*بخش دوم/ آشنایی با مفاهیم ریاضی در MATLAB:

ماتریس در MATLAB :

همانطور که قبلا نیز اشاره شد ، کار با ماتریس ها از مزایای مهم نرم افزارMATLAB است. یک ماتریس می تواند نماینده ی یک گراف، مدار، معادله، بازه و بسیاری دیگر از مفاهیم مهم ریاضیات و دیگرعلوم باشد و MATLAB از این مزیت برای تبدیل انواع مفاهیم علوم مختلف به toolboxهای کاربردی خود بهره برده است.

ماتریس ها بنا به موارد استفاده از آنها به طرق مختلف در MATLAB تعریف می شوند. اولین و ساده ترین راه ، تعریف مستقیم ماتریس است، به این معنا که تمامی درایه های ماتریس نوشته می شود. به عنوان مثال یک ماتریس دو در دو را در نظر می گیریم که درایه های سطر اول آن ۱ و ۲ و درایه های سطر دوم ۳ و ۴ هستند. این ماتریس که نام آنرا A میگذاریم اینگونه تعریف می شود:

```
>>A=[1 2;3 4]
```

گاهی ماتریس ما از درایه هایی تشکیل شده است که با یک تصاعد حسابی در کنار هم قرار گرفته اند. مثلا ماتریسB یک ماتریس یک سطری است که درایه ی اول آن 1 و درایه ی آخر آن 10 است و درایه های آن با گام ۳ افزایش می یابند. یعنی ماتریسB برابر است با [1 4 7 10] . این ماتریس می تواند به شکل زیر تعریف شود:

```
>>B=[1:3:10]
```

و یا <= `>>B=1:3:10`

اگر گام یک ماتریس ۱ باشد، می توان از نوشتن آن صرف نظر کرد. مثلا ماتریس[1 2 3 4 5 6 7 8]=C را مینویسیم:[1:8]=C و یا C=1:8 .

گام این بازه ها می تواند مثبت، منفی، عدد صحیح و یا یک عدد حقیقی باشد، اما نمی تواند موهومی باشد.

ما همچنین می توانیم یک ماتریس سطری را با بیان درایه ی اول و آخر و تعداد درایه هایی که می خواهیم این بازه داشته باشد تعریف کنیم. فرض کنید ماتریس D یک بازه است که درایه ی اول آن ۱ و درایه ی آخر آن ۳ باشد و ما میخواهیم این بازه ۱۰ درایه داشته باشد. در چنین حالتی که تشخیص گام تصاعد مشکل است از تابع linspace استفاده میکنیم. در این مثال D را با استفاده از این تابع اینگونه تعریف می کنیم :

```
>>D=linspace(1,3,10)
```

همانطور که می بینید اولین عنصرِ ورودی، درایه ی اول، دومی درایه ی آخر و ورودیِ سوم تعداد درایه هاست. حاصل اجرای این دستور اینگونه خواهد بود :

```
>> D=linspace(1,3,10)

D =

   Columns 1 through 9

     1.0000    1.2222    1.4444    1.6667    1.8889    2.1111    2.3333    2.5556    2.7778

   Column 10

     3.0000
```

ما در تعریف یک ماتریس می توانیم از روش مستقیم و بازه ای و یا حتی از یک ماتریس دیگر توأماً نیز استفاده کنیم.

مثلاً ماتریس E را در نظر بگیرید که در روش مستقیم به صورت E=[1 2 3 4 5 7;1 1 4 7 10;1 1.75 2.5 3.25 4] تعریف می شود. این ماتریس که دارای سه سطر و پنج ستون است، می تواند به اشکال زیر نیز تعریف شود:

```
>> E=[1:5 7;1 1:3:10;1:0.75:4]
```

و یا : >> E=[C(1:4) 7;1 B;linspace(1,4,5)]

