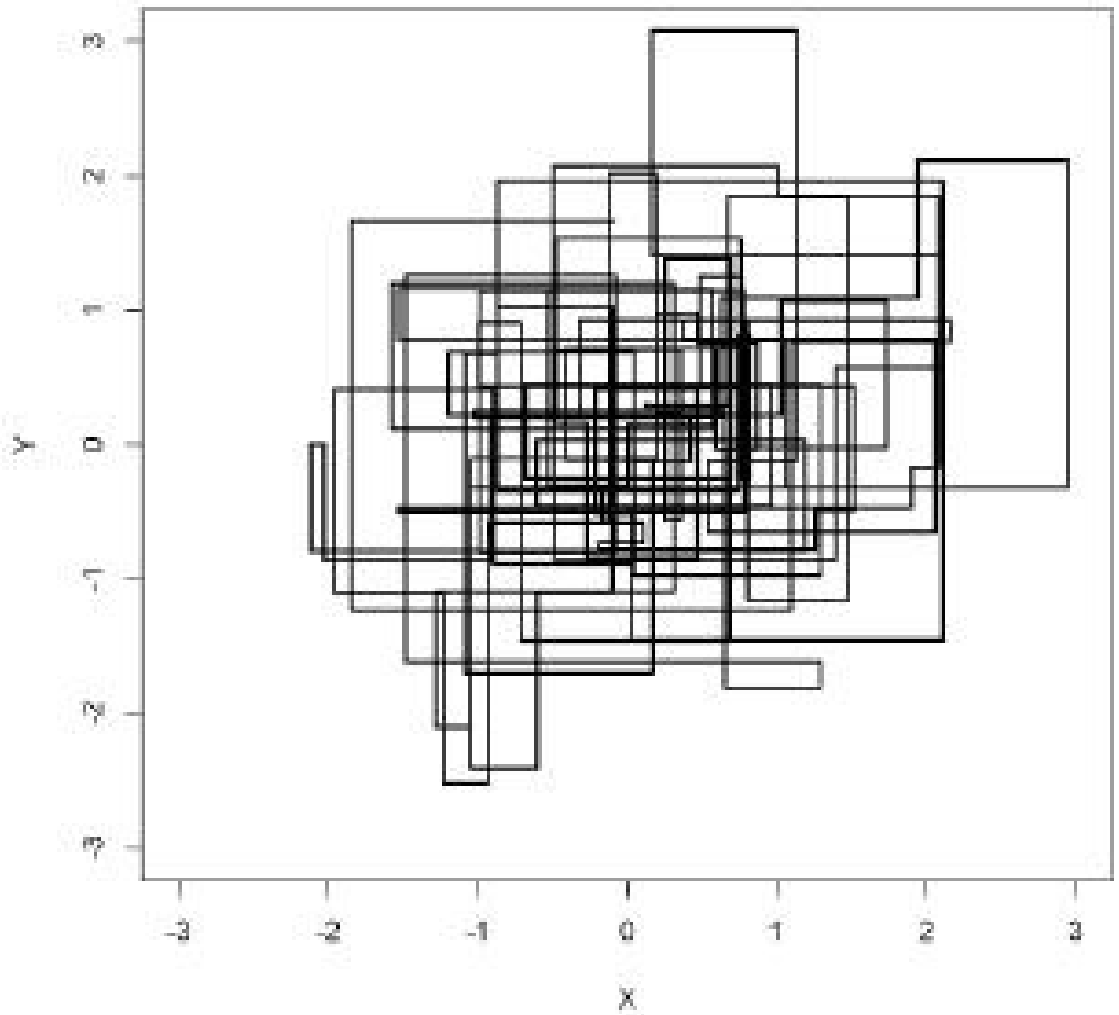
برای مقدار مفروض x_t، مقادیر زیر را تولید کن

$$\begin{aligned} Y_{t+1}|x_t &\sim N(\rho x_t, 1-\rho^2) \\ X_{t+1}|y_{t+1} &\sim N(\rho y_{t+1}, 1-\rho^2) \end{aligned}$$

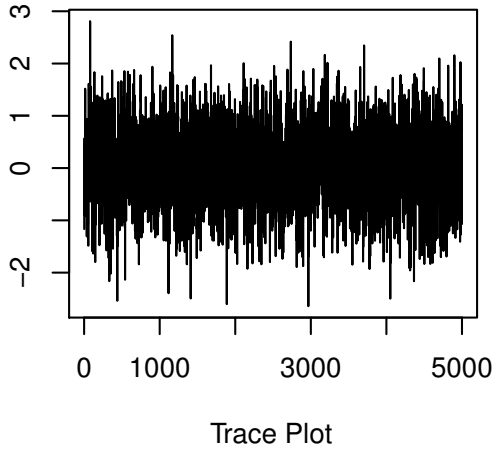
مثال: کد R

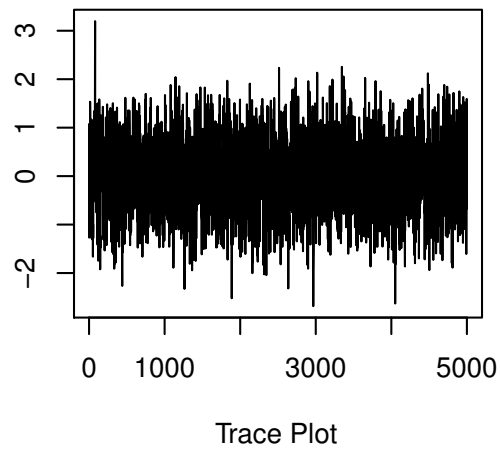
```
Nsim=5000 # initial values
rho=0.7
X=Y=array(0,dim=c(Nsim,1)) # init arrays
X[1]=rnorm(1) # init chains
Y[1]=rnorm(1)
for (i in 2:Nsim){ # sampling loop
     Y[i]=rnorm(1,rho*X[i-1],1-rho^2)
     X[i]=rnorm(1,rho*Y[i],1-rho^2)
}
mcmc=cbind(X,Y)
par(mfrow=c(2,2))
plot(ts(mcmc[,1]), xlab="Trace Plot", ylab="")
plot(ts(mcmc[,2]), xlab="Trace Plot", ylab="")
hist(mcmc[,1],40, main="", xlab="")
hist(mcmc[,2],40, main="", xlab="")
par(mfrow=c(1,1))
```

