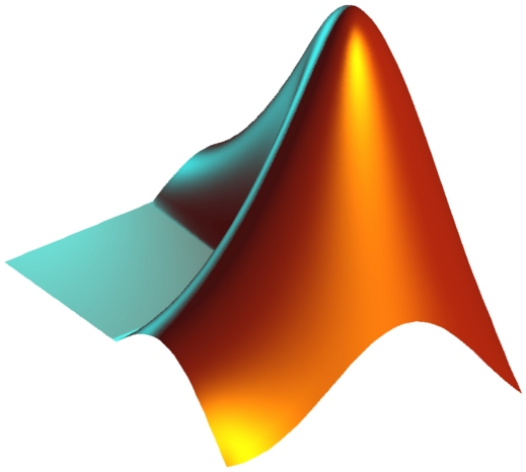
الفبای متلب

سمینار آنالیز عددی پیشرفته

میثم پورگنجی*

پائیز ۱۳۹۱

*E-mail: p.g.meysam@gmail.com



آ

نرم‌افزار
MATLAB®

آ.۱) تاریخچه متلب

Cleve Moler، مدیر دانشکده علوم کامپیوتر دانشگاه نیومکزیکو، در ۱۹۷۰ با هدف ایجاد یک رابط کاربری تعاملی برای EISPACK[۱] و LINPACK[۲] برای دانشجویان، توسعه MATLAB®[۳] را آغاز نمود. در ۱۹۸۴، Moler به همراه Jack Little و Steve Bangert متلب را به زبان C بازنویسی کردند و شرکت MathWorks را برای ادامه روند توسعه آن تاسیس نمودند.
متلب در ابتدا توسط مهندسین در رشته مهندسی کنترل استفاده شد اما دامنه استفاده از متلب سریعا به سایر بخش‌های علوم نیز رسید. امروزه می‌توان متلب را به عنوان ابزاری آموزشی در مبحث جبر خطی یا آنالیز عددی در دانشگاه‌ها و یا به خاطر وجود جعبه‌ابزار قدرتمندش، آنرا به عنوان ابزاری محاسباتی در دست مهندسین و محققان یافت.
متلب اکنون تبدیل به یک زبان تفسیری با دستور زبانی مشابه با C و HPF[۴] شده و ماتریس‌محوری قابل انعطاف آن، متلب را به یک زبان بسیار مناسب برای برنامه‌نویسی مسائل ماتریسی، تبدیل نموده است. واژه MATLAB اختصاری از Matrix Laboratory می‌باشد که خود شاهدی بر مدعای آخر است.

---

[۱] EISPACK یک کتابخانه نرم‌افزاری برای محاسبات عددی مربوط به مقادیر ویژه و بردارهای ویژه‌ی ماتریس‌ها می‌باشد و به زبان FORTRAN نوشته شده است.
[۲] LINPACK یک کتابخانه‌ی نرم‌افزاری برای جبر خطی عددی است که به زبان FORTRAN نوشته شده است.
[۳] در این نوشتار، از این به بعد از کلمه متلب برای ارجاع به نرم‌افزار MATLAB® استفاده خواهیم نمود.
[۴] HPF یا High Performance Fortran یک افزونه زبان برنامه‌نویسی FORTRAN است که امکان محاسبات موازی (Parallel computing) را به آن اضافه می‌کند.

آ.۲) ویژگی‌های متلب

آ.۲.۱) چندسکویی بودن:

نرم‌افزار متلب در سیستم‌عامل‌های ویندوز، مکینتاش و گنو/لینوکس قابل اجرا می‌باشد و شما می‌توانید کدهای نوشته شده به این زبان را، بدون هیچ تغییری در سیستم‌عامل‌های دیگر اجرا نمائید.

آ.۲.۲) جعبه‌ابزار قدرتمند:

بعد از نصب هر نسخه متلب، به صورت پیشفرض ابزارهای به همراه متلب برای شما نصب خواهد شد. به این مجموعه از ابزارها که معمولا از توابع ازپیش‌نوشته تشکیل شده است، جعبه‌ابزار (Toolbox) می‌گویند.

آ.۲.۳) مستندات کامل:

در بخش (ب.۲) بررسی خواهد شد.

آ.۲.۴) تعامل با سایر زبان‌های برنامه‌نویسی:

متلب این امکان را در اختیار شما قرار می‌دهد تا از کدهای نوشته شده به سایر زبان‌های برنامه‌نویسی در درون برنامه خود استفاده کنید. این کار سرعت رسیدن به نتیجه را چندین برابر (گاهی تا ۳۰۰ برابر) افزایش خواهد داد. برای دیدن یک نمونه به http://blogs.mathworks.com/loren/2011/07/18/a-mandelbrot-set-on-the-gpu مراجعه نمائید.

آ.۲.۵) جامعه کاربری گسترده:

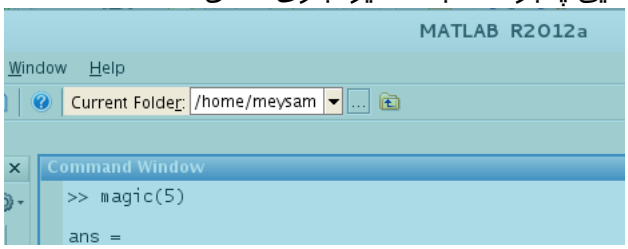
در این مجال، متلب را به عنوان یک نرم‌افزار در راستای اهداف پژوهشی-آموزشی در نظر گرفته‌ایم. با این دید، متلب در بین کارهای تحقیقی پژوهشی نمایه شده در http://citeseerx.ist.psu.edu پربسامدترین نرم‌افزار ریاضی می‌باشد. برای اثبات این مدعا، سه کلمه کلیدی Maple، Mathematica و Matlab را در citeseerx جستجو نمودیم. نتایج جستجو به ترتیب ۷۷۶۵ ورودی، ۷۳۸۱ ورودی و ۳۲۷۱۹ ورودی برای نرم‌افزارهای مختلف ریاضی ذکر شده در بالا را نشان می‌داد و این خود دلیلی بر استقبال جامعه ریاضی‌زبان از نرم‌افزار متلب می‌باشد.

ب محیط متلب

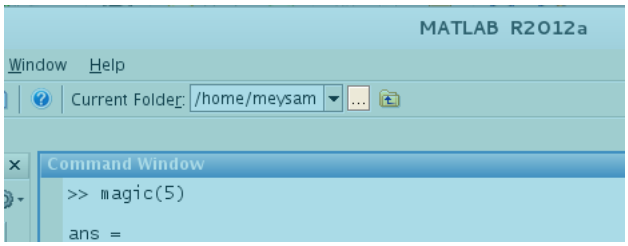
ب.۱) منو File

ب.۱.۱) پوشه جاری متلب:

در صورتی که بخواهید فایلی را به صورت شناور ذخیره یا بازخوانی نمائید، فایل باید در پوشه جاری[۵] متلب قرار داشته باشد. در محیط متلب، به صورت معمول در قسمت بالایی پنجره متلب، مسیر جاری نشان داده شده است:

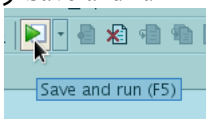


برای تغییر مسیر جاری می‌توان از دکمه Browse for folder در همان موقعیت استفاده نمود:



ب.۱.۲) File → New → Script:

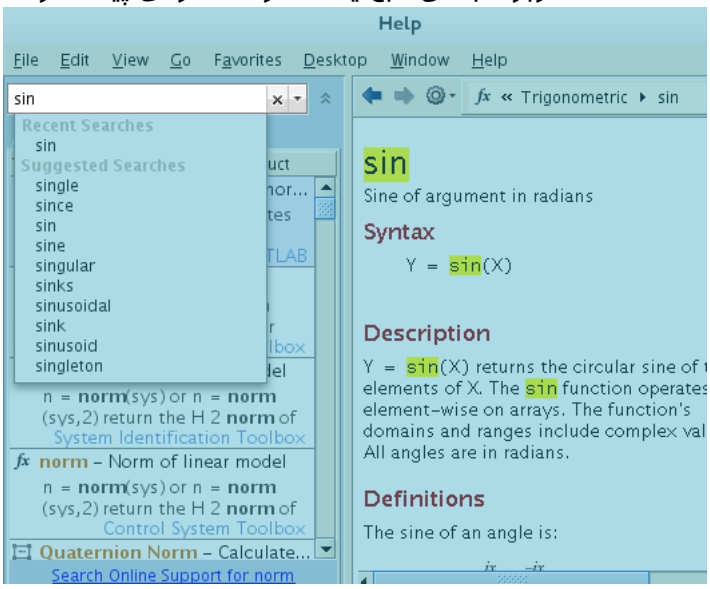
برای راحتی در کار و استفاده از محیط مناسب به همراه رنگ‌آمیزی کد[۶] می‌توان از مسیر بالا استفاده نمود. همچنین در صورتی که به صورت مستقیم در جعبه فرمان متلب کد وارد شود، برای اجتناب از بازنویسی کد در صورت نیاز و یا وقوع اشتباه، از ویرایشگر متلب، ذکر شده در بالا استفاده می‌شود.
برای ذخیره و اجرای کد از دکمه Save and run و یا میانبر F5 استفاده می‌شود:



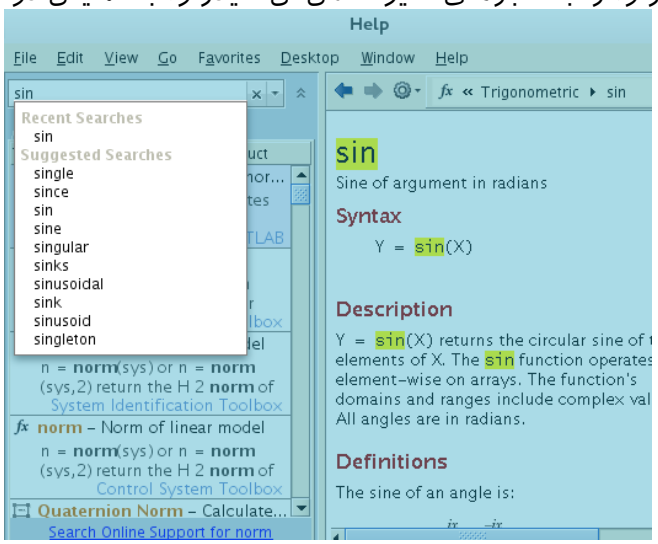
[۵] Current Folder
[۶] Syntax Highlighting

ب.۲) منو Help

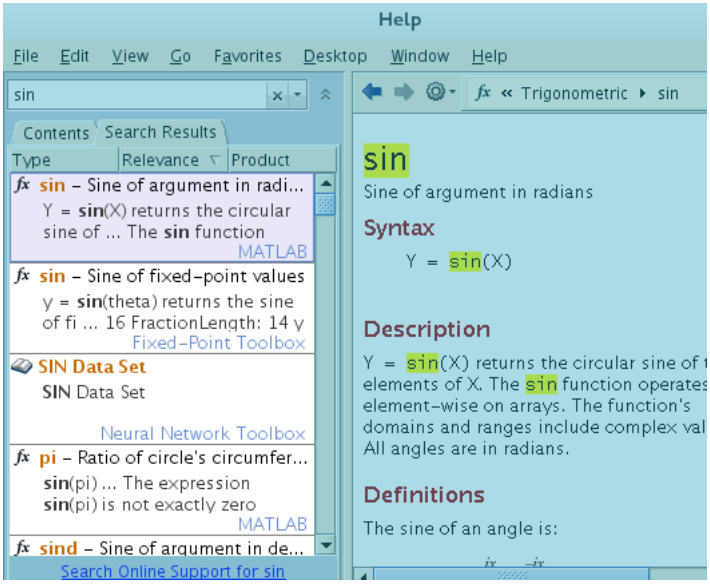
در صورتی که بخواهید در مورد تابع و یا عملکردی از آن اطلاعاتی کسب نمائید، می‌توانید از مسیر Help → Product help پنجره مستندات متلب را باز نمائید.
بعد از باز شدن پنجره مستندات، با وارد نمودن کلید جستجو در کادر سمت چپ می‌توان به مستندات مربوط به آن تابع یا عملکرد دسترسی پیدا نمود:



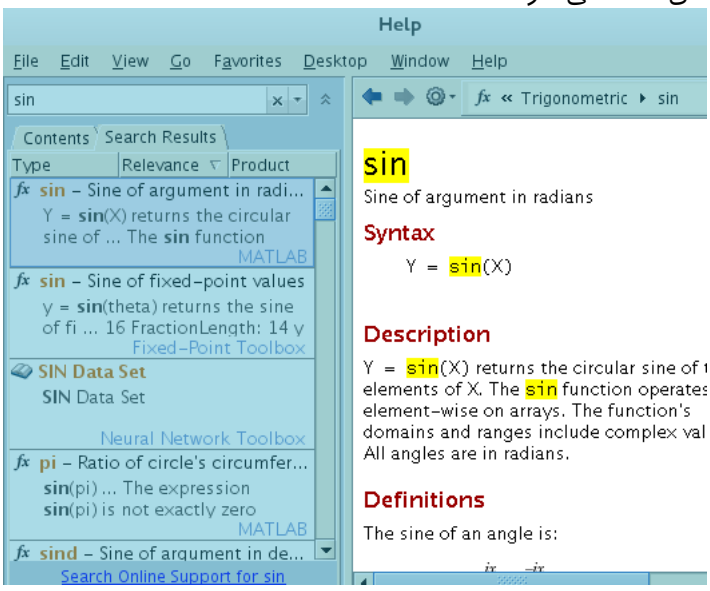
در هنگام وارد نمودن کلیدواژه جستجو، به صورت خودکار کلمات پیشنهادی شامل کلیدواژه و جستجوهای اخیر شامل آن کلیدواژه به نمایش در خواهد آمد:



بعد از تائید کلیدواژه جستجو، موضوعات شامل کلیدواژه، در کادر سمت چپ نشان داده خواهند شد:

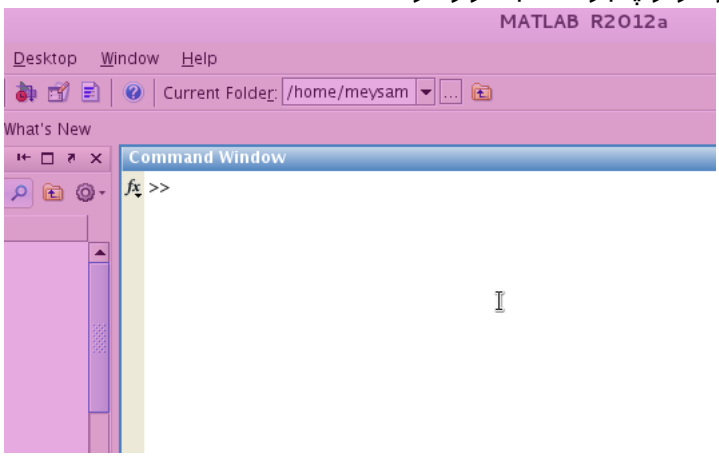


با کلیک بر روی هر موضوع پیشنهادی، مستندات مرتبط با آن، در کادر سمت راست نشان داده می‌شود:

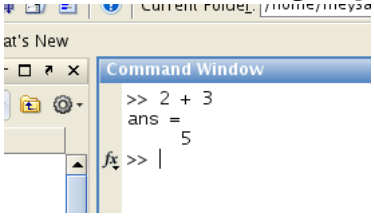


ب.۳) پنجره فرمان

اکثر دستورات نرم‌افزار متلب، در محیطی به نام پنجره فرمان[۷] وارد می‌شود. این پنجره در مرکز پنجره متلب قرار گرفته است:



در متلب فرمان دلخواه، جلوی علامت fx نوشته شده و با فشردن کلید Enter فرمان اجرا می‌شود. برای مثال:



پس اجرای دستور و رسیدن به نتیجه، متلب به خط جدید رفته و منتظر دریافت فرمان جدید خواهد ماند.