عبارت C(1:4) به معنای درایه ی اول تا چهارم از ماتریس C است. در شمارش درایه های یک ماتریس همیشه به صورت ستونی عمل می شود؛ بدین مفهوم که درایه ی اول، درایه ی اول ستون یک است، درایه ی دوم، دومین درایه از ستون یک، درایه ی سوم، سومین درایه از ستون اول و به همین ترتیب شمارش تا آخرین درایه ی ستون یک ادامه می یابد و پس از آن به سراغ درایه ی اول از ستون دوم خواهیم رفت و تا آخرین درایه ی این ستون می شماریم تا این ستون نیز تمام شود و سپس به سراغ ستون سوم، چهارم الی آخر می رویم. در این مثال چون ماتریس سطری است و در واقع در هر ستون یک درایه بیشتر وجود ندارد ، به نظر می رسد که شمارش سطری است، اما در موارد دیگر که ماتریس بیش از یک سطر دارد مشخص می شود که روند کار به همان شکل است که توضیح داده شد. بنابراین مثلاً در ماتریس E درایه ی E(6) برابر 1.75 خواهد بود. همچنین E(2,3) به معنای درایه ی سطر دوم از ستون سوم ماتریس E می باشد که مقدار آن ۴ است. شما حتی می توانید با ترتیب خاصّی بخشی از یک ماتریس را در ماتریس دیگری قرار دهید. مثلا ماتریس F را به شکل زیر تعریف نموده و حاصل را می بینیم:

```
>> F=E(:,1:2:end)
F =
   1.0000   3.0000   7.0000
   1.0000   4.0000  10.0000
   1.0000   2.5000   4.0000
```

همانطور که مشاهده می فرمایید ماتریس F از درایه های موجود در ستون های اول، سوم و آخر ماتریس E تشکیل شده است. علامت : که به تنهایی در سمت چپ ویرگول داخل پرانتز گذاشته شده است به مفهوم درایه های همه ی سطرهاست و بخش سمت راست ویرگول هم به این معناست که از اولین تا آخرین ستون را با گام ۲ جدا کن. اگر ماتریس E شش ستون داشت، با این دستور باز هم ستون های اول، سوم و پنجم جدا می شدند و در F قرار می گرفتند؛ پس end اینجا به این معنا نیست که حتما ستون آخر هم باید باشد، بلکه مفهومش اینست که تا آخرین ستون ماتریس E ، هر ستونی که می تواند با این گام جدا شود جزء ماتریس F باشد.

تابع دیگری که می توان از آن برای مشخص کردن یک آرایه سطری با تصاعد نمایی استفاده کرد، تابع logspace است. این تابع بر پایه ی توان عدد ده عمل می کند.به این صورت که توان اولین و آخرین عدد و تعداد درایه های ماتریس را از شما می گیرد و به صورت نمایی بین اولین و آخرین عدد را تقسیم بندی می کند. مثلا اگر بنویسید G=logspace(0,2,3) ، درایهٔ اول ده به توان صفر برابر یک و درایه ی سوم ده به توان دو برابر صد خواهد بود و چون سه عدد داریم و بین صفر و دو عدد یک است، پس درایه ی وسط ده به توان یک برابر ده خواهد شد. به مثال دیگر توجه کنید:

```
>> H=logspace(0,3,6)
H=
   1.0e+003 *
     0.0010   0.0040   0.0158   0.0631   0.2512   1.000
```

همانطور که مشاهده می کنید چون تقسیم بندی توانی به طور دقیق امکان نداشته است با اندکی تقریب این کار صورت گرفته و به طور تصاعد نمایی ماتریس H به وجود آمده است. در واقع می توان گفت که درایه های ماتریس حاصل از logspace(a,b,c)، ده به توان درایه های ماتریس linspace(a,b,c) می باشد که با تقریب خوبی محاسبه شده است. چگونگی نوشته شدن پاسخ خروجی تابع در این مثال نیز قابل توجه است.

اگر بنویسید logspace(a,b) ، ماتریس حاصل بر اساس تعداد پیش فرض ۵۰ درایه تشکیل می شود. اگر بنویسیدlogspace(a,pi,n) حاصل ماتریس نمایی با n درایه خواهد بود که درایه ی اول 10^a و درایه ی آخر تنها در این حالت خاص عدد پی(π=3.1416) است. نکته ی دیگری که در خصوص توابع logspace و linspace قابل ذکر است اینکه اگر n کوچکتر از ۲ باشد، فقط درایه ی آخر(b یا 10^b یا pi) به عنوان تنها درایه ی ماتریس حاصله در نظر گرفته می شود و اگر n یک عدد غیر صحیح باشد (اعشار داشته باشد)، نرم افزار فقط قسمت صحیح آنرا در نظر گرفته و در واقع آنرا گرد می کند.

اگر بنویسید J=F(:) ماتریس J ماتریسی تماماً ستونی است که از درایه های اول تا آخر ماتریس F تشکیل شده است.