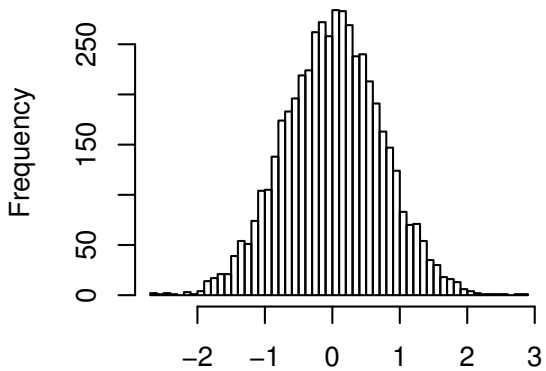
نمایش هندسی

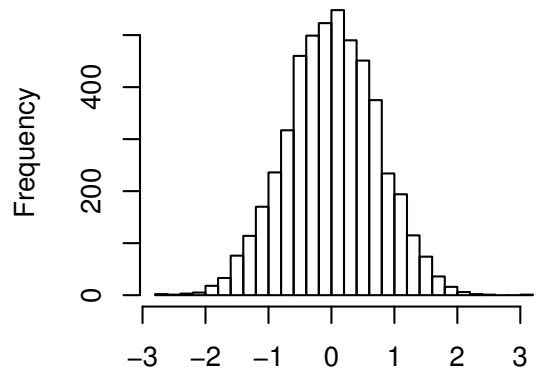


مثال: نمودارهای اثر









مثال: مدل نرمال

هدف: تولید نمونه از توزیع پسین (θ, σ^2) در مدل

$$X_i \sim N(\theta, \sigma^2),\ i = 1, \ldots, n$$

$$\theta \sim (\theta_0, \tau^2),\ \sigma^2 \sim IG(a, b)$$

می‌باشد، که در آن مقادیر θ_0، τ^2، a و b معلوم هستند. و داده‌های x مربوط به مطالعه متابولیسم در دختران ۱۵ ساله است که داده‌های آن میزان انرژی مصرف‌شده طی ۲۴ ساعت است:

```
> x=c(91,504,557,609,693,727,764,803,857,929,970,1043, 089,1195,1384,1713)
```

توزیع پسین به صورت زیر خواهد بود:

$$\pi(\theta, \sigma^2|x) \propto \frac{1}{(\sigma^2)^{n/2}} e^{-\sum_i (x_i-\theta)^2/2\sigma^2} \times \left[\frac{1}{\tau} e^{-(\theta-\theta_0)^2/2\tau^2}\right] \times \left[\frac{1}{(\sigma^2)^{a+1}} e^{1/b\sigma^2}\right]$$

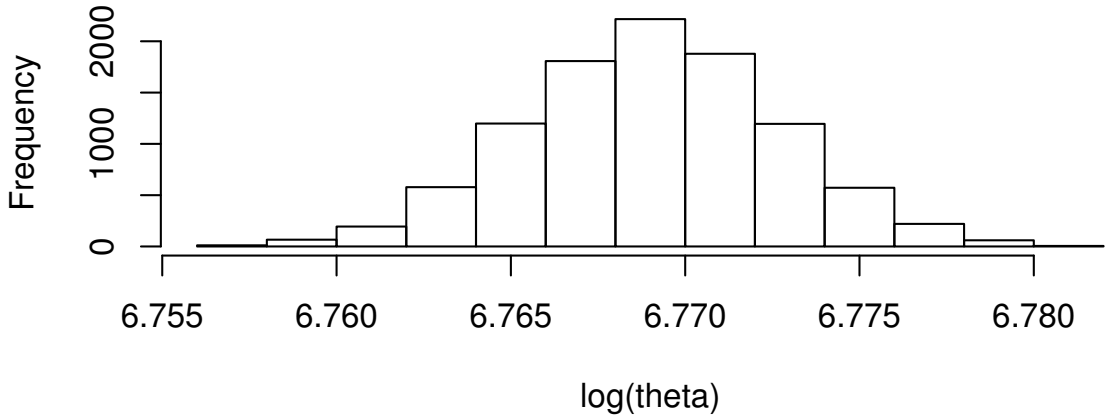
مثال: مدل نرمال

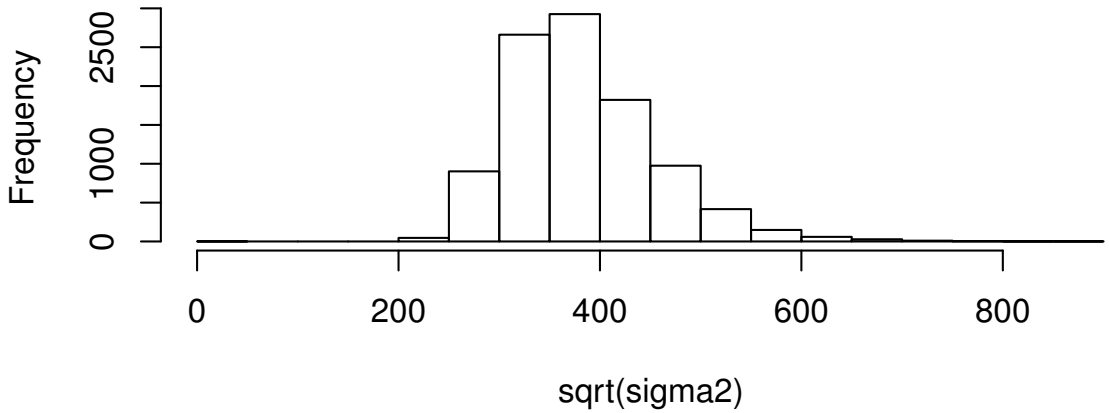
چگالی‌های شرطی کامل به صورت زیر قابل محاسبه‌اند (به عنوان تمرین نشان دهید):

$$\begin{aligned}
\theta|x,\sigma^2 &\sim N\left(\frac{\sigma^2}{\sigma^2+n\tau^2}\theta_0+\frac{n\tau^2}{\sigma^2+n\tau^2}\bar{x},\frac{\sigma^2\tau^2}{\sigma^2+n\tau^2}\right)\\
\sigma^2|x,\theta &\sim IG\left(\frac{n}{2}+a,\frac{1}{2}\sum_i(x_i-\theta)^2+b\right)
\end{aligned}$$

```
n=length(x)
Nsim=10000
a=b=0.1
tau2=10
theta0=xbar
xbar=mean(x)
sh1=(n/2)+a
sigma=theta=rep(0,Nsim) #init arrays
sigma[1]=1/rgamma(1,shape=a,rate=b) #init chains
B=sigma2[1]/(sigma2[1]+n*tau2)
theta[1]=rnorm(1,m=B*theta0+(1-B)*xbar,sd=sqrt(tau2*B))
for (i in 2:Nsim){
     B=sigma2[i-1]/(sigma2[i-1]+n*tau2)
     theta[i]=rnorm(1,m=B*theta0+(1-B)*xbar,sd=sqrt(tau2*B))
     ra1=(1/2)*(sum((x-theta[i])^2))+b
     sigma2[i]=1/rgamma(1,shape=sh1,rate=ra1)
}
##
> mean(theta)
[1] 870.459
> sqrt(mean(sigma2))
[1] 389.7115
##
par(mfrow=c(2,1))
hist(log(theta),main="")
hist(sqrt(sigma2),main="")
```

مثال: هیستوگرام





نمونه‌گیری گیبز چندمرحله‌ای

آنچه که بیان شد، معروف به الگوریتم نمونه‌گیر گیبز دومرحله‌ای است و به سادگی می‌توان آن را به حالت چندمرحله‌ای تعمیم داد.