[۷] Command Window

پ

متغیرها و اعمال حسابی

پ.۱) اعمال حسابی

اعمال حسابی چهارگانه (/ * - +) در متلب، مانند اکثر زبان‌های برنامه‌نویسی استاندارد دیگر می‌باشد، اما با توجه به ماهیت ماتریسی متلب، اعمال ویژه ماتریسی نیز در آن تعریف می‌شوند. این اعمال در لیست زیر آمده است:

- + : از عملگر + برای جمع استفاده می‌شود. برای مثال 5 = 3 + 2. در ماتریس‌های با اندازه یکسان، درایه‌های متناظر با هم جمع خواهند شد. برای مثال:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} + \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}6 & 8\\10 & 12\end{bmatrix}$$

- - : از عملگر - برای تفریق استفاده می‌شود. برای مثال 1- = 3 - 2. در ماتریس‌های با اندازه یکسان، درایه‌های متناظر از هم کم خواهند شد. برای مثال:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} - \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}-4 & -4\\-4 & -4\end{bmatrix}$$

- * : از عملگر * برای ضرب استفاده می‌شود. برای مثال 6 = 3 * 2. برای ضرب دو ماتریس، اگر شرایط ضرب جبری دو ماتریس وجود داشته باشد (تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر و مساوی با n باشند) ضرب به صورت زیر خواهد بود:

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j} \qquad (۱)$$

برای مثال:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} * \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}19 & 22\\43 & 50\end{bmatrix}$$

- *. : از عملگر *. در ماتریس‌های با اندازه یکسان، برای ضرب درایه به درایه استفاده می‌شود. به بیان دقیق‌تر:

$$C_{i,j} = A_{i,j} B_{i,j} \qquad (۲)$$

برای مثال:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} .* \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}5 & 12\\21 & 32\end{bmatrix}$$

- / : از عملگر / برای تقسیم استفاده می‌شود. برای مثال: 1.75 = 4 / 7.

- ∧ : ∧ عملگر توان می‌باشد. برای مثال: $2 \wedge 3 = 8$. توان nم یک ماتریس مربعی، با رابطه‌ی بازگشتی زیر تعریف می‌شود:

$$A^n = \begin{cases} (A^{n-1}) * A & n > 1 \\ A & n = 1 \end{cases} \quad (۳)$$

برای مثال:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \wedge 3 = \begin{bmatrix} 37 & 54 \\ 81 & 118 \end{bmatrix}$$

- .∧ : در ماتریس‌های با اندازه یکسان، عملگر .∧ از درایه‌ی ماتریس اول به عنوان پایه و از درایه متناظر در ماتریس دوم به عنوان نما استفاده خواهد نمود. به بیان دقیق‌تر:

$$C_{i,j} = A_{i,j}^{B_{i,j}} \quad (۴)$$

برای مثال:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .\wedge \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 64 \\ 21187 & 65536 \end{bmatrix}$$

پ.۲) توابع ریاضی پراستفاده

در لیست زیر، تعدادی از توابع ریاضی پراستفاده معرفی خواهند شد.

- abs() : تابع abs() قدر مطلق عدد را باز میگرداند. برای مثال abs(-2.3) = 2.3. در ماتریس‌ها، این تابع قدرمطلق درایه‌ها را باز می‌گرداند. برای مثال:

$$abs(\begin{bmatrix} 1 & 2 \\ -3 & 4 \end{bmatrix}) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- sin() : تابع sin() مقدار تابع مثلثاتی سینوس داده ورودی را بازمی‌گرداند. برای مثال: sin(0) = 0. این تابع در ماتریس‌ها، درایه‌وار عمل خواهد نمود. برای مثال:

$$sin(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}) = \begin{bmatrix} 0.8415 & 0.9093 \\ 0.1411 & -0.7568 \end{bmatrix}$$

توابع مثلثاتی کسینوس و تانژانت در متلب، به ترتیب cos() و tan() می‌باشند و اعمال آنها مشابه تابع sin() خواهد بود.

- asin() : تابع asin() تابع معکوس sin() می‌باشد. توابع متناظر برای cos() و tan() به ترتیب acos() و atan() می‌باشند. اعمال این توابع بروی اعداد و ماتریس‌ها مشابه با تابع sin() می‌باشد.

- log() : تابع log() مقدار لگاریتم طبیعی (لگاریتم عدد در مبنای e) را باز می‌گرداند. برای مثال: $log(5) = 1.6094$. برای ماتریس‌ها، تابع log() درایه‌وار عمل خواهد نمود. برای مثال:

$$log(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}) = \begin{bmatrix} 0 & 0.6931 \\ 1.0986 & 1.3863 \end{bmatrix}$$

از تابع log10() به صورت مشابه بالا، برای بدست آوردن لگاریتم در مبنای ۱۰ استفاده می‌شود.

- exp() : تابع exp(x) مقدار e^x را بازمی‌گرداند. برای مثال: $exp(10) = 2.2026e+04 = 2.2026\times10^4$ که در آن $2.2026e+04$. عملکرد تابع بر ماتریس‌ها، درایه‌وار خواهد بود. برای مثال:

$$exp(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}) = \begin{bmatrix}2.7183 & 7.3891\\ 20.0855 & 54.5982\end{bmatrix}$$

- sqrt() : تابع sqrt(x) مقدار $\sqrt{x}$ را بازمی‌گرداند. برای مثال: sqrt(2) = 1.4142. این تابع بر ماتریس‌ها به صورت درایه‌وار عمل خواهد نمود. برای مثال:

$$sqrt(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}) = \begin{bmatrix}1.0000 & 1.4142\\ 1.7321 & 2.0000\end{bmatrix}$$

- floor() : تابع floor() مقدار جز صحیح پایین را بازمی‌گرداند. برای مثال: $floor(1.9) = 1$. این تابع بروی ماتریس‌ها به صورت درایه‌وار عمل می‌نماید. برای مثال:

$$floor(\begin{bmatrix}1.2 & -1.2\\ 1.9 & -1.9\end{bmatrix}) = \begin{bmatrix}1 & -2\\ 1 & -2\end{bmatrix}$$

به صورت مشابه از تابع ceil() برای بدست آوردن جز صحیح بالای عدد استفاده می‌شود. برای مثال:

$$ceil(\begin{bmatrix}1.2 & -1.2\\ 1.9 & -1.9\end{bmatrix}) = \begin{bmatrix}2 & -1\\ 2 & -1\end{bmatrix}$$

پ.۳) متغیر، بردار، ماتریس

در متلب هر داده به صورت ماتریس ذخیره می‌شود. برای تعریف یک متغیر و ذخیره داده، راه‌های متفاوتی وجود دارد که در ادامه به آن می‌پردازیم.

پ.۱.۳) تعریف با استفاده از مقداردهی:

در متلب با انتساب یک مقدار می‌توان متغیر تعریف نمود. برای مثال در خط زیر، مقدار ۲ را در متغیر x ذخیره می‌نمائیم:

```
x = 2;
```

برای تعریف یک ماتریس از فرم [,;,] استفاده می‌شود که , جداگر ستون‌ها و ; جداگر سطرها می‌باشد. به عنوان مثال:

```
c = [1,2,3;4,5,6];
```

که ماتریس زیر را تعریف می‌نماید:

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$$

برای راحتی کار می‌توان به جای استفاده از , به عنوان جداگر ستون‌ها، از فضای خالی (Blank Space) استفاده نمود. به عنوان مثال، دستور زیر، نتیجه‌ای شبیه به بالا را رقم خواهد زد:

```
c = [1 2 3;4 5 6];
```

همچنین می‌توان به جای استفاده از ; به عنوان جداگر سطرها، از شیوه‌ی تعریف زیر برای تعریف یک ماتریس استفاده نمود:

```
c = [1 2 3
4 5 6];
```

پ.۲.۳) تعریف با استفاده از متدها:

برای تعریف برخی از ماتریس‌های معروف، می‌توان از متدهای متلب استفاده نمود. لیستی از این متدها رو زیر آمده است:

- ماتریس صفر: برای تعریف ماتریس صفر، می‌توان از متد zeros استفاده نمود. طریقه استفاده از این متد به صورت زیر است:

```
c = zeros(row,column);
```

که در آن row تعداد سطرها و column تعداد ستون‌های ماتریس تعریفی خواهد بود. برای مثال:

```
>> c = zeros(3,4)

c =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

برای تعریف کردن ماتریس مربعی صفر، کافی است تنها یک پارامتر به تابع ارسال شود. برای مثال:

```
>> c = zeros(3)

c =
     0     0     0
     0     0     0
     0     0     0
```

- ماتریس همانی: برای تعریف ماتریس همانی از متد eye استفاده می‌شود. شکل کلی ماتریس همانی به صورت زیر خواهد بود:

$$E_{i,j} = \delta_{i,j} \tag{۵}$$

که در آن $\delta_{i,j}$ تابع دلتای کرونکر است:

$$\delta_{i,j} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \tag{۶}$$

شکل کلی استفاده از متد eye به صورت زیر می‌باشد:

```
c = eye(row,column);
```

که در آن row تعداد سطرها و column تعداد ستون‌ها خواهد بود. برای مثال:

```
>> c = eye(3,4)

c =
     1     0     0     0
     0     1     0     0
     0     0     1     0
```

برای تعریف ماتریس مربعی، کافی است یک پارامتر به تابع ارسال شود. برای مثال:

```
>> c = eye(3)

c =
     1     0     0
     0     1     0
     0     0     1
```

- ماتریس جادویی: ماتریس جادویی، ماتریسی که جمع هر سطر با جمع هر ستون با جمع قطرها با هم برابرند. برای مثال ماتریس زیر یک ماتریس جادویی است:

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

برای تعریف این نوع ماتریس در متلب، از متد magic به صورت زیر استفاده می‌شود:

```
m = magic(Size);
```

که در آن، Size اندازه ماتریس مربعی است. برای مثال:

```
>> m = magic(5)

m =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

- ماتریس تصادفی یکنواخت: ماتریس تصادفی یکنواخت را، ماتریسی می‌خوانیم که درایه‌های آن از توزیع یکنواخت به صورت تصادفی بدست آمده است. برای تعریف یک ماتریس تصادفی یکنواخت از متد rand به صورت زیر استفاده می‌شود:

```
>> r = rand(row,column);
```

که در آن row تعداد سطرها و column تعداد ستون‌های ماتریس ساخته شده را نشان می‌دهد. برای مثال:

```
>> r = rand(3,4)

r =
    0.6887    0.8391    0.0842    0.0529
    0.1345    0.0346    0.3206    0.9817
    0.9852    0.0718    0.1617    0.6956
```

برای تعریف ماتریس مربعی تصادفی یکنواخت، تنها یک پارامتر به تابع فرستاده می‌شود. برای تولید ماتریس تصادفی نرمال -که درایه‌های آن با توزیع نرمال و به صورت تصادفی به دست آمده‌اند- از تابع randn استفاده می‌شود:

```
>> r = randn(row,column);
```

- تعریف بردار خطی: بردار خطی، از تقسیم یک پاره‌خط به چند قطعه مساوی ایجاد می‌شود. برای مثال، بردار زیر یک بردار خطی است:

$$\begin{bmatrix}1 & 1.25 & 1.5 & 1.75 & 2\end{bmatrix}$$

در این بردار، پاره‌خط ۱ تا ۲ با فاصله‌های مساوی ۰.۲۵ بریده شده‌اند. برای تعریف یک بردار خطی از متد linspace استفاده می‌شود. استفاده از این تابع به صورت زیر می‌باشد:

```
m = linspace(Start,End,Elements);
```

که در آن Start مقدار شروع پاره‌خط، End مقدار پایانی پاره‌خط و Elements تعداد تقسیم‌های پاره‌خط را مشخص می‌کند. برای مثال:

```
>> c = linspace(0,2,6)

c =
        0    0.4000    0.8000    1.2000    1.6000    2.0000
```

- عملگر کولُن: در متلب عملگر کولُن به صورت زیر تعریف می‌شود:

```
c = Start:Step:End;
```

این عملگر از مقدار Start شروع نموده و با طول گام Step تا End پیش می‌رود. برای مثال:

```
>> c = 0 : 0.3 : 1

c =
        0    0.3000    0.6000    0.9000
```

پ.۳.۳) دسترسی به عناصر یک ماتریس:

برای دسترسی به عناصر یک ماتریس از روش‌های زیر استفاده می‌شود:

- دسترسی با شماره سطر و ستون: در این روش از شکل زیر برای دسترسی به یک درایه استفاده می‌شود:

```
mat(row,column)
```

در این روش از ماتریس mat، درایه واقع در سطر row و ستون column را فرا می‌خوانیم. برای مثال:

```
>> a = rand(3)

a =
    0.7108   0.7985   0.3732
    0.2347   0.1690   0.7977
    0.7854   0.3817   0.3604

>> a(2,3)

ans =
    0.7977
```

- دسترسی با چینش خطی: در این روش همه‌ی درایه‌های ماتریس در یک صف به ترتیب از ستون اول قرار گرفته و شماره گذاری می‌شوند. به بیان دقیق‌تر:

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}_{m\times n} \rightarrow \begin{bmatrix} A(1) & A(m+1) & \cdots & A(mn-m+1) \\ A(2) & A(n+2) & \cdots & A(mn-m+2) \\ \vdots & \vdots & \ddots & \vdots \\ A(m) & A(2m) & \cdots & A(mn) \end{bmatrix}_{m\times n}$$

برای مثال:

```
>> a = rand(3)

a =
    0.7108   0.7985   0.3732
    0.2347   0.1690   0.7977
    0.7854   0.3817   0.3604

>> v1 = a(1)

v1 =
    0.7108

>> v2 = a(5)

v2 =
    0.1690

>> v3 = a(8)

v3 =
    0.7977
```

پ.۴.۳) تعریف یک ماتریس از ماتریس دیگر:

برای تعریف یک ماتریس از ماتریس دیگر، از یکی از فرم‌های زیر -و یا تلفیق آن- استفاده می‌شود:

```
newMat = oldMat(row,From:To);
```

در این روش، درایه‌های واقع در سطر row ماتریس oldMat از ستون From تا ستون To در ماتریس newMat رونوشت می‌شوند.

```
newMat = oldMat(From:to,column);
```

در این روش، درایه‌های واقع در ستون column ماتریس oldMat از سطر From تا ستون To در ماتریس newMat رونوشت می‌شوند.
برای فهم بهتر، لطفا به مثال‌های زیر توجه نمائید:

```
>> A = magic(5)

A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
```

```
    11    18    25     2     9

>> B1 = A(2,2:5)

B1 =
     5     7    14    16

>> B2 = A(1:4,3)

B2 =
     1
     7
    13
    19
```

با توجه به آنچه در بالا آمد، برای تعریف یک زیرماتریس از ماتریس معلوم، از روش زیر استفاده می‌شود:

```
newMat = oldMat(FromR:toR,FromC:ToC);
```

در این روش، درایه‌های واقع در سطر FromR تا toR و ستون FromC تا ToC ماتریس oldMat در ماتریس newMat رونوشت می‌شوند. برای مثال:

```
>> A = magic(5)

A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> B3 = A(1:4,2:5)

B3 =
    24     1     8    15
     5     7    14    16
     6    13    20    22
    12    19    21     3
```

پ.۵.۳) چند ثابت مهم:

در متلب، چند ثابت پراستفاده وجود دارند که در لیست زیر تعدادی از آنها آمده است:

- i: از i برای نشان دادن یکه موهومی استفاده می‌شود. برای مثال:

```
>> i^2

ans =
    -1
```

- ans: ans کلمه‌ی کلیدی است که برای دسترسی به آخرین نتیجه‌ی سرگردان استفاده می‌شود. نتیجه‌ی سرگردان، نتیجه‌ای است که به متغیر خاصی منسوب نشده باشد.