ماتریس های خاص:توابعی وجود دارد که انواع خاصی از ماتریس ها را تولید می کنند. برخی از این توابع عبارتند از:

ones(n) : این تابع یک ماتریس مربعی n در n تولید می کند که تمام درایه های آن عدد یک است. اگر بخواهیم ماتریس غیر مربعی داشته باشیم کافیست ابعاد آنرا به عنوان ورودی به این تابع بدهیم. به عنوان مثال ones(a,b) یک ماتریس با درایه های برابر یک تولید می کند که دارای a سطر و b ستون است. شما همچنین می توانید ماتریسی هم بعد با یک ماتریس دلخواه بسازید که همه ی درایه هایش یک باشد؛ بدین منظور اگر مثلاً بخواهیم ماتریس با ماتریس A هم بعد باشد مینویسیم :

. ones(size(A))

zeros : این تابع روند کاری مانند تابع ones دارد، با این تفاوت که تمام درایه های آن به جای یک، صفر است.

eye : از این تابع برای ساخت یک ماتریس همانی استفاده می شود. ماتریس همانی ماتریسی است که درایه های قطر اصلی آن یک و ما بقی درایه هایش صفر است. ماتریسones(n) یک ماتریس مربعی و n در n می باشد. به دو مثال زیر توجه کنید:

```
>> eye(4)
ans =
     1   0   0   0
     0   1   0   0
     0   0   1   0
     0   0   0   1

>> eye (3,4)
ans =
     1   0   0   0
     0   1   0   0
     0   0   1   0
```

همانطور که ملاحظه کردید ماتریس همانی eye لزوماً همیشه مربعی نیست و می توان یک ماتریس غیر مربعی نیز تعریف کرد که در واقع با حذف سطر یا سطر هایی از پایین یک ماتریس همانی مربعی و یا با حذف ستون یا ستونهایی از سمت راست آن به دست آمده است.

rand : این تابع با توجه به داده های ورودی آن ، یک ماتریس با درایه های اتفاقی تولید می کند و درایه های آن اعدادی بین صفر و یک است. اصول عملکرد تابع نسبت به تعداد ورودی های آن همانند ماتریس های حاصله از توابع قبلی است. به دو مثال زیر توجه کنید:

```
>> rand(3)
ans=
    0.9167   0.1056   0.9001
    0.0232   0.7890   0.0012
    0.3222   0.9797   0.5489
>> rand(2,3)
ans=
    0.3800   0.6934   0.3402
    0.2396   0.0010   0.8944
```

نتیجه ی اجرای این تابع یک ماتریس کاملاً اتفاقی است و هر بار اجرای این تابع برای ورودی های یکسان یک نتیجه ی جدید و متفاوت می دهد.

magic : این تابع نوعی ماتریس خاص تولید می کند که مجموع درایه های آن در هر سطر و هر ستون با هم برابر است. این تابع تنها یک ورودی می پذیرد و حاصل آن همیشه یک ماتریس مربعی با بعد عدد صحیحی است که به عنوان ورودی به آن داده ایم. مثال زیر را ملاحظه بفرمایید.دقت کنید که در این مثال که یک ماتریس مربعی 3 در3 است مجموع درایه ها در هر سطر و هر ستون برابر ۱۵ است :

```
>>magic(3)
ans=
    8  1  6
    3  5  7
    4  9  2
```

pascal : این تابع از توابع بسیار جالب MATLAB در تولید ماتریس های خاص می باشد. با استفاده از این تابع سه ماتریس با خواص منحصر به فرد می توان بوجود آورد. pascal(n) یک ماتریس مربعی با بعد n می سازد که همان مربع منسوب به پاسکال است. از خواص این ماتریس مربعی آنکه نسبت به قطر اصلی متقارن می باشد و درایه ی (a,b) برابر با مجموع درایه های(a,b-1) و (a-1,b) است. به مثال زیر توجه کنید:

```
>> pascal(4)
ans =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
```

همانطور که می بینید به عنوان مثال درایه ی سطر سوم از ستون چهارم(10) با مجموع درایه های سطر سه ستون سه(6) و سطر دو ستون چهار(4) برابر است. . از دیگر خواص این ماتریس آنست که همواره دترمینانی برابر یک دارد.