فرض کنید $X = (X_1, \ldots, X_p)$ و فرض کنید بتوان از چگالی‌های شرطی کامل به سادگی نمونه تولید کرد:

$$X_i | x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p \sim f(x_i | x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p)$$

الگوریتم نمونه‌گیر گیبز چندمرحله‌ای به صورت زیر قابل نمایش است:

- فرض کنید در تکرار $t = 1, 2, \ldots$، بردار $x^{(t)} = (x_1^{(t)}, \ldots, x_p^{(t)})$ مفروض باشد. در این صورت مقادیر زیر را تولید کن
- $X_1^{(t+1)} \sim f_1(x_1 | x_2^{(t)}, \ldots, x_p^{(t)})$
- $X_2^{(t+1)} \sim f_2(x_2 | x_1^{(t+1)}, x_3^{(t)}, \ldots, x_p^{(t)})$
- $\vdots$
- $X_p^{(t+1)} \sim f_p(x_p | x_1^{(t+1)}, \ldots, x_{p-1}^{(t+1)})$

محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۲

جلسه سیزدهم: روش‌های عددی و محاسبات آماری

حسین باغیشنی

دانشگاه شاهرود

۱۵ آذر ۱۳۹۲

مقدمه

قسمت‌هایی از مسایل استنباط آماری منتهی می‌شوند به پیدا کردن ریشه یک معادله:

- می‌نیمم‌سازی یک تابع
- یافتن برآوردگرهای ماکسیمم درستنمایی

این گونه مسایل، حل یک معادله یا یک دستگاه معادلات را شامل می‌شوند که حل آن‌ها به یک شکل تحلیلی منتهی نمی‌شود.

در چنین مواردی، گریزی از یافتن جواب‌های تقریبی برای آن‌ها نیست.

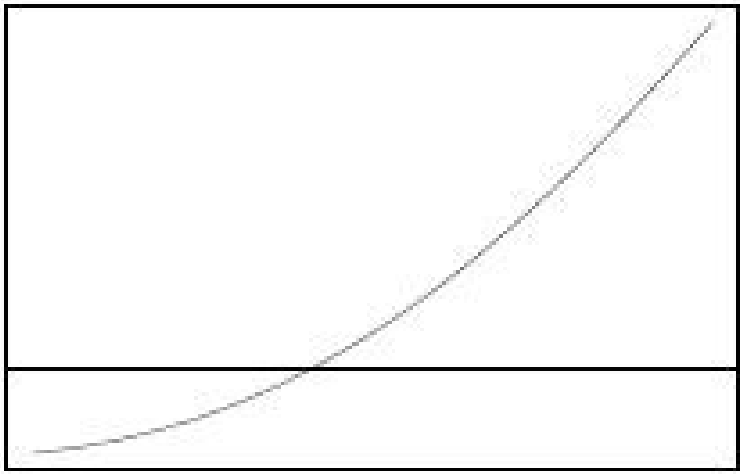
یک رهیافت برای حصول جواب‌های تقریبی، حل عددی معادله‌ها با روش‌های عددی است.

در این جلسه، مروری کوتاه بر برخی از روش‌های عددی پر استفاده در آمار خواهیم داشت.

پیدا کردن ریشه تابع

بیان مساله: برای تابع مفروض $f : \Re^k \longrightarrow \Re^d$، $\tilde{x}$ را طوری پیدا کنید که

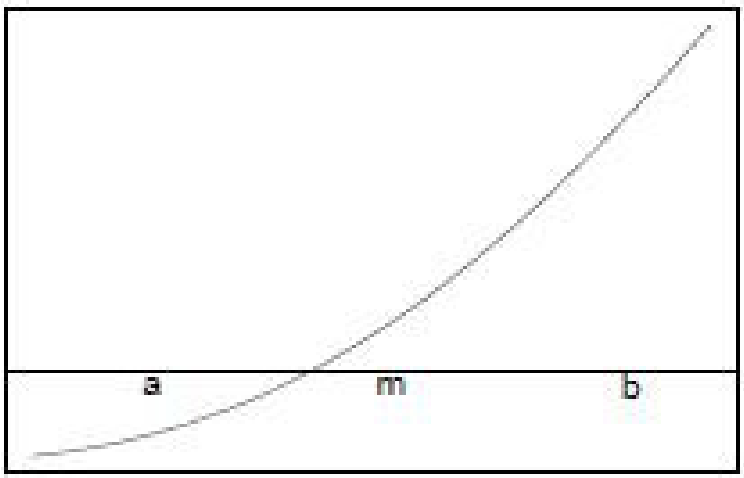
$$f(\tilde{x}) = ۰$$

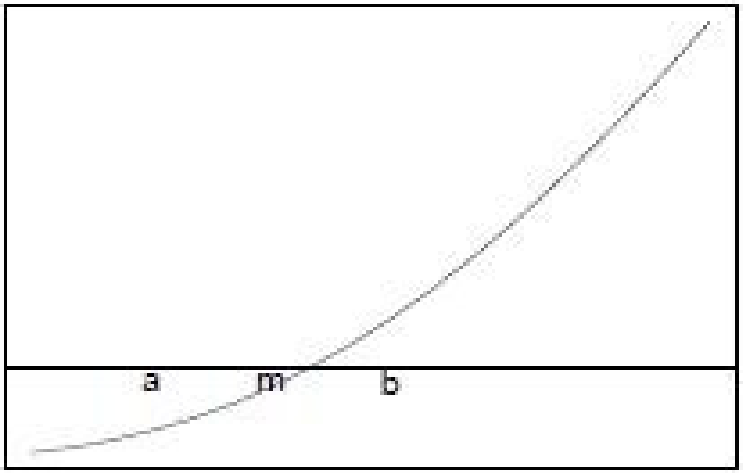


روش‌ها:

- دوبخشی *Bisection*
- نیوتون ـ رافسون
- سکانت
- درون‌یابی درجه دوم معکوس
- برنت *Brent*

روش دوبخشی





این روش بر اساس قضیه مقدار میانی، بنا شده است.

با مفروض بودن a و b که $f(a) < 0$ و $f(b) > 0$، مادامی که $|a - b| > \epsilon$:

- قرار بده $m = \frac{a+b}{2}$
- اگر $f(m) < 0$ آنگاه $a = m$، در غیر این صورت $b = m$

روش دوبخشی: ویژگی‌ها

- در هر مرحله طول فاصله‌ای که شامل ریشه است، نصف می‌شود
- این روش مطمئنا همگرا خواهد شد اما سرعت همگرایی می‌تواند نسبتا کند باشد. در واقع نرخ همگرایی این روش، خطی است.
- اگر تابع در فاصله $[a, b]$ بیشتر از یک ریشه داشته باشد، این روش یکی از ریشه‌ها را پیدا خواهد کرد.

روش دوبخشی: مثال

هدف، حل تابع

$$a^۲ + y^۲ + \frac{۲ay}{n-۱} = n - ۲$$

است، که در آن a یک ثابت مشخص و $n > ۲$ یک عدد صحیح است.

این معادله جواب بسته دارد که به صورت زیر خواهد بود:

$$y = \frac{-a}{n-۱} \pm \sqrt{n - ۲ + a^۲ + (\frac{a}{n-۱})^۲}.$$

این مثال خوبی است تا عملکرد روش عددی دوبخشی را دریابیم. پس مساله حل معادله زیر است:

$$f(y) = a^۲ + y^۲ + \frac{۲ay}{n-۱} - (n-۲) = ۰.$$

مثال: ادامه

اگر $a = ۱/۲$ و $n = ۲۰$ انتخاب شوند، دو ریشه مثبت و منفی وجود خواهند داشت. فرض کنید فقط دنبال یافتن ریشه مثبت در فاصله شروع $(۰, ۵n)$ باشیم.