- pi: ثابت pi تقریبی از عدد π را نشان می‌دهد:

```
>> pi

ans =
   3.1416
```

- Inf: ثابت Inf نمایشی برای ∞ بر اساس استاندارد IEEE می‌باشد. برای مثال:

```
>> 1/0

ans =
   Inf
```

- NaN: ثابت NaN (Not a Number) حاصل عملیاتی را نشان می‌دهد که یک مقدار عددی تعریف‌نشده است. برای مثال:

```
>> Inf - Inf

ans =
  NaN

>> 0 / 0

ans =
  NaN

>> 0 * Inf

ans =
  NaN
```

پ.۴) اعمال حسابی و توابع بروی متغیرها

در متلب، به راحتی می‌توان یک متغیر را به عنوان پارامتر به تابع ارسال نمود. به عنوان مثال داریم:

- توابع size() و length(): برای بدست آوردن، اندازه یک ماتریس از تابع size به صورت زیر استفاده می‌شود:

```
>> a = 5;
>> size(a)

ans =
    1    1

>> b = eye(3,4);
>> size(b)

ans =
    3    4
```

برای بدست آوردن طول یک بردار از تابع length به صورت زیر استفاده می‌شود:

```
>> a = [1,2,3,4];
>> length(a)

ans =
    4
```

این تابع در ماتریس‌ها، بزرگترین بعد را برمی‌گرداند. برای مثال:

```
>> b = eye(3,4)

b =
    1    0    0    0
    0    1    0    0
    0    0    1    0

>> length(b)

ans =
    4
```

- ترانهاده یک ماتریس: برای بدست آوردن ترانهاده یک ماتریس، از عملگر ' استفاده می‌شود. برای مثال:

```
>> b = eye(3,4)

b =
    1    0    0    0
    0    1    0    0
    0    0    1    0

>> b'

ans =
    1    0    0
    0    1    0
    0    0    1
    0    0    0
```

- تابع norm(): برای بدست آوردن نُرم یک ماتریس از تابع norm استفاده می‌شود. انواع نرم‌های قابل استفاده در لیست زیر آمده است:
  - نُرم-۱: برای استفاده از نُرم-۱ در بردارها و ماتریس‌ها، از شکل زیر استفاده می‌شود:

```
norm(a,1);
```

برای مثال داریم:

```
>> a = [1,2,3,4];
>> norm(a,1)

ans =
   10

>> b = [1,2,3,4;-5,-6,-7,-8];
>> norm(b,1)
```

```
 ۹ |
۱۰ | ans =
۱۱ |    12
```

- نُرم-۲: برای استفاده از نُرم-۲ در بردارها و ماتریس‌ها، از فرم زیر استفاده می‌شود:

```
۱ | norm(a,2);
```

- نُرم-∞: برای استفاده از نُرم بی‌نهایت در بردارها و ماتریس‌ها، از فرم زیر استفاده می‌شود:

```
۱ | norm(a,Inf);
```

این نُرم در بردارها، مقدار بزرگترین درایه از نظر اندازه را برمی‌گرداند.

- نُرم فروبنیوس: برای استفاده از نُرم فروبنیوس در ماتریس‌ها و بردارها از فرم زیر استفاده می‌شود:

```
۱ | norm(a,'fro');
```

- نُرم-p: نُرم-p برای بردارها به صورت زیر تعریف می‌شود:

$$||V||_p = \sqrt[p]{\sum_i |V_i|^p} \tag{۷}$$

طریقه استفاده از این نُرم به صورت زیر است:

```
۱ | norm(vector,p);
```

که در آن vector بردار موردنظر و p نیز همان است که در تعریف ۷ آمد. برای مثال:

```
۱ | >> a = [1,2,3,4];
۲ | >> norm(a,8)
۳ |
۴ | ans =
۵ |    4.0498
```

ت

رسم نمودار (۱)

ت.۱) نحوه رسم

متلب برای رسم یک نمودار به صورتی کاملا طبیعی، زوج‌های مرتب -و یا سه‌تائی‌های مرتب- را تشکیل می‌دهد و آنها رو در یک صفحه قرار میدهد، سپس هر دو نقطه مجاور را به هم وصل می‌نماید.

ت.۲) plot()

برای رسم دوبعدی از تابع plot استفاده می‌شود. فرض کنید می‌خواهیم تابع $\sin x$ را در بازه $[-\pi, \pi]$ رسم نمائیم. ابتدا یک مجموعه از دامنه را برای رسم انتخاب می‌نمائیم. برای مثال:

```
>> d = -pi:1:pi

d =
   -3.1416  -2.1416  -1.1416  -0.1416   0.8584   1.8584   2.8584
```

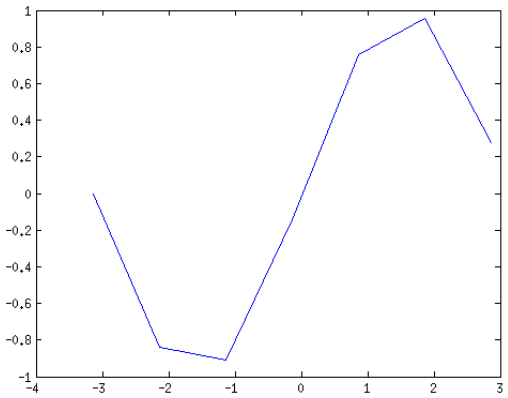
حال مقادیر تابع $\sin x$ را برای این مجموعه محاسبه می‌نمائیم:

```
>> r = sin(d)

r =
   -0.0000  -0.8415  -0.9093  -0.1411   0.7568   0.9589   0.2794
```

حال با استفاده از تابع plot نسبت به رسم اقدام می‌نمائیم:

```
>> plot(d,r)
```

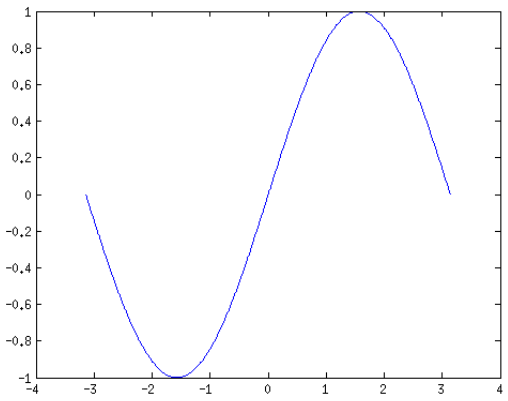
که نتیجه به صورت زیر خواهد بود:



برای افزایش کیفیت منحنی، می‌توان تعداد نقاط انتخابی از دامنه را افزایش داد و سپس به رسم اقدام نمود. برای این مثال:

```
>> d = -pi:0.01:pi;
>> r = sin(d);
>> plot(d,r)
```

که نتیجه به صورت زیر خواهد بود:



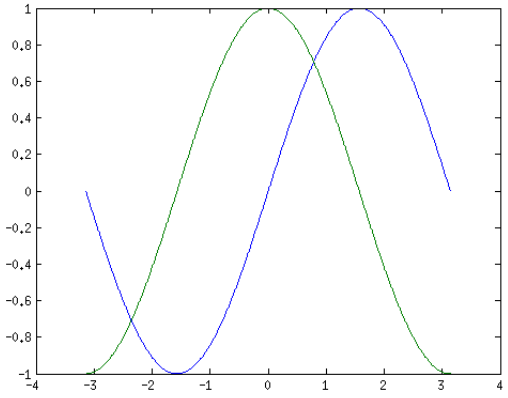
به صورت کلی، برای رسم دوبعدی چند تابع، بروی یک صفحه از فرم زیر استفاده می‌شود:

```
plot(d1,r1,d2,r2,d3,r3[,...])
```

برای مثال داریم:

```
>> d = -pi:0.01:pi;
>> r1 = sin(d);
>> r2 = cos(d);
>> plot(d,r,d,r2)
```

که نتیجه به صورت زیر خواهد بود:



برای تغییر در رنگ و نوع رسام، بعد از بردارهای مختصات، رشته‌ی خصوصیت رسام وارد می‌شود:

```
plot(d1,r1,'property1',d2,r2,'property2',d3,r3,'property3'[,...])
```

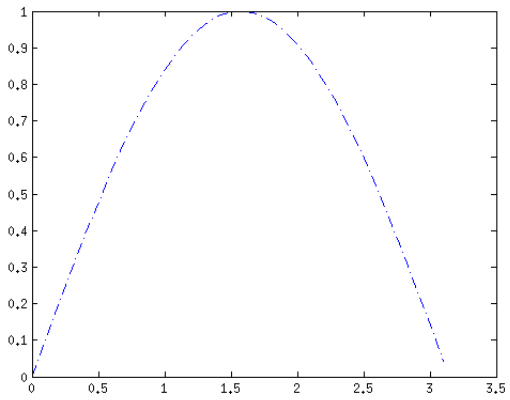
هر رشته‌ی خصوصیت به صورت 'SCT' می‌باشد که در آن:

- S: قالب خطوط متصل کننده را مشخص می‌کند. لیست قالب‌های قابل استفاده در زیر آمده است:
  - '-' : خطوط متصل به هم.
  - '--' : خطوط قطعه‌وار.
  - ':' : خطوط نقطه‌وار.
  - '-.' : خطوط قطعه-نقطه‌وار.
  - 'none' : بدون رسم خط.

برای مثال رسم سینوس در بازه $[0, \pi]$ با خاصیت .- به صورت زیر خواهد بود:

```
>> x = 0:0.1:pi;
>> y = sin(x);
>> plot(x,y,'-.')
```

که نتیجه‌ی آن به این صورت است:



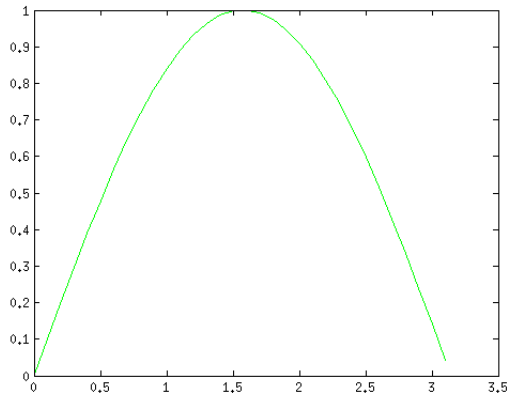
- C: رنگ رسام را مشخص می‌کند. رنگ‌های قابل استفاده در رسم در لیست زیر آمده است:
  - 'y' :yellow.
  - 'm' :magenta.
  - 'c' :cyan.
  - 'r' :red.

- ’g’ :green.
- ’b’ :blue.
- ’w’ :white.
- ’k’ :black.

برای مثال می‌خواهیم تابع سینوس را در بازه $[0, \pi]$ با رنگ سبز رسم نمائیم:

```
>> x = 0:0.1:pi;
>> y = sin(x);
>> plot(x,y,'g')
```

که نتیجه به صورت زیر خواهد بود:



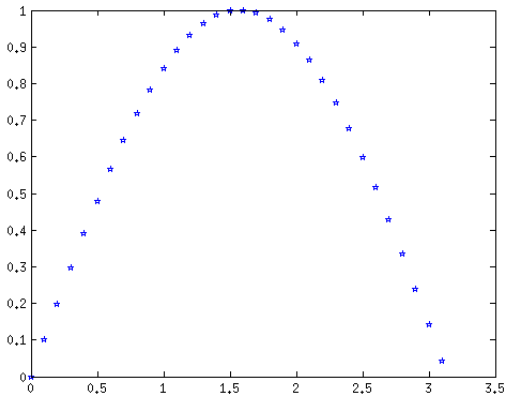
- T: نوع نشانگذار برای مشخص نمودن نقاط اصلی رسم را مشخص می‌نماید. لیست نشانگذارهای قابل استفاده در ادامه آمده است:
  - ’+’: علامت مثبت.
  - ’o’: دایره.
  - ’*’: ستاره.
  - ’.’: نقطه.
  - ’x’: ضربدر.
  - ’s’: مربع.
  - ’d’: لوزی.
  - ’^’: مثلث بالاجهت.
  - ’v’: مثلث پایین جهت.
  - ’>’: مثلث راست جهت.
  - ’<’: مثلث چپ جهت.
  - ’p’: ستاره ۵پر.

- 'h': ستاره ۶پر.
- 'none': بدون نشانگذار.