شکل بعدی استفاده از تابع pascal با استفاده از دستور pascal(n,1) تعریف می شود. ماتریس حاصله ماتریس مربعی با بعدn است و از خواص آن می توان به پایین مثلثی بودن، وجود درایه های 1 و1- به صورت یکی در میان روی قطر اصلی و برابر بودن معکوس ماتریس با خود ماتریس اشاره کرد. همچنین درایه ی(a,b) برابر با درایه ی(a-1,b) منهای (a-1,b-1) است. به نمونه ی زیر به ازای n=5 دقت کنید.

```
>> pascal(5,1)
ans =
     1     0     0     0     0
     1    -1     0     0     0
     1    -2     1     0     0
     1    -3     3    -1     0
     1    -4     6    -4     1
```

شکل دیگر استفاده از تابع پاسکال، pascal(n,2) است. ماتریس خروجی این تابع نیز یک ماتریس مربعی می باشد. دترمینان این ماتریس و هر توان صحیحی از آن همواره برابر یک است. در ضمن این ماتریس، ریشه ی سوم ماتریس همانی همبعد با خود نیز است . به مثال زیر توجه نمایید :

```
>> K=pascal(4,2)
K =
    -1    -1    -1    -1
     3     2     1     0
    -3    -2     0     0
     1     0     0     0
```

```
>> K^3
ans=
     1    0    0    0
     0    1    0    0
     0    0    1    0
     0    0    0    1
```

در اینجا K^3 به معنای سه بار ضرب ماتریسی ماتریس K در خود است.

کار روی ماتریس ها:

توابع مختلفی برای کار روی ماتریس ها وجود دارد که به برخی از آنها اشاره می کنیم :

det : از این تابع برای محاسبه ی دترمینان یک ماتریس مربعی استفاده می شود. مثال زیر گویای چگونگی به کارگیری این تابع است:

```
>> L=pascal(3)
L =
   1    1    1
   1    2    3
   1    3    6
>> det(L)
ans =
     1
```

inv : این تابع برای محاسبه ی ماتریس معکوس استفاده می شود. به عنوان مثال برای ماتریس L در مثال قبل ، داریم :

```
>> inv(L)
ans =
     3    -3     1
    -3     5    -2
     1    -2     1
```

در مثالی دیگر می توانیم معکوس ماتریس حاصل از تابع pascal(5,1) را محاسبه کنیم و ببینیم آیا همانطور که انتظار می رود معکوس آن با خودش برابر است یا خیر:

```
>> M=pascal(5,1)
M =
     1     0     0     0     0
     1    -1     0     0     0
     1    -2     1     0     0
     1    -3     3    -1     0
     1    -4     6    -4     1
```

```
>> inv(M)
ans =
     1     0     0     0     0
     1    -1     0     0     0
     1    -2     1     0     0
     1    -3     3    -1     0
     1    -4     6    -4     1
```

length و size : از این دو تابع برای بدست آوردن ابعاد یک ماتریس استفاده می شود. تابع length برای یافتن بزرگترین بعد یک ماتریس و تابع size برای بدست آوردن کلیه ی ابعاد ماتریس به کار می رود. از این دو تابع در موارد بسیاری جهت ایجاد انواع ماتریس های همبعد با یک ماتریس معلوم بهره می بریم.

به مثالهای زیر توجه فرمایید :

```
>> N=[1 2 3;4 5 6]
N =
    1     2     3
    4     5     6
>> length(N)
ans =
     3
>> size(N)
ans =
     2     3
>> eye(size(N))
ans =
     1     0     0
     0     1     0
>> O=1:0.5:3
O =
    1.0000    1.5000    2.0000    2.5000    3.0000
>> P=linspace(1,10,length(O))
P =
    1.0000    3.2500    5.5000    7.7500   10.0000
```

complex : اگر دو ماتریس هم بعد مانند Q و R داشته باشیم و بخواهیم ماتریس S را بدست آوریم که درایه های آن مختلط و قسمت حقیقی آنها درایه ی نظیر از ماتریسQ و قسمت موهومی درایه ی نظیر از ماتریس R باشد، از این تابع استفاده میکنیم . عملکرد تابع را در یک مثال نشان می دهیم:

```
>> Q=[1 2;3 4];
>> R=[5 6;7 8];
>> S=complex(Q,R)
S=
   1.0000 + 5.0000i    2.0000 + 6.0000i
   3.0000 + 7.0000i    4.0000 + 8.0000i
```