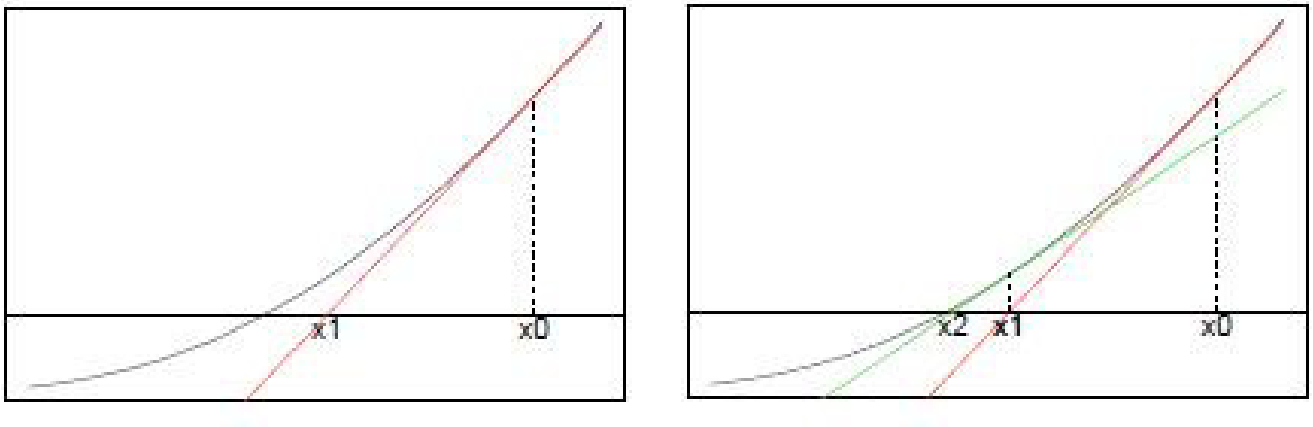
```
f <- function(y, a, n) { a^2 + y^2 + 2*a*y/(n-1) - (n-2) }
a <- 0.5
n <- 20
b0 <- 0
b1 <- 5*n
# solve using bisection
it <- 0
eps <- .Machine$double.eps^0.25
r <- seq(b0, b1, length=3)
y <- c(f(r[1], a, n), f(r[2], a, n), f(r[3], a, n))
if (y[1] * y[3] > 0)
    stop("f does not have opposite sign at endpoints")
while(it < 1000 && abs(y[2]) > eps) {
      it <- it + 1
      if (y[1]*y[2] < 0) {
          r[3] <- r[2]
          y[3] <- y[2]
      } else {
          r[1] <- r[2]
          y[1] <- y[2]
      }
      r[2] <- (r[1] + r[3]) / 2
      y[2] <- f(r[2], a=a, n=n)
      print(c(r[1], y[1], y[3]-y[2]))
}
```

مثال: ادامه

```
> r[2]
[1] 4.186845
> y[2]
[1] 2.984885e-05
> it
[1] 21
```

جواب‌های دقیق عبارتند از $y = ۴/۱۸۶۸۴۱$ و $y = -۴/۲۳۹۴۷۳$.

روش نیوتون ـ رافسون



با داشتن مقدار اولیه x_0، مراحل زیر را تکرار کن:

- خط مماس بر f را در نقطه x_0 رسم کن
- نقطه صفر خط را برابر x_1 قرار بده
- قرار بده $x_1 \longrightarrow x_0$

نیوتون ـ رافسون: ویژگی‌ها

- اگر مقدار اولیه x_0 نزدیک به جواب باشد، سرعت همگرایی آن بالاست
- ممکن است همگرا نشود
- نیازمند مشتق تابع است
- به ابعاد بالاتر نیز قابل تعمیم است
- نسخه‌های سازوارشده متفاوتی از آن معرفی شده است

نیوتون ـ رافسون: معادله به‌روزکننده

خط مماس f در x_0 عبارتست از:

$$f(x_0) + f'(x_0)(x - x_0).$$

نقطه صفر (ریشه) این خط، x_1، در معادله زیر صدق می‌کند:

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0,$$

که منتهی می‌شود به:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

به طور مشابه برای ابعاد بالاتر از یک، خواهیم داشت:

$$x_{i+1} = x_i - f'(x_i)^{-1} f(x_i),$$

که در آن $f'(x)$ ماتریس مشتق‌های جزیی است.

مثال

هدف، پیدا کردن جواب معادله

$$f(x) = \log(x) - \exp\{-x\} = ۰$$

است.

```
ftn <- function(x) {
        fx <- log(x) - exp(-x)
        dfx <- 1/x + exp(-x)
return(c(fx, dfx))
}
```

مثال

```
newtonraphson <- function(ftn, x0, tol = 1e-9, max.iter = 100) {
# Newton_Raphson algorithm for solving ftn(x)[1] == 0
# we assume that ftn is a function of a single variable that returns
# the function value and the first derivative as a vector of length 2
# x0 is the initial guess at the root and the algorithm terminates when the function value is within distance
# tol of 0, or the number of iterations exceeds max.iter
x <- x0
fx <- ftn(x)
iter <- 0
# continue iterating until stopping conditions are met
while ((abs(fx[1]) > tol) && (iter < max.iter)) {
       x <- x - fx[1]/fx[2]
       fx <- ftn(x)
       iter <- iter + 1
       cat("At iteration", iter, "value of x is:", x, "\n")
}
# output depends on success of algorithm
if (abs(fx[1]) > tol) {
    cat("Algorithm failed to converge\n")
    return(NULL)
    } else {
      cat("Algorithm converged\n")
    return(x)
    }
}
> newtonraphson(ftn, 2, 1e-06)
At iteration 1 value of x is: 1.12202
At iteration 2 value of x is: 1.294997
At iteration 3 value of x is: 1.309709
At iteration 4 value of x is: 1.3098
Algorithm converged
[1] 1.3098
```

نیوتون_رافسون: مرتبه همگرایی

فرض کنید $x_0, x_1, x_2, \ldots$ دنباله تقریب‌هایی برای رسیدن به جواب $\tilde{x}$ باشد.

یک روش به طور خطی (در مرتبه اول) همگرا می‌شود، اگر

$$|x_{i+1} - \tilde{x}| \leq C|x_i - \tilde{x}|, \quad C < 1.$$

روش به صورت درجه دو (در مرتبه دوم) همگرا می‌شود، اگر

$$|x_{i+1} - \tilde{x}| \leq K|x_i - \tilde{x}|^2, \quad K \in \Re.$$

یک روش درجه دو خیلی سریعتر از یک روش خطی، همگرا می‌شود.