برای مثال می‌خواهیم تابع سینوس را در بازه $[0, \pi]$ با نشانگذار ستاره ۵پر رسم نمائیم:

```
>> x = 0:0.1:pi;
>> y = sin(x);
>> plot(x,y,'p')
```

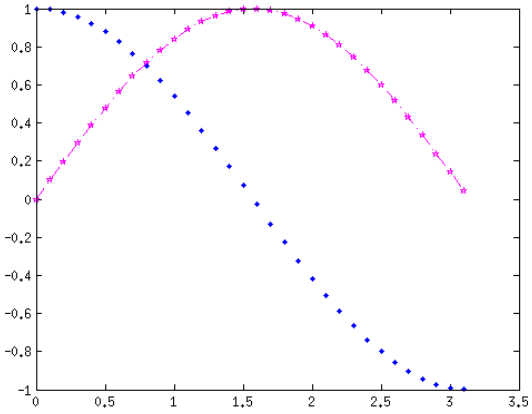
که نتیجه به صورت زیر خواهد بود:



به عنوان مثالی دیگر می‌خواهیم تابع سینوس را در بازه $[0, \pi]$ با خطوط متصل کننده «خط-نقطه»، نشانگذار «ستاره ۵پر» و به رنگ صورتی و تابع کسینوس را به صورت نقطه‌وار در همین بازه و در یک نمودار با هم رسم نمائیم:

```
>> x = 0:0.1:pi;
>> y = sin(x);
>> y2 = cos(x);
>> plot(x,y,'-.mp',x,y2,'.')
```

که نتیجه‌ی آن به صورت ذیل است:



ت.۳) plot3()

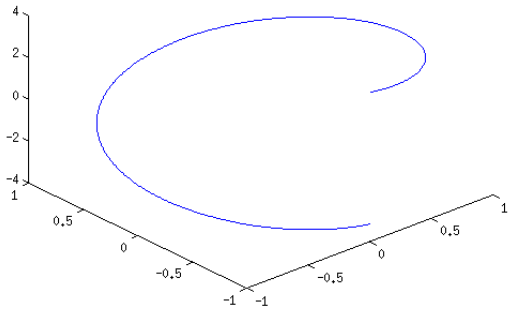
برای رسم ۳تائی‌های مرتب از تابع plot3 استفاده می‌شود. شکل کلی استفاده از این تابع به صورت زیر است:

```
plot3(x1,y1,z1[,...])
```

برای مثال داریم:

```
>> d = -pi:0.01:pi;
>> x1 = sin(d);
>> y1 = cos(d);
>> plot3(x1,y1,d)
```

که نتیجه‌ای مانند زیر را خواهد داشت:

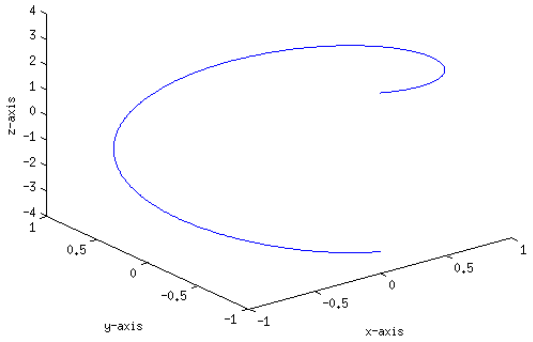


ت.۱.۳) برچسب گذاری محورها:

برای برچسب گذاری محورها از توابع، xlabel، ylabel و zlabel استفاده می‌شود. برای مثال:

```
>> d = -pi:0.01:pi;
>> x1 = sin(d);
>> y1 = cos(d);
>> plot3(x1,y1,d);
>> xlabel('x-axis');
>> ylabel('y-axis');
>> zlabel('z-axis');
```

و در نتیجه:



ت.۴) mesh() & surf()

برای رسم رویه‌ها در متلب، از توابع surf و mesh استفاده می‌شود. طریقه رسم یک رویه به این صورت است که ابتدا شبکه‌ای از نقاط دامنه تشکیل شده، مقادیر در آن نقاط به دست می‌آید و سپس مقادیر مجاور به هم متصل می‌شوند. تفاوت تابع surf و mesh در این است که تابع surf رویه را به صورت سایه‌دار رسم نموده اما تابع mesh آنرا به صورت خطوط درهم‌تنیده نشان می‌دهد.
برای تشکیل شبکه‌ی نقاط از تابع meshgrid به صورت زیر استفاده می‌شود:

```
[xN yN] = meshgrid(xV, yV);
```

که در آن xV بردار مختصات اول و yV بردار مختصات دوم برای تشکیل شبکه بوده و xN شبکه حاصل از بردار مختصات اول و yN شبکه حاصل از بردار دوم می‌باشد. به عنوان مثال:

```
xV =
     1     2     3

>> sV = [4 5 6]

yV =
     4     5     6
```

```
>> [xN yN]= meshgrid(xV,yV)

xN =

     1     2     3
     1     2     3
     1     2     3


yN =

     4     4     4
     5     5     5
     6     6     6
```

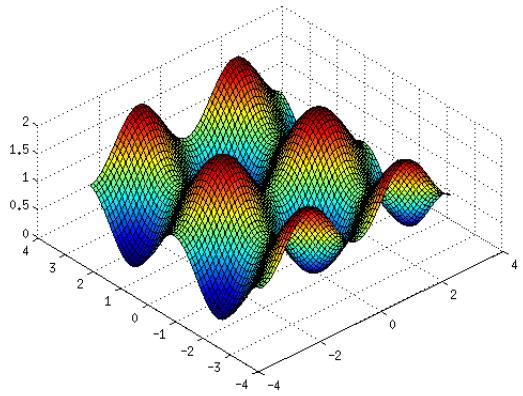
از تابع surf به صورت زیر استفاده می‌شود:

```
surf(xN,yN,zN);
```

که در آن xN شبکه متغیر اول، yN شبکه متغیر دوم و zN شبکه متغیر وابسته به آن دو می‌باشد. برای مثال:

```
>> xV = -pi:0.1:pi;
>> yV = -pi:0.1:pi;
>> [xN yN]= meshgrid(xV,yV);
>> zN = sin(xN).^2 + cos(yN).^2;
>> surf(xN,yN,zN);
```

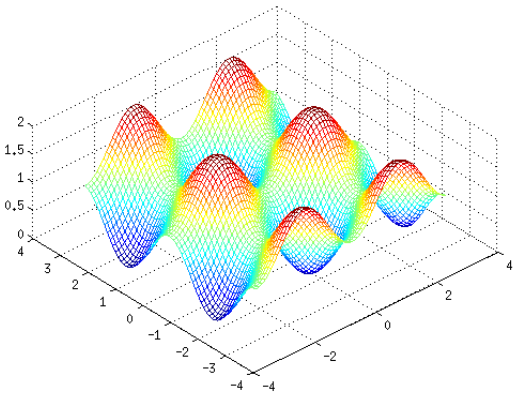
که نتیجه آن به صورت زیر خواهد بود:



استفاده از تابع mesh به صورت مشابه بالا خواهد بود، برای مثال:

```
>> xV = -pi:0.1:pi;
>> yV = -pi:0.1:pi;
>> [xN yN]= meshgrid(xV,yV);
>> zN = sin(xN).^2 + cos(yN).^2;
>> mesh(xN,yN,zN);
```

که نتیجه‌ی آن به صورت زیر است:


2
1.5
1
0.5
0
4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

ث

درونیابی

ث.۱) درونیابی یک‌بعدی

برای درونیابی دوبردار و بدست آوردن مقادیر درونیابی شده، می‌توان از تابع interp1 استفاده نمود. فُرم کلی تابع به این صورت می‌باشد:

```
۱ | i = interp1(xV,fV,pV,'Method');
```

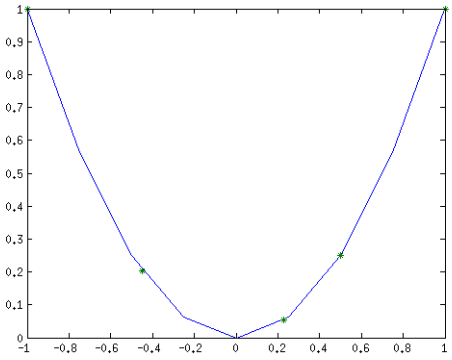
که در آن xV بردار نقاط حامی، fV بردار مقادیر حامی و pV بردار نقاطی است که می‌خواهیم آنها را درونیابی کنیم. متغیر i نیز برداری هم‌اندازه با بردار pV خواهد بود که در آن مقادیر درونیابی شده قرار دارد. رشته‌ی 'Method' نیز روش درونیابی مورد استفاده را مشخص می‌نماید، روش‌های قابل استفاده در متلب در لیست زیر آمده است:

- 'linear': Linear interpolation.
- 'nearest': Nearest neighbor interpolation.
- 'spline': Cubic spline interpolation
- 'pchip': Piecewise cubic Hermite interpolation.

برای مثال نقاط $\{-1, -0.45, 0.23, 0.5, 1\}$ را در بازه $[-1, 1]$ با مقادیر متناظرشان از تابع x^2 با گسسته‌سازی با طول ۰.۲۵ و روش اسپلاین درونیابی کرده و آنرا رسم می‌نمائیم:

```
۱ | pV = [-1,-0.45,0.23,0.5,1];
۲ | xV = -1:0.25:1;
۳ | fV = xV.^2;
۴ | i = interp1(xV,fV,pV,'spline');
۵ | plot(xV,fV,pV,i,'*');
```

که نتیجه‌ی حاصل به شکل زیر خواهد بود:



ث.۲) درونیابی دوبعدی

به صورت مشابه با درونیابی یک‌بعدی، درونیابی دوبعدی توسط تابع interp2 و به صورت زیر انجام می‌گیرد:

```
zR = interp2(xM,yM,zM,xPM,yPM,'Method');
```

در صورت بالا، xM و yM دو ماتریس به عنوان طول و عرض‌های حامی، zM ماتریس مقادیر حامی، متناظر با طول و عرض حامی، xPM و yPM ماتریس نقاطی که خواهان درونیابی آن هستیم و zR مقادیر درونیابی شده متناظر با آنها می‌باشد. رشته‌ی 'Method' نیز نوع درونیابی را مشخص می‌کند. روش‌های درونیابی قابل استفاده در درونیابی دوبعدی در لیست زیر آمده است:

- 'linear': Linear interpolation.
- 'nearest': Nearest neighbor interpolation.
- 'spline': Cubic spline interpolation.

در درونیابی دوبعدی نیاز است که ماتریس‌های xM و yM یکنوا باشند.

برای مثال می‌خواهیم تابع $f(x,y) = 1 - x^2 - y^2$ را در مجموعه نقاط زیر درونیابی کنیم:

$$\{(-4.5,-0.5),(-3.18,-0.23),(-0.5,0.2),(1.33,0.8),(4.85,0.93)\}$$

داریم:

```
>> x = [-5:0.5:5];
>> y = [-2:0.5:2];
>> [X Y] = meshgrid(x,y);
>> Z = 1 - X.^2 - Y.^2;
>> xP = [-4.5,-3.18,-0.5,1.33,4.85];
>> yP = [-0.5,-0.23,0.2,0.8,0.93];
>> zP = interp2(X,Y,Z,xP,yP,'spline');
>> zP
zP =
  -19.5000  -9.1653   0.7100  -1.4089 -23.3874
```

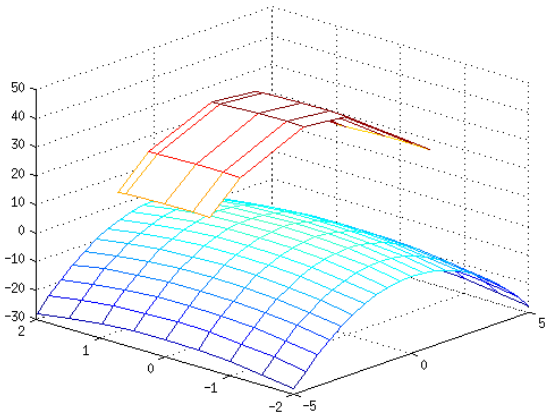
اگر بخواهیم نتیجه را رسم کنیم، باید قبل از درونیابی، مجموعه نقاط حامی را تبدیل به یک شبکه کنیم:

```
>> [XP,YP] = meshgrid(xP,yP);
```

دراینصورت:

```
>> x = [-5:0.5:5];
>> y = [-2:0.5:2];
>> [X Y] = meshgrid(x,y);
>> Z = 1 - X.^2 - Y.^2;
>> xP = [-4.5,-3.18,-0.5,1.33,4.85];
>> yP = [-0.5,-0.23,0.2,0.8,0.93];
>> [XP YP] = meshgrid(xP,yP);
>> ZP = interp2(X,Y,Z,XP,YP,'spline');
>> mesh(X,Y,Z);
>> hold
Current plot held
>> mesh(XP,YP,ZP + 40);
>> hold off
```

در تکه کد بالا، در خط ۷ یک شبکه از نقاط حامی تشکیل داده و سپس این شبکه از نقاط را درونیابی می‌کنیم، سپس در خط ۹ تابع اصلی را رسم می‌نمائیم. با استفاده از تابع hold (خط ۱۰) پنجره کنونی را برای رسم و بازنویسی دوباره، آماده نگه می‌داریم، سپس برای اینکه تفاوت تابع اصلی و تابع دورنیابی‌شده مشهود باشد، مقادیر درونیابی شده را با یک اختلاف ۴۰تایی رسم می‌نمائیم (خط ۱۲)، در نهایت با دستور hold off حالت بازنویسی بروی رسم فعلی را غیرفعال می‌نمائیم. نتیجه به صورت زیر خواهد بود:



ج | تبدیل سریع فوریه

تبدیل فوریه، نامیده شده به نام ژوزف فوریه، ریاضیدان فرانسوی، یک تبدیل ریاضی است که در فیزیک و مهندسی کاربردهای بسیار زیادی دارد. به بیان ساده این تبدیل یک تابع $f(t)$ بر حسب زمان را به یک تابع جدید $F(\omega)$ که در آن ω در واحد رادیان بر ثانیه است می‌برد. این تبدیل به صورت زیر تعریف می‌شود:

$$F(\omega) = F\{f(t)\} = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \qquad (۸)$$

در صورتی که تابع f ذکر شده در بالا، گسسته بوده و به صورت $\{x_i\}_{i=1}^{N-1}$ باشد، تبدیل گسسته فوریه به صورت زیر تعریف می‌شود:

$$X_k = DFT\{x_i\} = \sum_{n=0}^{N-1} x_n e^{-i(2\pi kn/N)} \qquad (۹)$$

این تبدیل در حقیقت یک تقریب از تبدیل پیوسته فوریه می‌باشد که برای محاسبات کامپیوتری مناسب است. برای پیاده‌سازی‌های تقریب گسسته فوریه راه‌های متفاوتی وجود دارد که بهترین الگوریتم آن، تبدیل سریع فوریه می‌باشد. به صورت مشابه تبدیل گسسته فوریه در حالت دوبعدی تعریف می‌شود و برای این تبدیل نیز الگوریتم تبدیل سریع فوریه کارا خواهد بود.