sum : این تابع دو حالت دارد. اگر T یک ماتریس دو بعدی باشد،sum(T,1) یا sum(T) یک ماتریس سطری خواهد بود که به اندازه ی تعداد ستون های ماتریسT درایه دارد و هر درایه ی آن برابر با مجموع درایه های ستون نظیر از ماتریسT است. sum(T,2) یک ماتریس ستونی تولید میکند که به تعداد سطر های ماتریس T درایه دارد و هر درایه ی آن برابر با مجموع درایه های سطر نظیر از ماتریس T می باشد. مثال زیر مسئله را بهتر روشن می سازد:

```
>> T=[1 2;3 4]
T=
   1    2
   3    4
>> sum(T,1)

ans=
     4     6

>> sum(T,2)

ans=
     3
     7
```

عملگرهای ریاضی :

+ : عملگر جمع ریاضی و جمع ماتریس هاست. در مورد ماتریس ها تنها زمانی می توان از آن استفاده کرد که دو ماتریس همبعد باشند و در این حالت حاصل جمع دو ماتریس، یک ماتریس با همان ابعاد و درایه های آن مجموع درایه های متناظر است.

- : عملگر تفریق ریاضی و تفریق ماتریس هاست و همانند عملگر جمع در مورد ماتریس ها بین درایه های دو ماتریس همبعد عمل می کند.

* : عملگر ضرب ریاضی است و در مورد ماتریس ها به عنوان عملگر ضرب ماتریسی استفاده می شود و در این حالت باید تعداد ستون های ماتریس سمت چپ علامت با تعداد سطرهای ماتریس سمت راست آن برابر باشد و حاصلضرب ماتریسی است با تعداد سطر برابر با تعداد سطور ماتریس سمت چپ عملگر و تعدادستونی برابر با ماتریس سمت راست آن. مثال :

```
>> U=[1 2 3;4 5 6];
>> V=[1 2 3 4;5 6 7 8;9 10 11 12];
>> U*V
ans=
      38    44    50    56
      83    98   113   128
```

*. : این عملگر نیز برای ضرب استفاده می شود، با این تفاوت که بین دو ماتریس همبعد قرار گرفته و درایه ها را نظیر به نظیر در هم ضرب می کند و حاصلضرب یک ماتریس همبعد با ماتریس های ضرب شده است. مثال:

```
>>A=[1 2 3];
>>B=[4 5 6];
>>A.*B
ans=
     4    10    18
```

' و '. : این عملگرها ماتریس ترانهاده را به ما می دهند و تفاوت آنها زمانی آشکار می شود که ماتریس دارای درایه هایی از نوع اعداد مختلط باشد، زیرا ' هم این ماتریس را ترانهاده نموده و هم مزدوج اعداد مختلط را به جای آنها قرار می دهد، در حالی که '. فقط ماتریس را ترانهاده می کند و تغییری در درایه ها ایجاد نمی کند. در مورد ماتریسی که درایه های حقیقی داشته باشد نتیجه ی استفاده از دو عملگر یکسان است. اگر عملگر ' روی یک عدد مختلط عمل کند حاصل آن مزدوج عدد خواهد بود. به مثالهای زیر توجه کنید:

```
>>a=[1 1+i*1];b=1+i*1;
>>a'
ans=
      1.0000
      1.0000  -  1.0000i
>>a.'
ans=
     1.0000
     1.0000  + 1.0000i
>>b'
```

```
ans=
      1.0000  -  1.0000i
```

^ و ^. : از این عملگرها برای به توان رساندن استفاده می شود. عملگر ^ بیشتر برای یک عدد استفاده میشود و عدد سمت چپ خود را به توان عدد سمت راست می رساند. استفاده از این عملگر جلوی یک ماتریس زمانی مجاز است که ماتریس مربعی باشد و معنایش آنست که آن ماتریس به تعداد رقمی که سمت راست عملگر است باید در خود ضرب ماتریسی شود. عملگر ^. که بر روی ماتریس ها عمل می کند برای به توان رساندن درایه های ماتریس استفاده می شود و تک تک درایه های ماتریس را به توان عدد سمت راست خود می رساند.مثالهای زیر طریقه ی عملکرد این عملگرها را بهتر روشن میسازد:

```
>>A=[1 2;3 4];
>>A^2
ans=
      7    10
     15    22
>>A.^2
ans=
      1     4
      9    16
```