روش نیوتون_رافسون، یک روش با همگرایی مرتبه دوم است (تمرین)

مثال

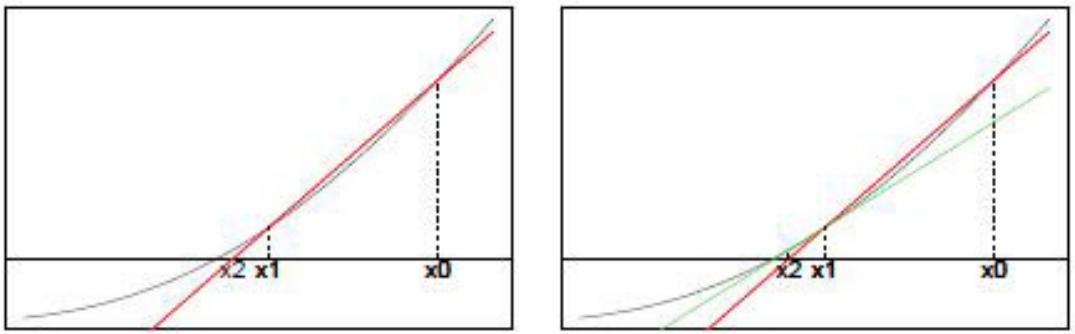
روش دوبخشی

```
> f=function(x){(x-2)*(1+x^2-0.3*(x-1)^3)}
> A=1.4; B=2.9
> for (i in 1:20) {m=(A+B)/2
+ if (f(A)*f(m)>0) A=m else B=m
+ print(A,digits=12)}
[1] 1.4
[1] 1.775
[1] 1.9625
[1] 1.9625
[1] 1.9625
[1] 1.9859375
[1] 1.99765625
[1] 1.99765625
[1] 1.99765625
[1] 1.99912109375
[1] 1.99985351562
[1] 1.99985351562
[1] 1.99985351562
[1] 1.99994506836
[1] 1.99999084473
[1] 1.99999084473
[1] 1.99999084473
[1] 1.99999656677
[1] 1.9999994278
[1] 1.9999994278
```

روش نیوتون–رافسون

```
> f=function(x){(x-2)*(1+x^2-0.3*(x-1)^3)}
> f1=function(x){1+x^2-0.3*(x-1)^3+(x-2)*(2*x-0.9*(x-1)^2)}
> g=function(x){x-f(x)/f1(x)}
> xold=2.9
> for (i in 1:20){ xnew=g(xold); xold=xnew
+ print(xnew,digits=12)}
[1] 2.21416533654
[1] 2.0235925475
[1] 2.00035651707
[1] 2.0000000838
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
[1] 2
```

روش سکانت

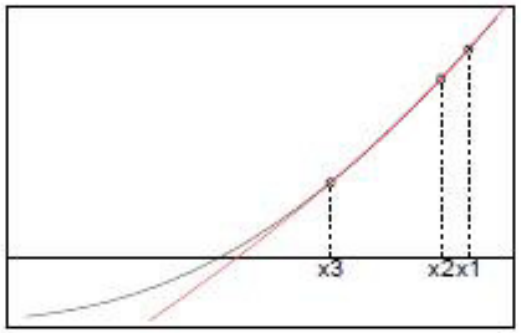


با داشتن مقادیر اولیه x_0 و x_1، مراحل زیر را تکرار کن:

- خطی که از نقاط $(x_0, f(x_0))$ و $(x_1, f(x_1))$، خط سکانت، می‌گذرد را به دست آور
- نقطه صفر خط را برابر x_2 قرار بده
- نقاط را به‌روز کن

در این روش اگر $x_0 \approx x_1$، خیلی شبیه به روش نیوتون–رافسون عمل خواهد کرد.
نیاز به فرمول مشتق ندارد.

روش درون‌یابی درجه دو



با داشتن مقادیر اولیه x_0، x_1 و x_2 مراحل زیر را تکرار کن:

- منحنی که از نقاط $(x_0, f(x_0))$، $(x_1, f(x_1))$ و $(x_2, f(x_2))$ می‌گذرد را به دست آور
- نقطه صفر منحنی را برابر x_3 قرار بده
- نقاط را به‌روز کن

در این روش اگر به جای منحنی درجه دو، یک خط رسم شود، همان روش سکانت خواهد بود. نیاز به فرمول مشتق ندارد.

روش برنت

این روش، یک روش ترکیبی است که سعی می‌کند سرعت روش نیوتون و اطمینان از همگرایی روش دوبخشی را به ارث ببرد.

تکرارهای این روش، بسته به مقدار جاری در هر تکرار، از روش‌های دوبخشی، سکانت یا درون‌یابی درجه دو استفاده می‌کنند.

این روش در R با تابع $uniroot$ اجرا می‌شود.

```
f=function(x){(x-2)*(1+x^2-0.3*(x-1)^3)}
> uniroot(f,c(1,3))
$root
[1] 1.999998
$f.root
[1] -8.559716e-06
$iter
[1] 7
$estim.prec
[1] 6.103516e-05
```

انتگرال‌گیری عددی

به طور کلی یک انتگرال به شکل

$$\int_a^b f(x)\,dx$$

به صورت عددی، به صورت مجموعی به شکل

$$\sum_{i=0}^{n} f(x_i)w_i$$

برآورد می‌شود، که در آن $\{x_i\}$ نقاطی در بازه $[a, b]$ هستند و $\{w_i\}$ وزن‌های مناسبی هستند که در روش‌های مختلف، به طور متفاوتی به دست می‌آیند.

به عنوان مثال، در روش ذوزنقه، فاصله $[a, b]$ به n زیرفاصله مساوی با طول $h = (b - a)/n$ تقسیم می‌شود که در هر زیرفاصله مساحت ذوزنقه حاصل برای برآورد انتگرال در آن زیرفاصله استفاده می‌شود:

$$\frac{f(x_i) + f(x_{i+1})}{h/2}.$$

انتگرال‌گیری عددی

بنابراین برآورد انتگرال در روش ذوزنقه به شکل زیر حاصل می‌شود:

$$\frac{h}{۲}f(a) + h\sum_{i=۱}^{n-۱} f(x_i) + \frac{h}{۲}f(b).$$

تمرین: تابعی در R بنویسید که انتگرال تابعی را در فاصله $[a, b]$ به روش ذوزنقه محاسبه کند.

- روش‌های عددی محاسبه انتگرال، می‌توانند سازوار یا ناسازوار باشند:
  - روش‌های ناسازوار بر روی کل ناحیه انتگرال‌گیری، قاعده وزن‌دهی یکسانی را به کار می‌برند
  - زمانی که تابع زیر انتگرال در قسمتی از ناحیه انتگرال‌گیری خوش‌رفتار است و در قسمتی دیگر چندان خوش‌رفتار نیست، بهتر است در هر قسمت به طور متفاوتی عمل کرد
  - روش‌های سازوار، زیرفاصله‌ها را بر اساس رفتار موضعی تابع انتخاب می‌کنند

روش‌های تربیع‌بندی

روش‌های تربیع‌بندی ($Quadrature$) تابع را در تعداد متناهی از نقاط (موسوم به گره‌ها)، که لزوما در فواصل یکسان قرار ندارند، مقداردهی می‌کنند.

یکی از پرمصرف‌ترین روش‌های تربیع‌بندی، روش تربیع‌بندی گاوس ـ ارمیت است که در آن وزن‌های $\{w_i\}$ توسط چندجمله‌ای‌های متعامد گاوس ـ ارمیتی مشخص می‌شوند.

برای دیدن جزییات این روش می‌توانید به کتاب زیر مراجعه کنید:
Givens, G.H. & Hoeting, J.A. (2005). Computational Statistics. Wiley, New Jersey.

تابع $integrate$ در R، از روش تربیع‌بندی سازوار برای تقریب انتگرال یک تابع تک‌متغیره استفاده می‌کند.

مثال

هدف، محاسبه انتگرال

$$\int_{0}^{\infty} \frac{dy}{(\cosh y - \rho r)^{n-1}},$$

است، که در آن $-1 < \rho < 1$ و $-1 < r < 1$ ثابت هستند و $n \geq 2$ یک عدد صحیح است.