ج.۱) توابع fft() & ifft()

در متلب، تابع fft پیاده‌سازی الگوریتم تبدیل فوریه سریع می‌باشد و به صورت زیر مورد استفاده قرار می‌گیرد:

```
Y = fft(X);
```

که در آن X بردار و یا ماتریس ورودی و Y تبدیل گسسته فوریه آن می‌باشد. در صورتی که X ماتریس باشد، تابع fft تبدیل را برای هر ستون آن محاسبه می‌نماید. همچنین برای اعمال معکوس تبدیل سریع فوریه از تابع ifft استفاده می‌شود:

```
X = ifft(Y);
```

برای مثال:

```
>> x = [1 2 3 4];
>> y = fft(x)

y =
  10.0000  -2.0000 + 2.0000i  -2.0000  -2.0000 - 2.0000i

>> x2 = ifft(y)

x2 =
    1    2    3    4
```

ج.۲) توابع fft2() & ifft2()

برای استفاده از تبدیل گسسته فوریه دو بعدی، از تابع fft2 و ifft2 استفاده می‌شود. طریقه استفاده از این تابع به صورت زیر می‌باشد:

```
Y = fft2(X,m,n);
```

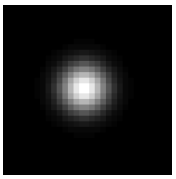
در تعریف بالا X بردار یا ماتریس ورودی و Y بردار یا ماتریس متناظر حاصل از تبدیل گسسته فوریه می‌باشد. در تابع بالا m و n اعدادی است که بوسیله آنها، قبل از انجام تبدیل، بردار یا ماتریس را با افزودن صفر به یک ماتریس $m \times n$ تبدیل می‌کنیم.
برای مثال داریم:

```
>> x = [1 2;3 4];
>> y = fft2(x)

y =
   10   -2
   -4    0

>> x2 = ifft2(y)

x2 =

    1    2
    3    4

>> y2 = fft2(x,3,4)

y2 =
  10.0000             4.0000-6.0000i     -2.0000             4.0000 + 6.0000i
  -0.5000-6.0622i     -3.9641-2.5981i    -0.5000+0.8660i     2.9641 - 2.5981i
  -0.5000+6.0622i     2.9641+2.5981i     -0.5000-0.8660i    -3.9641 + 2.5981i

>> x3 = ifft2(y2)

x3 =
    1.0000   2.0000        0        0
    3.0000   4.0000        0        0
   -0.0000   0.0000  -0.0000  -0.0000
```

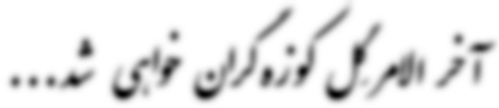
ج.۳) یک کاربرد در پردازش تصویر

یکی از کاربردهای تبدیل فوریه در پردازش تصویر، مات‌زدایی[۸] می‌باشد. مات شدن یک تصویر، نتیجه‌ی پخش‌شدن نور هر نقطه تصویر در قسمت‌های مجاور آن می‌باشد. با استفاده از تبدیل فوریه و آزمون و خطا نتیجه می‌شود که تصاویر دارای ماتی حاوی تصویری به مانند زیر می‌باشند:

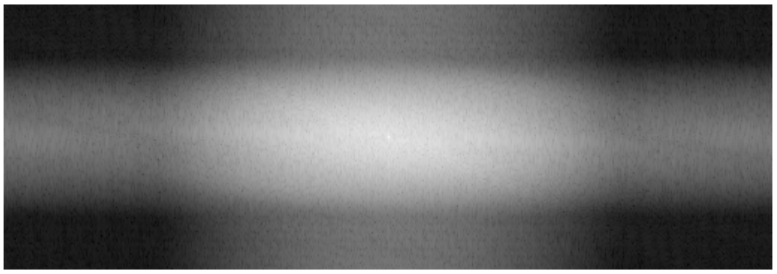
[۸] Deblurring



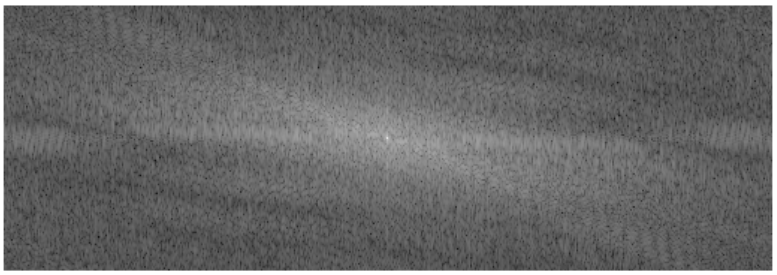
برای مثال تصویر زیر را در نظر بگیرید:



تصویر حاصل از تبدیل فوریه بروی این تصویر به صورت زیر می‌باشد:



در حالیکه تصویر حاصل از تبدیل فوریه بروی تصویر مات‌نشده به صورت ذیل است:



در حقیقت می‌توان ادعا نمود که تصویر مات‌شده، حاصلضرب ماتریس تصویر اولیه و تصویر پراکندگی نور می‌باشد. بنابراین برای رفع ماتی تصویر، می‌توان به صورت عکس عمل نموده و تصویر مات را در معکوس تصویر پراکندگی نور ضرب نمود.
تکه‌کد زیر، قسمتی از این عمل را نشان می‌دهد:

```
۱ fftBlurImage = fft2(blurImage);
۲ fftBlurKernel = fft2(kernelImage);
```

```
۳
۴ fftUnblurImage = fftBlurimage ./ fftkernelImage;
۵
۶ unBlurImage = ifft2(fftUnblurImage);
```

که نتیجه‌ی این عمل، تصویر زیر خواهد بود[۹]:

[۹] m-فایل مثال مورد بحث را می‌توان از http://meysam.ws/math/matlab/debluring.zip دانلود نمود.

چ | تبدیل لاپلاس

چ.۱) تعریف متغیرهای نمادین

اشیاء نمادین[۱۰] گونه‌ی خاصی از متغیرهای مختص متلب هستند که به کاربر اجازه انجام اعمال ریاضی به صورت تحلیلی و بدون محاسبه مقادیر عددی را می‌دهند.

چ.۱.۱) کلمه کلیدی **syms**:

متغیرهای نمادین در متلب با کلمه کلیدی syms و به صورت زیر تعریف می‌شوند:

```
syms var_name;
```

برای مثال داریم:

```
>> syms x y;
>> x * x + y + y

ans =
x^2 + 2*y
```

چ.۲) تابع **laplace()**

برای بدست آوردن تبدیل لاپلاس یک تابع در متلب، از تابع laplace به صورت زیر استفاده می‌شود:

```
L = laplace(F,w,z);
```

که آن F تابع، w متغیر مستقل تابع F و z متغیری است که تابع L، حاصل از تبدیل لاپلاس بر اساس آن تعریف می‌شود. برای مثال می‌خواهیم تبدیل لاپلاس تابع $f(t) = -1.25 + 3.5te^{-2t} + 1.25e^{-2t}$ را محاسبه نمائیم:

```
>> syms t s;
>> f = -1.25 + 3.5*t*exp(-2*t) + 1.25*exp(-2*t);
>> L = laplace(f,t,s)

L =
5/(4*(s + 2)) + 7/(2*(s + 2)^2) - 5/(4*s)
```

چ.۳) توابع **pretty() & simplify()**

برای ساده‌سازی عبارات جبری حاصل از اعمال ریاضی بروی متغیرهای نمادین، از تابع simplify به صورت زیر استفاده می‌شود:

```
s = simplify(S);
```

[۱۰] Symbolic Objects

که در آن S یک عبارت جبری نمادین و s عبارت نمادین حاصل از ساده‌سازی می‌باشد. برای مثال، برای تابع بدست آمده از تبدیل لاپلاس در بالا داریم:

```
>> l = simplify(L)

l =
(s - 5)/(s*(s + 2)^2)
```

با استفاده از تابع pretty می‌توان عبارت بدست آمده در بالا را به صورت ریاضی‌وار چاپ نمود:

```
>> pretty(l)
   s - 5
  ----------
           2
  s (s + 2)
```

چ.۴) تابع ilaplace()

تابع ilaplace معکوس یک تابع حاصل از تبدیل لاپلاس را محاسبه می‌نماید. طریقه استفاده از این تابع به صورت زیر می‌باشد:

```
F = ilaplace(L);
```

که در آن L تابع حاصل از تبدیل و F تابع حاصل از تبدیل معکوس می‌باشد. برای مثال بالا داریم:

```
>> F2 = ilaplace(l)

F2 =
(5*exp(-2*t))/4 + (7*t*exp(-2*t))/2 - 5/4
```

ح رسم نمودار (۲)

برای رسم توابع چندظابطه‌ای، از متغیرهای نمادین و تابع هوی‌ساید[۱۱] استفاده می‌شود. تابع هوی‌ساید به صورت زیر تعریف می‌شود:

$$H(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \qquad (۱۰)$$

در متلب، تابع هوی‌ساید به صورت بالا تعریف می‌شود، با این تفاوت که $H(0) = 0.5$. تابع هوی‌ساید در متلب به صورت زیر استفاده می‌شود:

```
f = heaviside(s);
```

که در آن s یک متغیر نمادین و f مقدار تابع در آن می‌باشد. برای مثال:

```
>> heaviside(sym(2))

ans =
1

>> heaviside(sym(0))

ans =
1/2

>> heaviside(sym(-4))

ans =
0
```

در مثال بالا توضیح این نکته لازم است که در متلب، برای تعریف اعداد نمادین از کلمه کلیدی sym استفاده می‌شود.

ح.۱) ایده‌ی اصلی

ایده‌ی کلی برای رسم توابع چندظابطه‌ای به این صورت است که می‌خواهیم تابع را بر اساس بازه به صورت یک حاصلجمع از از حاصلضرب توابع هوی‌ساید و ظابطه‌ها بنویسیم. برای مثال تابع زیر را در نظر بگیرید:

$$f(x) = \begin{cases} sin(x) & x \in [-\pi, 1) \\ 1 & x \in [1, 2) \end{cases}$$

می‌خواهیم برای این تابع در بازه‌ی اول، مقدار ظابطه اول را گرفته و بقیه ظابطه‌ها صفر گردند. برای این کار، ابتدا با استفاده از تابع هوی‌ساید، مقدار ظابطه را در $(-\infty, \pi)$ صفر می‌کنیم:

$$g_1(x) = \begin{cases} 0 & x \in (-\infty, \pi) \\ sin(x) & x \in [-\pi, \infty) \end{cases} = sin(x) \begin{cases} 0 & x \in (-\infty, -\pi) \\ 1 & x \in [-\pi, \infty) \end{cases} = sin(x) \times H(x+\pi)$$

[۱۱] Heaviside

چرا که در $H(x+\pi)$ برای $x+\pi \geq 0$ داریم: $H(x)=1$. به صورت مشابه، ظابطه را در $(1,\infty)$ صفر می‌کنیم:

$$g_2(x)=\begin{cases} sin(x) & x\in(-\infty,1) \\ 0 & x\in[1,\infty) \end{cases} = sin(x)\begin{cases} sin(x) & x\in(-\infty,1) \\ 0 & x\in[1,\infty) \end{cases} = sin(x)\times H(1-x)$$

بنابراین، برای فصل مشترک این دو یعنی $[\pi,1)$ داریم:

$$g(x)=\begin{cases} 0 & x\in(-\infty,-\pi) \\ sin(x) & x\in[-\pi,1) \\ 0 & x\in[1,\infty) \end{cases} = sin(x)\times H(x+\pi)\times H(1-x)$$

با عمل بروی ظابطه‌ی دیگر به صورت مشابه، در نهایت نتیجه زیر را خواهیم داشت:

$$f(x)=sin(x)\times H(x+\pi)\times H(1-x)+H(x-1)\times H(2-x)$$

به صورت کلی، برای نوشتن یک تابع چندظابطه‌ای به صورت یک تابع تک‌ظابطه‌ای با استفاده از تابع هوی‌ساید، داریم:

$$f(x)=\begin{cases} f_1(x) & x\in[a_1,b_1) \\ f_2(x) & x\in[a_2,b_2) \\ \vdots & \vdots \\ f_k(x) & x\in[a_k,b_k) \end{cases} = \sum_{i=1}^{k}(H(x-a_i)\times H(b_i-x)\times f_i(x)) \qquad (۱۱)$$

ح.۲) تابع ezplot

برای رسم توابع چندظابطه‌ای از تابع ezplot به فُرم زیر استفاده می‌شود:

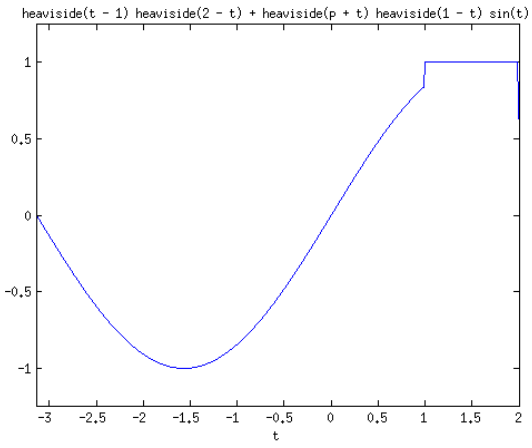
```
ezplot(f,[xmin,xmax]);
```

که در آن، f یک تابع بر حسب متغیرهای نمادین، xmin مقدار کمینه متغیر مستقل و xmax مقدار بیشینه متغیر مستقل می‌باشد. برای مثال تابع زیر را رسم می‌کنیم:

$$f(x)=\begin{cases} sin(x) & x\in[-\pi,1) \\ 1 & x\in[1,2) \end{cases} = sin(x)\times H(x+\pi)\times H(1-x)+H(x-1)\times H(2-x)$$

```
>> syms t;
>> f = sin(t) * heaviside(t+pi) * heaviside(1-t) + heaviside(t-1) * heaviside
    (2-t)

f =
heaviside(t - 1)*heaviside(2 - t) + heaviside(pi + t)*heaviside(1 - t)*sin(t)

>> ezplot(f,[-pi,2])
```

نتیجه به صورت زیر خواهد بود:


heaviside(t - 1) heaviside(2 - t) + heaviside(p + t) heaviside(1 - t) sin(t)
1
0.5
0
-0.5
-1
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
t

خ

حل ODE

خ.۱) یافتن جواب دقیق

در متلب برای حل دقیق معادلات دیفرانسیل از تابع dsolve استفاده می‌شود. طریقه استفاده از این تابع به صورت زیر است:

```
S = dsolve(equation,condition,valueName);
```

که در آن equation یک معادله نمادین، condition شرایط اولیه یا مرزی معادله، valueName نام متغیر مستقل نمادین و S جواب بدست آمده از حل معادله می‌باشد. برای مثال می‌خواهیم معادله مرتبه اول $y'(x) = xy$ را حل نمائیم. داریم:

```
>> syms y(x);
>> eqn = 'Dy = y*x';
>> dsolve(eqn,'x')

ans =
C2*exp(x^2/2)

>> pretty(ans)

       / 2 \
       | x |
  C2 exp| -- |
       \ 2 /
```

در خط ۱ متغیر نمادین مستقل x و تابع نمادین y را بر حسب آن تعریف می‌کنیم. با استفاده از نماد D در خط ۲ مرتبه‌ی معادله مشخص شده و سپس معادله را بدون مقدار اولیه حل نمودیم و جواب $C_2 e^{\frac{x^2}{2}}$ بدست آمده است.
برای مثالی دیگر می‌خواهیم معادله $y''(x) + 8y'(x) + 2y(x) = cos(x)$ را با شرایط اولیه $y(0) = 1, y'(0) = 1$ حل کنیم. داریم:

```
>> syms y(x);
>> eqn = 'D2y+8*Dy+2*y=cos(x)';
>> condi = 'y(0)=0, Dy(0)=1';
>> y = dsolve(eqn,condi,'x')

y =

(14^(1/2)*exp(4*x - 14^(1/2)*x)*exp(x*(14^(1/2) - 4))*(sin(x) - cos(x)
    *(14^(1/2) - 4)))/(28*((14^(1/2) - 4)^2 + 1)) - (14^(1/2)*exp(4*x +
    14^(1/2)*x)*exp(-x*(14^(1/2) + 4))*(sin(x) + cos(x)*(14^(1/2) + 4)))
    /(28*((14^(1/2) + 4)^2 + 1)) - (14^(1/2)*exp(-x*(14^(1/2) + 4))
    *(7*14^(1/2) + 27))/(28*(8*14^(1/2) + 31)) - (14^(1/2)*exp(x*(14^(1/2) -
    4))*(393*14^(1/2) + 1531))/(28*(8*14^(1/2) - 31)*(8*14^(1/2) + 31)^2)

>> simplify(y)

ans =
```

```
cos(x)/65 + (8*sin(x))/65 - (exp(-4*x)*exp(14^(1/2)*x))/130 - (exp(-4*x)*exp
    (-14^(1/2)*x))/130 + (53*14^(1/2)*exp(-4*x)*exp(14^(1/2)*x))/1820 -
    (53*14^(1/2)*exp(-4*x)*exp(-14^(1/2)*x))/1820

>> pretty(ans)

                                 1/2                    1/2
  cos(x)  8 sin(x)  exp(-4 x) exp(14 x)   exp(-4 x) exp(- 14 x)
  ------ + -------- - --------------------- - ----------------------- +
    65       65             130                    130

        1/2              1/2          1/2               1/2
     53 14   exp(-4 x) exp(14 x)   53 14   exp(-4 x) exp(- 14 x)
     ------------------------------ - --------------------------------
                 1820                            1820
```