عملگرهای ریاضی دارای نوعی اولویت بندی هستند که رعایت آن در نوشتن عبارات ریاضی لازم است. به عنوان مثال در نوشتن یک چند جمله ای اولین اولویت با عبارات داخل پرانتز، سپس با محاسبه ی توابع و جایگزین نمودن مقدار آنها به جای هر تابع، بعد توان و پس از آن ضرب یا تقسیم که اولویت یکسانی دارند و اولویت بین آن دو با علامتی می باشد که از سمت چپ در اولویت است. آخرین اولویت نیز برای جمع یا تفریق است که باز بستگی به اولویت قرارگیری هر یک در جمله دارد. به طور خلاصه اولویت ها به ترتیب پرانتز، توابع، توان، ضرب یا تقسیم و جمع یا تفریق است.مثال:

```
>> a=2;b=3;c=4;d=5;
>> a+b-c       %2+3-4
ans=
      1
>> a*b/c*d       %((2*3)/4)*5
ans=
      7.5000
>> a*b/(c*d)   %(2*3)/(4*5)
ans=
      0.3000
>> a*b^c/d       %(2*(3^4))/5
ans=
      32.4000
>>a*b+c^d/a+b     %((2*3)+((4^5)/2))+3
ans=
     521
```

آشنایی با برخی توابع ریاضی:

قبل از بیان این توابع قرارداد می کنیم که x ، y و z به عنوان اعداد حقیقی، a ، b و c اعـداد صـحیح و n ، m و k اعـداد طبیعی محسوب می شوند و به جای اعداد مختلط از Xc ، Yc و Zc استفاده می کنیم.

الف) توابع مثلثاتی: این توابع که می توان آنها را در سه گروه اصلی تقسیم بندی نمود عبارتند از:

۱- توابع مثلثاتی ساده برای زوایای مثلثاتی بر حسب رادیـان : این توابع که نام آنها از سه حرف نام هر تابع مثلثاتی تشکیل شـده است با زوایایی کار می کنند که بر حسب رادیان باشد. اگر خود تابع باشد (مثل sin یا tan) ورودی آن بر حسب رادیـان، و اگر معکوس تابع باشد (این توابع با اضافه کردن حرف a به نام تابع مستقیمـشان شـناخته مـی شـوند، مثـل asin و atan) خروجی آن بر حسب رادیان خواهد بود. جدول و مثالهای زیر بیانگر انواع این توابع است:

نام M-File	عملکرد
sin(x)	سینوس زاویه ی x بر حسب رادیان
cos(x)	کسینوس زاویه ی x بر حسب رادیان
tan(x)	تانژانت زاویه ی x بر حسب رادیان
cot(x)	کتانژانت زاویه ی x بر حسب رادیان
sec(x)	سکانت زاویه ی x بر حسب رادیان sec(x)=1/cos(x)
csc(x)	کسکانت زاویه ی x بر حسب رادیان csc(x)=1/sin(x)
asin(x)	معکوس سینوس عدد x ، نتیجه بر حسب رادیان
acos(x)	معکوس کسینوس عدد x ، نتیجه بر حسب رادیان
atan(x)	معکوس تانژانت عدد x ، نتیجه بر حسب رادیان
acot(x)	معکوس کتانژانت عدد x ،نتیجه بر حسب رادیان
asec(x)	معکوس سکانت عدد x ، نتیجه بر حسب رادیان
acsc(x)	معکوس کسکانت عدد x ، نتیجه بر حسب رادیان

```
>> sin(pi/2)
ans=
     1
>> acos(1)
ans=
     0
>> csc(pi/6)
ans=
      2.0000
>> atan(inf)
ans=
     1.5708
```

۲- توابع مثلثاتی ساده برای زوایای مثلثاتی بر حسب درجه:نام این توابع از همان نام توابع مثلثاتی برحسب رادیان بـه اضـافه ی حرف d در انتهای آن ساخته می شود (مثل sind، asind) و تفاوت آن برای توابع مستقیم در ورودی تابع و بـرای توابـع معکوس در خروجی تابع است. به مثال های زیر توجه کنید و آنها را با مثالهای قسمت قبل مقایسه کنید.

```
>> sin(90)
ans=
    1
>> acos(1)
ans=
    0
>> csc(30)
ans=
     2.0000
>> atan(inf)
ans=
    90
```