برای مقادیر ثابتی از پارامترها (مثلا $n = 10$، $r = 0/5$ و $\rho = 0/2$) انتگرال به راحتی قابل محاسبه است:

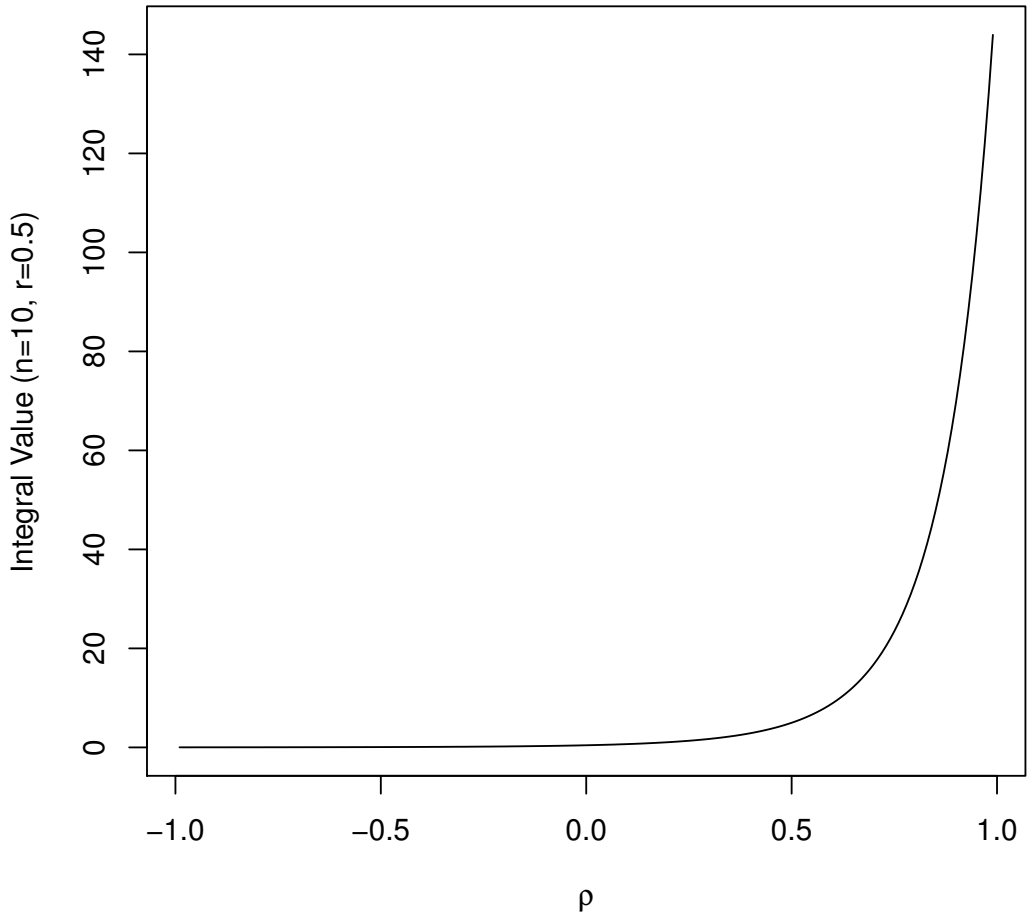
```
> integrate(function(y){(cosh(y)-0.1)^(-9)}, lower=0, upper=Inf)
1.053305 with absolute error < 2.3e-05
```

مثال: ادامه

برای مقادیر دلخواه پارامترها، می‌توان تابع را کلی نوشت:

```
f <- function(y, N, r, rho) {
     (cosh(y) - rho * r)^(1 - N)
}
ro <- seq(-.99, .99, .01)
v <- rep(0, length(ro))
for (i in 1:length(ro)) {
     v[i] <- integrate(f, lower=0, upper=Inf,
               rel.tol=.Machine$double.eps^0.25,
               N=10, r=0.5, rho=ro[i])$value
     }
plot(ro, v, type="l", xlab=expression(rho),
         ylab="Integral Value (n=10, r=0.5)")
```

مثال: ادامه



مثال: توزیع نمونه‌ای ضریب همبستگی

ضریب همبستگی جامعه، ρ، معمولا با ضریب همبستگی نمونه برآورد می‌شود:

$$R = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{[\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2]^{1/2}}$$

اگر نمونه‌های $\{(X_i, Y_i), i = 1, \ldots, n\}$ iid از توزیع نرمال دومتغیره $N_2(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho)$ باشند، چنانچه $\rho = 0$، تابع چگالی R به صورت زیر به دست می‌آید:

$$f(r) = \frac{\Gamma((n-1)/2)}{\Gamma(1/2)\Gamma((n-2)/2)}(1 - r^2)^{(n-4)/2}, \quad -1 < r < 1.$$

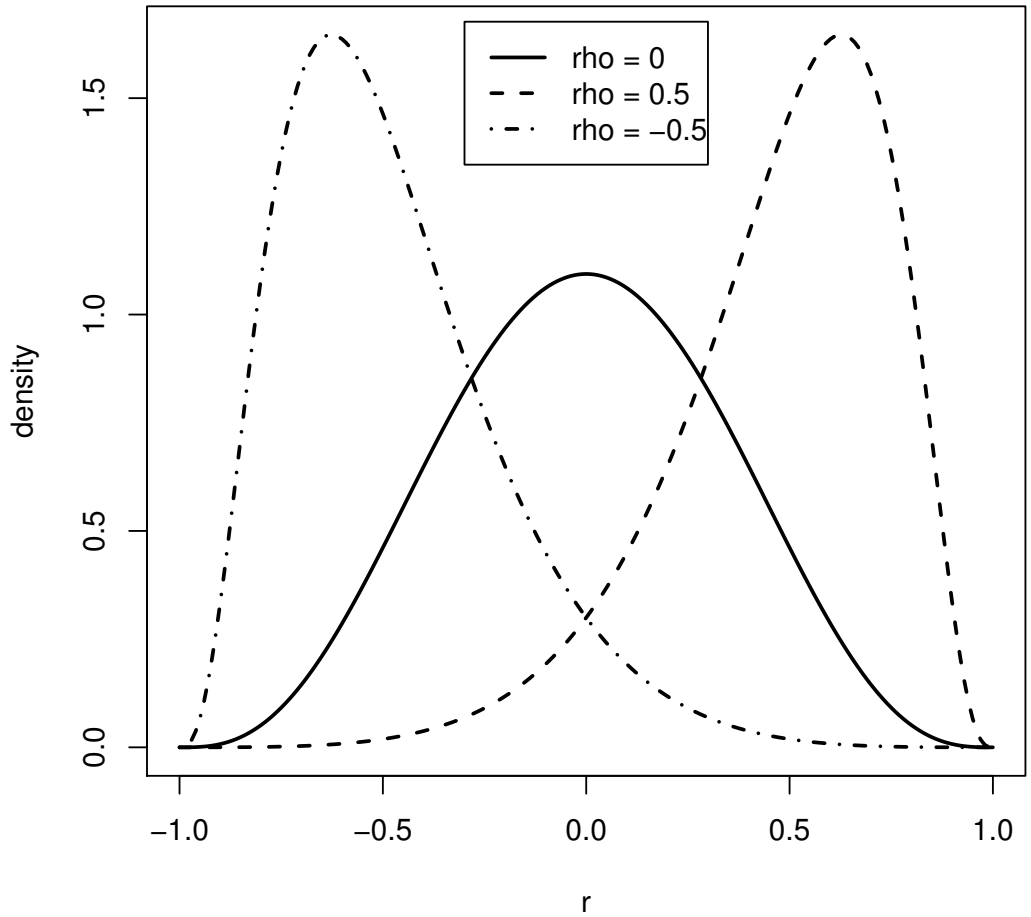
برای $0 < |\rho| < 1$، تابع چگالی پیچیده‌تر می‌شود. یکی از شکل‌های تابع چگالی به صورت زیر ارایه شده است:

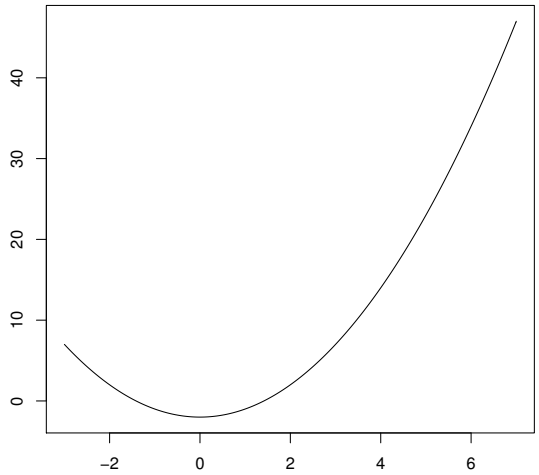
$$f(r) = \frac{(n-2)(1-\rho^2)^{(n-1)/2}(1-r^2)^{(n-4)/2}}{\pi} \int_0^\infty \frac{dw}{(\cosh w - \rho r)^{n-1}}.$$

مثال: ادامه

```
.dcorr <- function(r, N, rho=0) {
# compute the density function of sample correlation
if (abs(r) > 1 || abs(rho) > 1) return (0)
if (N < 4) return (NA)
if (isTRUE(all.equal(rho, 0.0))) {
    a <- exp(lgamma((N - 1)/2) - lgamma((N - 2)/2)) /
         sqrt(pi)
    return (a * (1 - r^2)^((N - 4)/2))
}
# if rho not 0, need to integrate
f <- function(w, R, N, rho)
     (cosh(w) - rho * R)^(1 - N)
# need to insert some error checking here
i <- integrate(f, lower=0, upper=Inf,
               R=r, N=N, rho=rho)$value
     c1 <- (N - 2) * (1 - rho^2)^((N - 1)/2)
     c2 <- (1 - r^2)^((N - 4) / 2) / pi
     return(c1 * c2 * i)
}
r <- as.matrix(seq(-1, 1, .01))
d1 <- apply(r, 1, .dcorr, N=10, rho=.0)
d2 <- apply(r, 1, .dcorr, N=10, rho=.5)
d3 <- apply(r, 1, .dcorr, N=10, rho=-.5)
plot(r, d2, type="l", lty=2, lwd=2, ylab="density")
lines(r, d1, lwd=2)
lines(r, d3, lty=4, lwd=2)
legend("top", inset=.02,
c("rho = 0", "rho = 0.5", "rho = -0.5"), lty=c(1,2,4), lwd=2)
```

مثال: ادامه





پیش‌فرض: از آنجا که ماکسیمم‌سازی یک تابع معادل می‌نیمم‌سازی منهای آن است، بنابراین در این جا فقط به بحث می‌نیمم‌سازی می‌پردازیم.

طرح مساله: برای تابع مفروض $F : \Re^k \longrightarrow \Re$ مقدار $\tilde{x}$ را طوری بیابیم که

$$F(\tilde{x}) = \min_x F(x).$$

می‌نیمم‌سازی

برای می‌نیمم‌سازی بدون قید، $\tilde{x}$ مقدار صفر مشتق تابع است: $F'(\tilde{x}) = 0$.

روش‌ها:

- هر روشی که برای یافتن ریشه تابع به کار می‌رود
- نیوتون ـ رافسون
- شبه‌نیوتون
- نلدر ـ مید
- الگوریتم‌های فراوان دیگر

شبه‌نیوتون *Quasi Newton*

الگوریتم نیوتون‌ـ رافسون برای حل $F'(\tilde{x}) = 0$، از تکرارهای زیر استفاده می‌کند:

$$x_{i+1} = x_i - F''(x_i)^{-1} F'(x_i), \quad F''(x) = (\frac{\partial^2}{\partial x_i x_j} F(x)).$$

یک روش شبه‌نیوتون ماتریس $F''(x_i)$ را با ماتریسی که محاسبه آن ساده‌تر است و الگوریتمی پایدارتر را نتیجه می‌دهد، عوض می‌کند.

این روش در R با توابع nlm و $optim$ (روش $BFGS$) قابل اجراست.

مسایل درستنمایی ماکسیمم

در خیلی از مسایل آماری، علاقه‌مند به استنباط مبتنی بر درستنمایی ماکسیمم (ML) هستیم.

معروف است که برآوردگرهای ML، در صورت وجود، ویژگی‌های مجانبی خوبی مانند سازگاری، کارایی و نرمال مجانبی بودن، دارند. به همین دلیل استفاده از این روش طرفداران زیادی دارد.

فرض کنید مشاهدات $x_1, \ldots, x_n$ از خانواده چگالی‌های $f(x;\theta)$ آمده باشند.

تابع درستنمایی $L(\theta)$ به صورت زیر تعریف می‌شود:

$$L(\theta) = f(x_1, \ldots, x_n; \theta).$$

اگر مشاهدات تشکیل یک نمونه تصادفی ساده بدهند، آنگاه:

$$L(\theta) = \prod_{i=1}^{n} f(x_i; \theta).$$

برآورد ML مقداری از θ مانند $\hat{\theta}$ است که $L(\theta)$ را ماکسیمم می‌کند.

درستنمایی ماکسیمم

بنابراین یک کلاس مهم از مسایل بهینه‌سازی، در آمار، مسایل مبتنی بر درستنمای ماکسیمم است.

دقت کنید که ماکسیمم‌سازی $L(\theta)$ معادل است با ماکسیمم‌سازی $\ell(\theta) = \log L(\theta)$.