خ.۱.۱) رسم توابع نمادین با **plot**

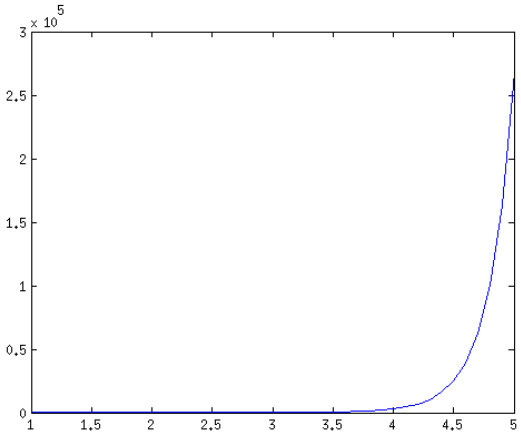
در بخش (ح.۲) یک روش برای رسم توابع نمادین معرفی شد. در این بخش می‌خواهیم توابع نمادین را به صورت معمول و با تابع plot رسم نمائیم. ایده در اینجا به این صورت است که ابتدا نقاطی که می‌خواهیم تابع را در آن رسم کنیم را به متغیر مستقل تخصیص داده، سپس مقادیر آنرا را در تابع نمادین ارزیابی می‌کنیم و مقادیر بدست آمده را رسم می‌کنیم.
برای انجام این رویکرد، به صورت زیر عمل می‌کنیم:

```
f = eval(vectorize(y));
```

که در آن y تابع نمادین وابسته به متغیر مستقل و f بردار مقادیر ارزیابی شده نقاط در تابع نمادین است. برای مثال:

```
>> syms y(x);
>> eqn = 'Dy = y*x';
>> cond = 'y(0) = 1';
>> y = dsolve(eqn,cond,'x');
>> x = 1:0.1:5;
>> f = eval(vectorize(y));
>> plot(x,f)
```

و نتیجه به صورت زیر خواهد بود:



در تکه‌کد بالا، ابتدا در خط ۱ متغیر نمادین x و تابع y وابسته به آن تعریف شده و سپس در خط ۵ به x مقدار عددی داده شده است.

خ.۲) یافتن جواب عددی

برای یافتن تقریب‌های عددی برای جواب معادلات دیفرانسیل با استفاده از روش‌های عددی از دو تابع اصلی ode23 و ode45 استفاده می‌شود. تابع ode23 پیاده‌سازی روش رانگ-کوتا مرتبه دوم و سوم و تابع ode45 پیاده‌سازی روش رانگ-کوتا مرتبه چهارم و پنجم می‌باشد.
برای استفاده از توابع ذکر شده در متلب، به شیای به نام تابع ناشناس[۱۲] و یا Function handler نیاز می‌باشد.

خ.۱.۲) توابع ناشناس:

توابع ناشناس به صورت زیر تعریف می‌شود:

```
handle = @(arguments)anonymous_function
```

به وسیله ابزار توابع ناشناس، می‌توان یک تابع را به عنوان پارامتر به تابع دیگر فرستاد و یا برای استفاده‌های بعدی ذخیره نمود.

خ.۲.۲) **ode23**:

صورت کلی تابع ode23 به صورت زیر می‌باشد:

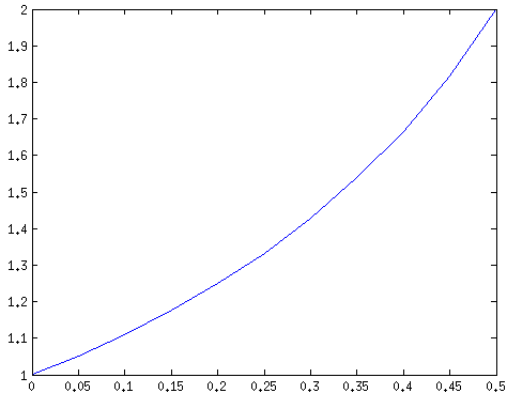
```
[X Y] = ode23(odeFun, tspan, y0);
```

که در آن odeFun یک دستگیره[۱۳] از معادله دیفرانسیل تعریف شده در (خ.۱.۲)، tspan بازه موردنظر برای تقریب و y0 بردار مقادیر اولیه مساله می‌باشد.
برای مثال می‌خواهیم معادله $y' = xy^2 + y$ با شرایط اولیه $y(0) = 1$ را بر بازه $[0, 0.5]$ تقریب زده و رسم نمائیم. داریم:

[۱۲] Anonymous function
[۱۳] handler

```
>> syms y(x);
>> f = @(x,y) x*y^2 + y;
>> [X,Y] = ode23(f,[0,0.5],1);
>> plot(X,Y)
```

و نتیجه به این شکل خواهد بود:



خ.۳.۲) **ode45**:

شکل کلی تابع ode45 مانند ode23 بوده و به صورت زیر می‌باشد:

```
[X Y] = ode45(odeFun, tspan, y0);
```

د نوشتن تابع دلخواه

در متلب می‌توان یک الگوریتم دلخواه را پیاده‌سازی نموده و از آن به دفعات مکرر و یا در سکوهای دیگر استفاده نمود. در این بخش، ابزارهای موردنیاز برای پیاده‌سازی یک الگوریتم دلخواه بررسی می‌شوند.

د.۱) File → New → Function:

برای نوشتن توابع دلخواه، به مانند آنچه در بخش (ب.۲.۱) آمد، می‌توان از ویرایشگر داخلی متلب استفاده نمود. برای دسترسی به ویرایشگر مخصوص تابع مسیر File → New → Function را در محیط متلب بپیمایید.

د.۲) شکل کلی یک تابع

شکل کلی یک تابع در متلب به صورت زیر می‌باشد:

```
۱ function returnF = funcName(Parameters)
۲     %Comment's Here!
۳     Code goes here!
۴     .
۵     .
۶     .
۷     returnF = data...;
۸ end
```

با کلمه کلیدی function محیط تابع آغاز و با کلمه کلیدی end پایانی، محیط به پایان می‌رسد. نام تابع برای صدا زدن funcName و مقادیر ورودی Parameters می‌باشند. تابع پس از انجام محاسبات لازم، مقدار returnF را باز می‌گرداند. در متلب قطعه‌ها[۱۴] با تورفتگی[۱۵] مشخص شده و کلماتی که در جلوی علامت % قرار گیرند، نادیده گرفته می‌شوند. در هنگام ذخیره تابع در متلب، بایستی نام تابع و نام فایل ذخیره‌شونده یکسان باشند.
برای مثال تابعی به نام firstSum می‌نویسم که دومقدار را از کاربر گرفته و پس از جمع، مقدار مجموع را بازگرداند:

```
۱ function result = firstSum(a,b)
۲     %Sum a with b :D
۳     s = a + b;
۴
۵     %return result
۶     result = s;
۷ end
```

پس از نوشتن کد، آنرا با نام firstSum.m در پوشه جاری متلب (ر.ک. به (ب.۱.۱)) ذخیره نمائید.

[۱۴] Blocks
[۱۵] Identition

د.۳) عملگرهای رابطه‌ای

عملگرهای رابطه‌ای، رابطه‌ی بین دو شی را بررسی کرده و نتیجه را با توجه به آن دو بازمی‌گردانند. عملگرهای رابطه‌ای متلب در لیست زیر آمده است:

- ==, &, ~=: برای مقایسه برابری یا عدم برابری دو شی از این دوعملگر استفاده می‌شود. عملگر == در صورتی که دوطرف عملگر با هم برابر باشند مقدار صحیح و در غیراینصورت مقدار غلط را باز می‌گرداند. در متلب، هر چیز ناصفر معادل با درست و صفر معادل با غلط می‌باشد. برای مثال این عملگر داریم:

```
>> 2==2

ans =
     1

>> 2==5

ans =
     0
```

عملگر ~= به صورت عکس عملگر == عمل می‌نماید، یعنی الگر دو عملوند با هم برابر نباشند مقدار درست و درغیراینصورت مقدار غلط را باز می‌گرداند. برای مثال:

```
>> 2~=2

ans =
     0

>> 2~=5

ans =
     1
```

در ماتریس‌های با اندازه مشابه درایه‌ها نظیر به نظیر با هم مقایسه شده و نتیجه در یک ماتریس با اندازه مشابه نشان داده می‌شود. برای مثال:

```
>> a = zeros(3);
>> b = zeros(3);
>> a==b

ans =
     1     1     1
     1     1     1
     1     1     1

>> b(5) = 1;
>> a==b

ans =
     1     1     1
     1     0     1
     1     1     1
```

در صورتیکه یکی از عملوندها ماتریس و دیگری عدد باشد، داریه‌های ماتریس به صورت مجزا با عدد مقایسه شده و نتیجه در یک ماتریس با سایز مشابه ماتریس مورد مقایسه، بازگردانده می‌شود. برای مثال:

```
>> a = magic(3)

a =
     8     1     6
     3     5     7
     4     9     2

>> a == 4

ans =
     0     0     0
     0     0     0
     1     0     0

>> a ~= 4

ans =
     1     1     1
     1     1     1
     0     1     1
```

- < & =< عملگر < همان عملگر ترتیب در یک میدان مرتب می‌باشد. در صورتی که عملوند اول بزرگتر از عملوند دوم باشد مقدار صحیح و در غیراینصورت مقدار غلط بازمی‌گرداند. عملگر =< در صورتی که عملوند اول بزرگتر یا مساوی عملگر دوم باشد مقدار صحیح و در غیراینصورت مقدار غلط باز می‌گرداند. برای مثال:

```
>> a = magic(3)

a =
     8     1     6
     3     5     7
     4     9     2

>> a > 2

ans =
     1     0     1
     1     1     1
     1     1     0

>> a >= 2

ans =
     1     0     1
     1     1     1
     1     1     1
```

- < & =<: عملگر < در صورتی که مقدار عملوند اول کوچکتر از عملوند دوم باشد مقدار صحیح و در غیراینصورت مقدار غلط باز می‌گرداند. عملگر

=> درصورتی که عملوند اول کوچکتر یا مساوی عملوند دوم باشد، مقدار صحیح و درغیراینصورت مقدار غلط را بازمی‌گرداند.

```
>> a = magic(3)

a =
     8     1     6
     3     5     7
     4     9     2

>> a < 3

ans =
     0     1     0
     0     0     0
     0     0     1

>> a <= 3

ans =
     0     1     0
     1     0     0
     0     0     1
```

- && & ||: عملگر && در صورتیکه هردومقدار همزمان دارای ارزش صحیح باشند، مقدار صحیح و در غیراینصورت مقدار غلط باز می‌گرداند. برای مثال:

```
>> 5 && 1

ans =
     1

>> 5 && 0

ans =
     0
```

عملگر || درصورتیکه یکی از دومقدار دارای ارزش صحیح باشند مقدار صحیح و در غیراینصورت مقدار غلط باز می‌گرداند. برای مثال:

```
>> 5 || 1

ans =
     1

>> 5 || 0

ans =
     1

>> 0 || 0

ans =
     0
```

هر دو عملگر && و || تنها برای عملوندهایی قابل استفاده‌اند که قابلیت تبدیل به یک اسکالر عددی را داشته باشند، بنابراین این دو عملوند برای ماتریس‌ها و بردارها قابل اعمال نیستند.

- ~: عملگر ~ تنها بر روی یک عملوند اثر می‌کند و در صورتی که آن عملوند درست باشد، مقدار غلط و در صورتی که ارزش عملوند درست مقدار غلط را باز می‌گرداند. برای مثال:

```
>> ~5

ans =
     0

>> ~0

ans =
     1
```

د.۴) دستور زبان متلب

اگر موارد بحث شده در (د.۳) وقبل از آنرا را به پیچ و مهره تشبیه کنیم، در این فصل طریقه استفاده از ابزارهایی را خواهیم آموخت که به وسیله آنها می‌توان یک ابزار برای رفع یک مشکل طراحی نمود که بارها و بارها قابل استفاده باشد. تمام این ابزارها در ۲ دسته کلی تقسیم‌بندی می‌شوند: ۱) حلقه‌ها، ۲) شرطی‌ها.

۱. if/else/elseif: برای بررسی یک شرط و اجرای متناسب برنامه با توجه به نتیجه‌های حاصل از شرطها، از کلمات کلیدی if/else/elseif استفاده می‌شود.

- if: در صورتی که عملوند کلمه کلیدی if درست باشد، قطعه متناظر با آن اجرا می‌شود و درغیراینصورت از اجرای آن صرفنظر می‌شود.
  برای مثال می‌خواهیم تابعی بنویسم که دو عدد را دریافت کرده و در صورتی که دو عدد متوالی باشند حاصلجمع آنها را بازگرداند:

```
function res = orderSum(a,b)
   if(abs(a-b)==1)
      res = a+b;
   end
end
```

به عنوان نمونه‌ای از اجرای این برنامه داریم:

```
>> orderSum(1,2)

ans =
     3

>> orderSum(1,3)
>>
```

- else: کلمه کلیدی else به همراه و بعد از قطعه کد مربوط به کلمه کلیدی if به کار می‌رود. ساختار اجرای else به این صورت است که در صورتی که عملوند if ارزش درست نداشته باشد، قطعه کد مربوط به خود را اجرا می‌کند.

برای مثال، در برنامه بالا می‌خواهیم در صورتی که دو عدد متوالی نبودند، تفریق دو عدد ورودی را بازگرداند:

```
function res = orderSum(a,b)
    if(abs(a-b)==1)
        res = a+b;
    else
        res = a-b;
    end
end
```

به عنوان مثالی از نتیجه‌ی اجرا:

```
>> orderSum(1,2)

ans =
     3

>> orderSum(1,3)

ans =
    -2
```

- elseif: در صورتی که شقوق مساله، بیش از دو باشند، می‌توان با استفاده از کلمه کلیدی elseif برای سایر شقوق مساله نیز شرط تعریف نمود.
برای مثال فرض کنید در تابع قبل می‌خواهیم اگر فاصله دو ورودی ۲ باشد، تابع حاصلضرب آنها را بازگرداند:

```
function res = orderSum(a,b)
    if(abs(a-b)==1)
        res = a+b;
    elseif(abs(a-b)==2)
        res = a*b;
    else
        res = a-b;
    end
end
```

به عنوان مثالی از نتیجه‌ی اجرای برنامه داریم:

```
>> orderSum(2,3)

ans =
     5

>> orderSum(2,4)

ans =
     8

>> orderSum(2,5)

ans =
    -3
```

۲. for: کلمه کلیدی for اولین جز از بخش حلقه‌ها می‌باشد. ساختار کلی for به صورت زیر می‌باشد:

```
for(var=startVal:step:endVal)
    %Code goes here.
end
```

در این ساختار متغیر var تعریف شده، مقدار آغازین startVal به آن اختصاص می‌یابد و سپس با طول گام step تغییر می‌کند و تا زمانی که به مقدار پایانی endVal نرسیده است، این فرآیند و اجرای کدهای داخل قطعه‌اش ادامه می‌یابد.
برای مثال می‌خواهیم اعداد ۳ تا ۸ را چاپ نمائیم:

```
>> for(i=3:1:8)
i
end

i =
     3

i =
     4

i =
     5

i =
     6

i =
     7

i =
     8
end
```

در تکه‌کد بالا، ابتدا در متغیر i مقدار ۳ قرار داده می‌شود، سپس بررسی می‌شود که آیا ۳ کوچکتر یا مساوی ۸ می‌باشد یا خیر، چون کوچکتر است پس قطعه کد بین حلقه -چاپ متغیر- اجرا شده و سپس مقدار متغیر i به اندازه طول گام -در اینجا برابر ۱ واحد- تغییر می‌کند، سپس باز شرط کوچکتر یا مساوی بودن با ۸ بررسی می‌شود و ادامه فرآیند تا زمانی که متغیر مقداری بیشتر از ۸ را انتخاب ننموده است، ادامه می‌یابد.

۳. while: while دومین جز از حلقه‌ها می‌باشد و ساختاری به صورت زیر دارد:

```
while(condition)
    %Code goes here!
end
```

در این ساختار condition یک عبارت می‌باشد که به صورت صحیح یا غلط ارزیابی می‌شود، اجرای حلقه while تا زمان رسیدن به اولین غلط ادامه می‌یابد.

۴. break & continue: در یکی از حلقه‌ها، در صورت استفاده از کلمه کلیدی continue ادامه اجرای بقیه کد داخل حلقه متوقف می‌شود و حلقه به اجرای تکرار بعدی می‌پردازد.

برای مثال می‌خواهیم برنامه‌ای بنویسم که یک عدد نامنفی را گرفته و تمامی اعداد زوج قبل از آنرا چاپ کند:

```
function evenNums(a)
    a = abs(a);
    i = 1;
    while(i<=a)
      if(rem(i,2)~=0)
         i = i+1;
         continue;
      end
      i
      i = i+1;
    end
end
```

به عنوان مثالی از اجرای این تابع داریم:

```
>> evenNums(9)

i =
     2

i =
     4

i =
     6

i =
     8
```

در صورت استفاده از کلمه کلیدی break در حلقه‌ها، متلب در صورت برخورد با آن، اجرای حلقه را به پایان رسانده و به اجرای ادامه کد می‌پردازد.
برای مثال می‌خواهیم برنامه‌ای بنویسم که یک عدد را دریافت کند و اولین عدد قبل از آن، که بر ۳ بخش‌پذیر است را بازگرداند:

```
function number = divThree(a)
    while(a>=0)
      if(rem(a,3)==0)
         number = a;
         break;
      end
      a = a-1;
    end
end
```

به عنوان مثالی از اجرای این برنامه داریم:

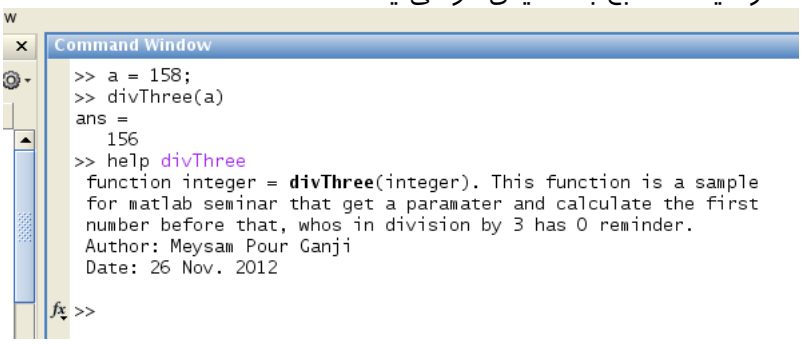
```
>> divThree(158)

ans =
   156
```

د.۵) مستندسازی

برای افزایش خوانایی برنامه در استفاده‌های بعدی و یا توسط اشخاص دیگر، لازم است که در هنگام ذخیره نهایی تابع، اطلاعاتی در مورد تابع، پارامترهای ورودی و مقادیر خروجی، الگوریتم مورد استفاده، نام نویسنده و تاریخ به عنوان مستندات تابع، به آن افزوده شود. محل قرار گرفتن مستندات در دومین خط تابع و بعد از تعریف نام و ویژگی‌های تابع می‌باشد. به مثال زیر توجه کنید:

```
۱  function number = divThree(a)
۲  %function integer = divThree(integer). This function is a sample
۳  %for matlab seminar that get a paramater and calculate the first
۴  %number before that, whos in division by 3 has 0 reminder.
۵  %Author: Meysam Pour Ganji
۶  %Date: 26 Nov. 2012
۷      while(a>=0)
۸        if(rem(a,3)==0)
۹           number = a;
۱۰          break;
۱۱       end
۱۲       a = a-1;
۱۳    end
۱۴ end
```

حال در صورتی که از فرمان help و نام تابع در پنجره فرمان متلب استفاده نمائید، توضیحات تابع به نمایش در می‌آید:



این کار خوانایی و قابل استفاده بودن[۱۶] تابع را بسیار بالاتر می‌برد.

[۱۶] Useability

ذ

دقت متغیرها و
قالب نمایش اعداد

ذ.۱) دقت متغیرها و کلاس‌های ذخیره متغیر

دقت محاسبات در کامپیوتر، با مقدار حافظه‌ی تخصیصی برای ذخیره‌ی عملوندها متناسب است. برای مثال اگر مقداری از حافظه را به یک متغیر اختصاص دهیم که ۴ عدد را ذخیره نماید، آنگاه عدد π در آن محاسبه برابر با ۳.۱۴۱ خواهد بود، اگر حافظه‌ی تخصیصی را به ۷ رقم تغییر دهیم، آنگاه عدد π برابر ۳.۱۴۱۵۹۲ خواهد بود.
برای تعریف دقت یک متغیر، وابسته به زبان برنامه‌نویسی از کلمات کلیدی مختلفی استفاده می‌شود، کلمات مورد استفاده در زبان متلب در لیست زیر آمده است:

- intX: برای ذخیره اعداد صحیح از کلاس intX استفاده می‌شود. این کلاس شما طول بازه‌های متفاوتی است که با جایگزینی X در intX مشخص می‌شوند. در لیست زیر انواع داده‌های اعداد صحیح آمده است:
  - int8: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی -2^7 تا 2^7-1 استفاده می‌شود.
  - int16: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی -2^{15} تا $2^{15}-1$ استفاده می‌شود.
  - int32: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی -2^{31} تا $2^{31}-1$ استفاده می‌شود.
  - int64: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی -2^{63} تا $2^{63}-1$ استفاده می‌شود.

  در صورتی که عددی بیشتر از طول باز باشد، به نزدیک‌ترین کران بازه گرد می‌شود. برای مثال برای نوع int8 بیشترین مقدار قابل ذخیره ۱۲۷ می‌باشد، در صورت ذخیره عدد ۱۲۸ داریم:

```
>> int8(128)
ans =
  127
```

- uintX: برای ذخیره اعداد حسابی[۱۷] از نوع uint استفاده می‌شود. انواع قابل استفاده در این کلاس، با توجه به بازه ذخیره‌سازی در لیست زیر آمده است:
  - uint8: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی صفر تا 2^8-1 استفاده می‌شود.
  - uint16: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی صفر تا $2^{16}-1$ استفاده می‌شود.

[۱۷] اعداد صحیح بدون علامت

- uint32: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی صفر تا $2^{32}-1$ استفاده می‌شود.
- uint64: از این نوع برای ذخیره‌ی اعداد صحیح در بازه‌ی صفر تا $2^{64}-1$ استفاده می‌شود.

در صورتی که عدد ذخیره شده در این نوع، از طول بازه ذخیره‌سازی بیشتر شود، به نزدیک‌ترین کران گرد خواهد شد.

- single: برای ذخیره اعداد اعشاری با دقت معمولی، از کلاس single استفاده می‌شود. این کلاس، اعداد اعشاری در بازه $[1.175494\times10^{-38}, 3.402823\times10^{38}]$ و $[-3.402823\times10^{38}, -1.175494\times10^{-38}]$ را ذخیره می‌نماید.
- double: از کلاس double برای ذخیره اعداد اعشاری با دقت مضاعف استفاده می‌شود. محدوده ذخیره‌سازی اعداد اعشاری در این کلاس، برای اعداد منفی و مثبت، به ترتیب به صورت زیر می‌باشد:

$$[-1.79769\times10^{308}, -2.22507\times10^{-308}]$$
$$[2.22507\times10^{-308}, 1.79769\times10^{308}]$$

در هر دو کلاس single و double در صورتی که مقدار ذخیره شده بیشتر از کران بالا باشد، به مقدار Inf و اگر کمتر از کران پائین باشد به Inf- تبدیل می‌شود.

ذ.۲) خصوصیات یک متغیر

برای بدست آوردن خصوصیات یک متغیر از کلمه کلیدی whos به صورت زیر استفاده می‌شود:

```
whos var_name
```

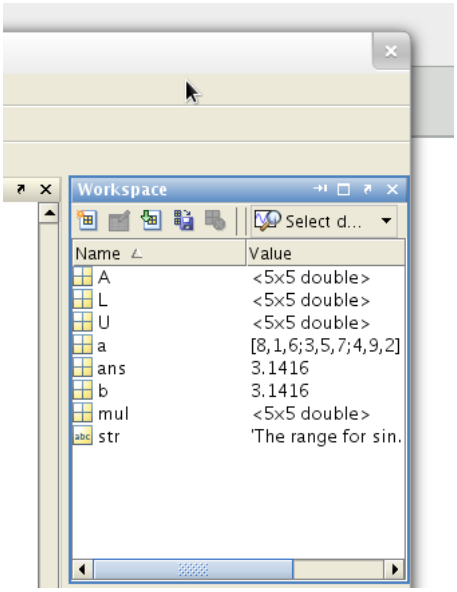
برای مثال داریم:

```
>> a = magic(3);
>> whos a
  Name      Size            Bytes  Class     Attributes

  a         3x3                72  double
```

در صورتی که از کلمه‌ی کلیدی whos به تنهایی استفاده شود، ویژگی‌های تمام متغیرهای تعریف شده را به نمایش در خواهد آورد. برای مثال:

```
>> whos
  Name      Size            Bytes  Class     Attributes

  A         5x5               200  double
  L         5x5               200  double
  U         5x5               200  double
  a         3x3                72  double
  ans       1x1                 8  double
  b         1x1                 4  single
  mul       5x5               200  double
  str       1x54              108  char
```

همچنین در محیط متلب، با استفاده از پنجره محیط کار، می‌توان اطلاعات متغیرهای تعریف شده را مشاهده نمود. این پنجره به معمولا در سمت راست و بالای پنجره متلب قرار دارد:



همچنین این پنجره از مسیر Window → Workspace و یا کلید ترکیبی Ctrl + 3 در دسترس خواهد بود.

ذ.۳) تبدیل نوع متغیرها

برای تبدیل نوع متغیر[۱۸]، از کلمات کلیدی تعریف شده در (ذ.۱) به عنوان تابع استفاده می‌شود. برای مثال داریم:

```
>> a = magic(3);
>> whos a
  Name      Size          Bytes Class    Attributes

  a         3x3              72 double

>> b = uint8(a);
>> whos b
  Name      Size          Bytes Class   Attributes

  b         3x3               9 uint8

>> whos
  Name      Size          Bytes Class    Attributes

  a         3x3              72 double
  b         3x3               9 uint8
```

[۱۸] Variable casting

در صورتی که مقدار متغیر از بازه تعریف کلاس فراتر رود، با آن مطابق با آنچه در (ذ.۱) آمده رفتار می‌شود.

ذ.۴) قالب‌های نمایش اعداد

در یک نوشته، بسته به تقریب موردنیاز برای رساندن دقت موضوع، از تعداد ارقام متفاوتی برای نمایش عدد استفاده می شود. برای مثال در یک روزنامه صبح، در مقاله‌ای در مورد مساحت یکی از میادین شهر، برای عدد π می‌توان از نمایش ۳.۱۴۱۶ استفاده نمود، در حالیکه برای نمایش π در یک مطلب تخصصی شاید نمایش ۳.۱۴۱۵۹۲۶۵۳ چندان مناسب نبوده و باید از نمایش بهتر و دقیق‌تری استفاده نمود.
برای تغییر تعداد ارقام یک نمایش از عدد در متلب، از دستور format به همراه قالب‌های متفاوتی استفاده می‌شود. طریقه‌ی کلی استفاده از دستور format به صورت زیر می‌باشد:

```
format Type
```

که در آن Type کلمه‌ی کلیدی موردنظر، برای تغییر قالب می‌باشد. لیست قالب‌های قابل استفاده در متلب، در زیر آمده است:

- short: نمایش اعداد با چهار رقم بعد از علامت اعشار. برای مثال:

```
>> format short
>> pi
ans =
    3.1416
```

- long: نمایش ۱۵ رقم بعد از علامت اعشار در دقت مضاعف و ۷ رقم بعد از علامت اعشار در دقت معمولی. برای مثال:

```
>> format long
>> single(pi)
ans =
   3.1415927
>> double(pi)
ans =
   3.141592653589793
```

- shortE: نمایش ۴ رقم بعد از علامت اعشار به صورت نماد علمی. برای مثال:

```
>> format shortE
>> pi
ans =
   3.1416e+00
>> pi * 102
ans =
   3.2044e+02
```

- longE: نمایش ۱۵ رقم بعد از علامت اعشار برای دقت مضاعف و ۷ رقم بعد از علامت اعشار برای دقت معمولی با استفاده از نماد علمی. برای مثال:

```
>> format longE
>> single(pi)
ans =
   3.1415927e+00
>> double(pi)
ans =
    3.141592653589793e+00
>> double(pi) * 1368
ans =
    4.297698750110837e+03
```

- shortG: همانند قالب short می‌باشد، اما در نمایش ماتریسی با پهنه‌ی تغییر بزرگ، نمایش خواناتری از اعداد را ارائه می‌دهد. برای مثال:

```
>> a = [0.25 1 5 25365 2369544];
>> format short
>> a
a =
   1.0e+06 *
    0.0000   0.0000   0.0000   0.0254   2.3695
>> format shortG
>> a
a =
        0.25          1          5       25365  2.3695e+06
```

در نمایش shortG در مثال، به جای ضرب کل ماتریس در یک ضریب، هر درایه به صورتی مناسب نمایش داده شده است.