اگر تابع چگالی $f(x;\theta)$ هموار باشد، MLE از حل معادله

$$\sum_{i=1}^{n} \dot{\ell}_\theta(x_i) = 0,$$

به دست می‌آید، که در آن $\dot{\ell}_\theta$ تابع امتیاز است:

$$\dot{\ell}_\theta(x) = \frac{\partial}{\partial\theta} \log f(x;\theta).$$

امتیازدهی فیشر *Fisher Scoring*

تکرارهای نیوتون–رافسون عبارتند از:

$$\theta_{i+1} = \theta_i - \frac{\sum_{i=1}^{n} \dot{\ell}_{\theta_i}(x_i)}{\sum_{i=1}^{n} \ddot{\ell}_{\theta_i}(x_i)},$$

که در آن

$$\ddot{\ell}_\theta(x) = \frac{\partial^2}{\partial \theta^2} \log f(x;\theta).$$

ماتریس هسیان $-\sum_{i=1}^{n} \ddot{\ell}_\theta(x_i)$ ماتریس اطلاع مشاهده‌شده نیز نامیده می‌شود و در رابطه زیر صدق می‌کند:

$$-\mathbb{E}_\theta \sum_{i=1}^{n} \ddot{\ell}_\theta(X_i) = nI_\theta,$$

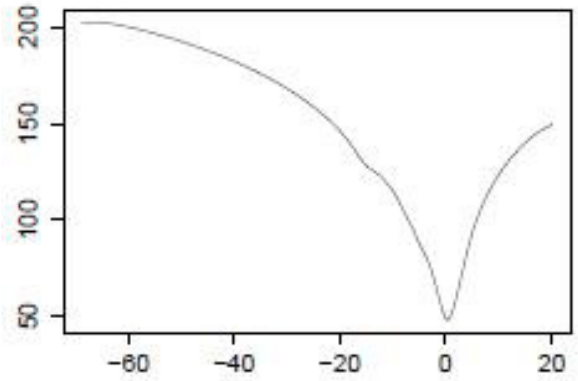
که در آن I_θ ماتریس اطلاع فیشر می‌باشد. تکرارهایی که به جای ماتریس هسیان، از ماتریس اطلاع فیشر nI_θ استفاده می‌کنند، امتیازدهی فیشر نامیده می‌شوند.

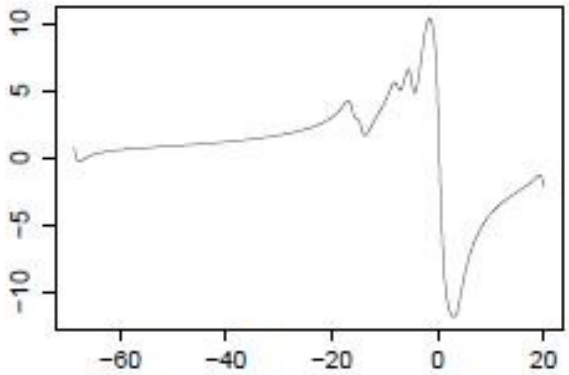
ریشه عکس ماتریس $nI_{\hat{\theta}}$ برآوردی است برای خطای استاندارد MLE.

مثال: توزیع کوشی

برای توزیع کوشی با پارامتر مکان θ، بر اساس یک نمونه به دنبال محاسبه MLE پارامتر هستیم.
توابع درستنمایی و امتیاز به ترتیب متناسب هستند با:

$$\prod_{i=۱}^{n} \frac{۱}{۲\pi} \frac{۱}{۱+(x_i-\theta)^۲}, \quad \sum_{i=۱}^{n} \frac{۲(x_i-\theta)}{۱+(x_i-\theta)^۲}.$$



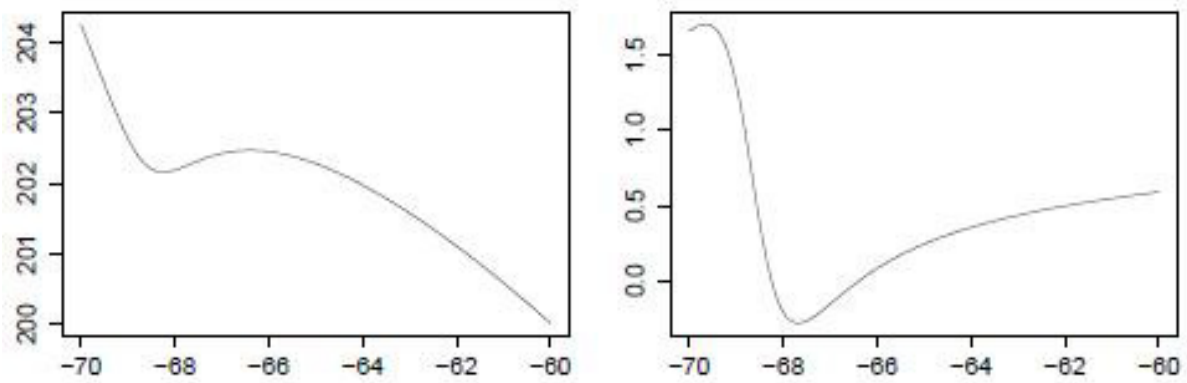


مثال: ادامه

```
set.seed(1244); x=rcauchy(25)
q=seq(min(x),max(x),by=0.01)
minloglik=function(q,x){apply(log(1+outer(x,q,"-")^2),2,sum)}
minloglik1=function(q,x){
     v=outer(x,q,"-"); 2*apply(v/(1+v^2),2,sum)}
> uniroot(minloglik1,median(x)+c(-3,3),x=x)
$root
[1] 0.3691374
> optim(median(x),method="BFGS",minloglik,x=x,hessian=TRUE)
$par
[1] 0.369137
$hessian
         [,1]
[1,] 12.99744 # estimated se is 1/sqrt(12.99744)
> optim(median(x),lower=median(x)-3,upper=median(x)+3,
+ method="Brent",minloglik,x=x,hessian=TRUE)
$par
[1] 0.3691371
```

مثال: می‌نیمم موضعی (نسبی)

```
> optim(-70,method="BFGS",minloglik,x=x)
$par
[1] -68.24447
$convergence
[1] 0
> uniroot(minloglik1,c(-68,-64),x=x)
$root
[1] -66.39614
$iter
[1] 5
$estim.prec
[1] 6.103516e-05
```



در مثال قبلی، فرض کنید توزیع کوشی دارای دو پارامتر مکان θ و مقیاس σ باشد.
تابع درستنمایی در این حالت عبارتست از:

$$\prod_{i=۱}^{n} \frac{۱}{۲\pi\sigma} \frac{۱}{۱ + (x_i - \theta)^۲/\sigma^۲}.$$

بهینه‌سازی چندمتغیره: ادامه

```
set.seed(1244); x=rcauchy(25)
minloglik=function(par,x){q=par[1]; s=par[2];
length(x)*log(s)+apply(log(1+outer(x,q,"-")^2/s^2),2,sum)}
$par
[1] 0.3668736 1.2556457
$value
[1] 48.00677
> OP=optim(c(median(x),mad(x)),method="BFGS",minloglik,x=x,hessian=TRUE); OP # Quasi Newton
$par
[1] 0.3670376 1.2552827
$value
[1] 48.00677
$hessian
          [,1]      [,2]
[1,] 9.6044413 0.2627547
[2,] 0.2627547 6.2611816
> sqrt(diag(solve(OP$hessian)))
[1] 0.3228594 0.3998723
```

چند نکته

- MLE لزوما یکتا نیست.
- برای حصول اطمینان نسبی از گیر نکردن در اکسترمم‌های موضعی، الگوریتم را با چند مقدار اولیه متفاوت اجرا و جواب‌ها را با هم مقایسه کنید.
- تابع $optim$ روش‌هایی را برای بهینه‌سازی مقید، مانند روش $L-BFGS-B$، هم دارد.
- روش پیش‌فرض تابع $optim$ روش نلدر ـ مید است.
- یک روش پرمصرف دیگر که در تابع $optim$ وجود دارد، روش نوردیدن شبیه‌سازی شده $Simulated\ Annealing$ است.
- توابع و بسته‌های مختلف دیگری در R وجود دارند که برای بهینه‌سازی، به طور کلی، و محاسبه MLE، به طور خاص، استفاده می‌شوند.