- longG: مانند long می‌باشد، اما در ماتریس‌های با گستره‌ی اعداد بالا، مانند shortG نمایش بهتری از عدد را فراهم می‌کند.
- shortEng: نمایش مهندسی یک عدد به این صورت که ۴ رقم بعد از علامت اعشار را در نماد علمی، با ۳ رقم در توان ۱۰ نشان می‌دهد. برای مثال:

```
>> format shortEng
>> pi
ans =
    3.1416e+000
>> pi * 1368
ans =
    4.2977e+003
```

- longEng: نمایش مهندسی یک عدد با ۱۶ رقم بعد از علامت اعشار و نمایش در نماد علمی با ۳ رقم عدد در توان ۱۰. برای مثال:

```
>> format longEng
>> pi
ans =
   3.14159265358979e+000
>> pi * 1368
ans =
   4.29769875011084e+003
```

ر

اندازه‌گیری زمان اجرا

ر.۱) نمایش زمان اجرا

برای نمایش زمان اجرای یک قطعه کد، آنرا در بین کلمات کلیدی tic و toc قرار می‌دهیم. برای مثال:

```
tic;
%Code goes here!
toc;
```

برای نمونه می‌خواهیم برنامه‌ای بنویسیم که ۱۰ بار مقدار تابع سینوس را از یک تا ۱۰۰ محاسبه نماید و در هر مرتبه محاسبه، زمان اجرا را نمایش دهد. داریم:

```
function sinTime
    for i = 1:10
        tic;
        y = 0;
        for j=1:100
            y = sin(i);
        end
        toc;
    end
end
```

به عنوان اجرایی از این تابع داریم:

```
>> sinTime
Elapsed time is 0.000013 seconds.
Elapsed time is 0.000008 seconds.
Elapsed time is 0.000011 seconds.
Elapsed time is 0.000010 seconds.
Elapsed time is 0.000009 seconds.
Elapsed time is 0.000009 seconds.
Elapsed time is 0.000009 seconds.
Elapsed time is 0.000010 seconds.
Elapsed time is 0.000009 seconds.
Elapsed time is 0.000009 seconds.
```

ر.۲) زمان اجرا به عنوان یک متغیر

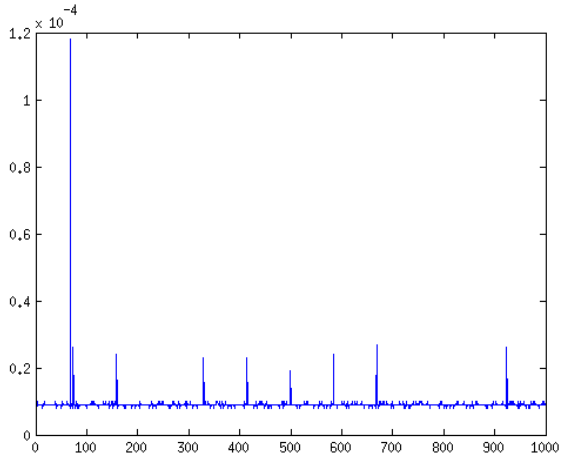
فرض کنید بخواهیم زمان اجرای یک قطعه کد را در یک متغیر ذخیره نمائیم. در این صورت، کلمه‌ی کلیدی toc را به یک متغیر انتصاب می‌دهیم. برای مثال:

```
tic;
%Code goes here!
t = toc;
```

در این صورت به جای چاپ عبارت Elapsed time is x.yyyyyy seconds مقدار x.yyyyyy به متغیر t اختصاص داده می‌شود.
برای مثال می‌خواهیم در برنامه‌ی بخش قبل، ۱۰۰۰ مرتبه فرآیند اجرا شود و سپس نمودار زمان اجرا را رسم نمائیم. داریم:

```
function sinTime
    n = 1000;
    t = zeros(1,n);
    for i = 1:n
        tic;
        y = 0;
        for j=1:100
            y = sin(i);
        end
        t(i) = toc;
    end
    plot(t);
end
```

که نتیجه به صورت زیر خواهد بود:



ز چند پیاده‌سازی

ز.۱) تجزیه LU

یکی از راه‌های حل دستگاه $Ax = b$ استفاده از روش LU می‌باشد. در این روش، ماتریس A به صورت حاصلضرب ماتریس پایین‌مثلثی L و ماتریس بالامثلثی U تجزیه می‌شود و سپس با قرار دادن $y = L^{-1}b$ و بدست آوردن y از آن، دستگاه به صورت $Ux = y$ حل می‌شود.

ز.۱.۱) الگوریتم:

یک رویکرد برای بدست آوردن تجزیه LU ماتریس A، در نظر گرفتن ماتریس A به صورت حاصلضرب دو ماتریس مجهول L و U با خواص موردنظر و حل دستگاه حاصل از آن می‌باشد.
این فرآیند به صورت زیر خواهد بود:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{m1} & l_{m2} & \dots & l_{mm} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1m} \\ 0 & u_{22} & \dots & u_{2m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & u_{mm} \end{bmatrix}$$

$$\Rightarrow \begin{cases} l_{11}u_{11} & & & & = a_{11} \\ l_{21}u_{11} & & & & = a_{21} \\ \vdots & & & & \vdots \\ l_{m1}u_{11} & & & & = a_{m1} \\ l_{11}u_{12} & & & & = a_{12} \\ l_{21}u_{12} & +l_{22}u_{22} & & & = a_{22} \\ \vdots & & & & \vdots \\ l_{m1}u_{12} & +l_{m2}u_{22} & & & = a_{m2} \\ \vdots & & & & \vdots \\ l_{11}u_{1m} & & & & = a_{1m} \\ \vdots & & & & \vdots \\ l_{m1}u_{1m} & +l_{m2}u_{2m} & +\dots & +l_{mm}u_{mm} & = a_{mm} \end{cases} \quad (۱۲)$$

دستگاه حاصل از ضرب ماتریس‌های مجهول در ۱۲ یک دستگاه m^2 معادله و $m^2 + m$ مجهول است. در این صورت طبق دانسته‌های جبر خطی می‌توان برای m مجهول دستگاه مقدار دلخواه اختیار نموده و سپس دستگاه را حل نمود. برای این منظور درایه‌های قطری ماتریس L را یک قرار می‌دهیم، بنابراین دستگاه

معادلات در ۱۲ به این صورت تبدیل می‌شود:

$$
\begin{cases}
u_{11} & = a_{11} \\
l_{21} & = \frac{a_{21}}{u_{11}} \\
\vdots & \vdots \\
l_{m1} & = \frac{a_{m1}}{u_{11}} \\
u_{12} & = a_{12} \\
u_{22} & = a_{22} - l_{21}u_{12} \\
l_{32} & = \frac{a_{32} - l_{31}u_{12}}{u_{22}} \\
\vdots & \vdots \\
l_{m2} & = \frac{a_{m2} - l_{m1}u_{12}}{u_{22}} \\
\vdots & \vdots \\
u_{1k} & = a_{1k} \\
u_{2k} & = a_{2k} - l_{21}u_{1k} \\
\vdots & \vdots \\
u_{kk} & = a_{kk} - \sum_{i=1}^{k-1} l_{ki}u_{ik} \\
l_{k+1\ k} & = \frac{a_{k+1\ k} - \sum_{i=1}^{k-1} l_{k+1\ i}u_{ik}}{u_{kk}} \\
\vdots & \vdots \\
l_{mk} & = \frac{a_{mk} - \sum_{i=1}^{k-1} l_{mi}u_{ik}}{u_{kk}} \\
\vdots & \vdots \\
u_{1m} & = a_{1m} \\
u_{2m} & = a_{2m} - l_{21}u_{1m} \\
\vdots & \vdots \\
u_{mm} & = a_{mm} - \sum_{i=1}^{m-1} l_{mi}u_{im}
\end{cases}
\qquad (۱۳)
$$

بنابراین در یک فرآیند تکراری، ابتدا مجهولات ستون iم ماتریس U را بدست آورده و بعد از آن مجهولات سطر iم ماتریس L را بدست می‌آید.
الگوریتم به صورت زیر خواهد بود:

Data: A$\in M^m(\mathbb{R})$.
Result: L$\in M^m(\mathbb{R})$ as Lower triangular matrix, U$\in M^m(\mathbb{R})$ as Upper triangular matrix such that A = LU.

begin
  $L_{ii} \longleftarrow 1,\ i \in \{1, \ldots, m\}$
  for $i \leftarrow 1$ ***to*** m **do**
    for $j \leftarrow 1$ ***to*** i **do**
      $U_{ji} \longleftarrow A_{ji} - \sum_{k\leftarrow 1}^{i-1} L_{jk}U_{ki}$
    end
    for $j \leftarrow i+1$ ***to*** m **do**
      $L_{ji} \longleftarrow \dfrac{A_{ji} - \sum_{k\leftarrow 1}^{i-1} L_{jk}U_{ki}}{U_{ii}}$
    end
  end
end

ز.۲.۱) پیاده‌سازی:

```
۱  function [L U] = LUDeco(A)
۲  %Function LUDeco. Factorize an square matrix to
۳  %multiplation of two matrix L and U such that L
۴  %is a Lower triangular matrix and U is an Upper
۵  %triangular matrix.
۶  %Usage: [L U]=LUDeco(A);
۷  %input:
۸  %      A: An square matrix.
۹  %Output:
۱۰ %      L: Lower triangular matrix,
۱۱ %      U: Upper triangular matrix.
۱۲     [m n] = size(A);
۱۳     if(m~=n)
۱۴        error('Matrix most be square.');
۱۵     else
۱۶        LTemp = eye(m);
۱۷        UTemp = zeros(m);
۱۸        for i=1:m
۱۹           for j=1:i
۲۰              sumTemp = 0;
۲۱              for k=1:i-1
۲۲                 sumTemp = sumTemp + (LTemp(j,k) * UTemp(k,i));
۲۳              end
۲۴              UTemp(j,i) = A(j,i) - sumTemp;
۲۵           end
۲۶           for j=i+1:m
۲۷              sumTemp = 0;
۲۸              for k=1:i-1
۲۹                 sumTemp = sumTemp + (LTemp(j,k) * UTemp(k,i));
۳۰              end
۳۱              LTemp(j,i) = (A(j,i) - sumTemp) / UTemp(i,i);
۳۲           end
۳۳        end
۳۴        L = LTemp;
```

```
        U = UTemp;
    end
end
```

در برنامه‌ی بالا ابتدا تعداد سطر و ستون ماتریس ورودی محاسبه شده، در صورتی که ماتریس مربعی نباشد (تعداد سطر با ستون برابر نباشد) توسط تابع error یک پیام خطا نمایش داده شده (خط ۱۴) و اجرای برنامه به اتمام می‌رسد، در غیراینصورت، الگوریتم ذکر شده ماتریس‌های مطلوب را محاسبه می‌نماید. در پایان (خطوط ۳۴ و ۳۵) ماتریس‌های به مقادیر معرف برای بازگشت انتصاب داده می‌شوند.
واضح است که این پیاده‌سازی و الگوریتم، از منظر محاسبات عددی، یک روش بهینه نمی‌باشد. برای تمرین سعی کنید الگوریتم تجزیه LU را به گونه‌ای تغییر دهید که یک روش مطلوب محسوب شود، سپس الگوریتم تغییر داده شده را به زبان متلب پیاده‌سازی نمائید.
برای مثالی از اجرای برنامه‌ی بالا داریم:

```
>> A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>> [L U] = LUDeco(A)
L =
    1.0000         0         0         0         0
    1.3529    1.0000         0         0         0
    0.2353   -0.0128    1.0000         0         0
    0.5882    0.0771    1.4003    1.0000         0
    0.6471   -0.0899    1.9366    4.0578    1.0000
U =
   17.0000   24.0000    1.0000    8.0000   15.0000
         0  -27.4706    5.6471    3.1765   -4.2941
         0         0   12.8373   18.1585   18.4154
         0         0         0   -9.3786  -31.2802
         0         0         0         0   90.1734
>> mul = L*U;
>> A==mul
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

در خط ۲۲ برنامه‌ی بالا، ماتریس اصلی و ماتریس حاصل از ضرب ماتریس‌های تجزیه شده مقایسه شده است که با توجه به نتیجه، دو ماتریس برابرند[۱۹].

ز.۲) الگوریتم نویل

در درونیابی‌ها، ممکن است نیازی به محاسبه تابع درونیاب نبوده و تنها بخواهیم مقدار یک نقطه را با استفاده از درونیابی، بدست آوریم، در این صورت الگوریتم

[۱۹] به بخش (د.۳) رجوع شود.

نویل[۲۰] یکی از رویکردهای خوب برای رسیدن به جواب است. در این روش، الگوریتم فرمول چندجمله‌ای درونیاب را به یکباره بدست نمی‌آورد، بلکه ابتدا مساله‌ی درونیابی را برای زیرمجموعه‌هایی از نقاط حامی حل کرده و سپس سعی می‌کند به تدریج جواب مساله را برای کل مجموعه نقاط حامی بدست آورد.

ز.۱.۲) الگوریتم:

گیریم $P_{i_0,i_1,\ldots,i_k}$ چندجمله‌ای از درجه‌ی حداکثر k باشد که نقاط حامی $(x_{i_j}, f_{i_j})_{j=0,\ldots,k}$ را که در آن $\{i_1,\ldots,i_k\} \subseteq \{1,\ldots,n\}$، درونیابی می‌کند. ثابت می‌شود که این چندجمله‌ای‌ها در روابط بازگشتی زیر صدق می‌کنند:

$$\begin{cases} P_i(x) = f_i, & i = 0,\ldots,n \\ P_{i_0,\ldots,i_k}(x) & = \dfrac{(x - x_{i_0})P_{i_1,\ldots,i_k}(x) - (x - x_{i_k})P_{i_0,\ldots,i_{k-1}}(x)}{x_{i_k} - x_{i_0}} \end{cases} \qquad (۱۴)$$

تعریف می‌کنیم: $T_{i+k,k} := P_{i,\ldots,i+k}$. در اینصورت فرمول ۱۴ به صورت زیر در خواهد آمد:

$$\begin{cases} T_{i,0} = f_i \\ T_{i,k} = \dfrac{(x - x_{i-k})T_{i,k-1} - (x - x_i)T_{i-1,k-1}}{x_i - x_{i-k}} \end{cases} \qquad (۱۵)$$

بنابراین الگوریتم به صورت زیر خواهد بود:

Data: X:Vector of suppurt points, F: Vector of support values, x: Point for interpolationg.
Result: f: Interpolated value of x.
begin
  for $i \leftarrow 0$ ***to*** n **do**
    $T[i] \longleftarrow F[i]$
    for $j \leftarrow i - 1$ ***to*** 0 **do**
      $T[j] \longleftarrow T[j+1] + (T[j+1] - T[j]) \times \dfrac{x - X[i]}{X[i] - X[j]}$
    end
  end
  $f \longleftarrow T[0]$
end

ز.۲.۲) پیاده‌سازی:

```
۱ function result = nevilleInt(X,F,x)
۲ %Function nevilleInt. Interpolate a point and
۳ %calculate value without calculating Interpolation
۴ %polynomial.
۵ %Usage: result=nevilleInt(X,F,x);
۶ %input:
۷ %      X: Support points vector.
۸ %      F: Support values vector.
۹ %      x: Value for interpolating.
```

[۲۰] Neville

```
%Output:
%       result: Interpolated value of x.
    lX = length(X);
    lF = length(F);
    if(lX ~= lF)
        error('X and F most be same size.');
    else
        T = zeros(1,lX);
        for i=1:lX
            T(i) = F(i);
            for j=i-1:-1:1
                xTemp = (x - X(i))/(X(i)-X(j));
                xTemp = xTemp * (T(j+1)-T(j));
                T(j) = T(j+1) + xTemp;
            end
        end
        result = T(1);
    end
end
```

به عنوان مثالی از اجرای این تابع داریم:

```
>> X = [0.1 0.2 0.3 0.4 0.5];
>> F = [-1.6228 -0.8218 -0.3027 0.1048 0.4542];
>> f = nevilleInt(X,F,0.15)
f =
   -1.1719
```