۳- توابع مثلثاتی هیپربولیک برای زوایای مثلثاتی بر حسب رادیـان : این توابع که از تعاریف زیر پیروی می کننـد، از نـام توابـع مثلثاتی ساده بر حسب رادیان به علاوة حرف h در انتهای آن ساخته می شود. این توابع عبارتند از:

```
sinh(x)=(exp(x)-exp(-x))/2
cosh(x)=(exp(x)+exp(-x))/2
tanh(x)=sinh(x)/cosh(x)
coth(x)=1/tanh(x)
sech(x)=1/cosh(x)
csch(x)=1/sinh(x)
```

معکوس این توابع نیز با اضافه کردن حرف a به ابتدای آنها ساخته می شود؛ مثل asinh(x) . مثال:

```
>>sinh(pi)
ans=
    11.5487
>>asinh(11.5487)
ans=
    3.1416
>>atanh(1)
ans=
    Inf
>>coth(-inf)
ans=
     -1
```

ب) توابع نمایی و توانی : این توابع زمانی استفاده می شوند که بخواهیم از یک عدد لگاریتم بگیریم و یا آنرا بـه تـوان برسانیم و یا اینکه ریشه ی توانی یک عدد را به دست آوریم. جدول و مثال های زیر تعدادی از توابع پرکاربرد از این دسـت توابع را نشان می دهد.

نام M-File	عملکرد
exp(x)	عدد نپر(e) را به توان x می رساند
log(x)	لگاریتم طبیعی بر پایه ی عدد نپر(Ln x)
log10(x)	لگاریتم x بر پایه ی عدد 10
log2(x)	لگاریتم x بر پایه ی عدد 2
sqrt(x)	محاسبه ی جذر x
power(x,y)	x را به توانy می رساند. در مورد ماتریس ها معادل با x.^y است.
nthroot(x,n)	ریشه ی nام x را بدست می آورد. معادل x.^(1/n) است.

همانطور که مشاهده فرمودید ما تنها لگاریم بر سه پایه ی e ، 10 و 2 را در اختیار داریم. فرض کنید مـی خـواهیم لگـاریتم عدد x را برپایه ی 3 محاسبه کنیم. تمامی روابط زیر می تواند این مقدار را محاسبه کند:

```
log(x)/log(3)
log10(x)/log10(3)
log2(x)/log2(3)
```

در ریاضیات به طور کلی از تقسیم لگاریتم x برپایه ی a ، بر لگاریتم b بر پایه ی a، لگاریتم x بر پایه ی b محاسبه میشود.

حال به مثال های زیر توجه کنید.

```
>> exp(2)      %e^2
ans=
     7.3891
>> log(2.7183)
ans=
     1.0000
>> log10(1000)
ans=
     3
>> sqrt(-100)
ans=
          0 +10.0000i
```

```
>> power(2,5)
ans=
     32
>> nthroot([27 216;0.729  -1000],3)
ans=
    3.0000        6.0000
    0.9000      -10.0000
>> log2(81)/log2(3)
ans=
    4
```

ج) توابع لازم برای گردکردن : گاهی لازم است که یک عدد غیر صحیح را به یک عدد صحیح گرد کنیم. این عدد صحیح میتواند عدد صحیح بزرگتر از آن و یا عدد صحیح کوچکتر باشد. توابع زیر هر یک به طریقی این کار را انجام میدهند.

نام M-File	عملکرد
ceil(x)	x را به عدد صحیح بزرگتر گرد می کند.
floor(x)	x را به عدد صحیح کوچکتر گرد می کند.
round(x)	بدون توجه به علامت x، آن را به عدد صحیحی که از لحاظ قدرمطلق به آن نزدیکتر است گرد می کند.
fix(x)	قسمت اعشار عدد را حذف نموده و بخش صحیح آن را بر می دارد.

به عنوان مثال فرض کنید ماتریس A که حاوی تعدادی عدد غیر صحیح است را داشته باشیم. توابع فوق روی این ماتریس عمل نموده و نتایج زیر بدست می آیند:

```
>> A=[-1.9 -1.5 -1.1 -0.01 0.01 1.1 1.5 1.9]
A=
   -1.9  -1.5  -1.1  -0.01   0.01   1.1   1.5   1.9
>> ceil(A)
ans=
    -1    -1    -1     0     1     2     2     2
>> floor(A)
ans=
    -2    -2    -2    -1     0     1     1     1
>> round(A)
ans=
    -2    -2    -1     0     0     1     2     2
>> fix(A)
ans=
    -1    -1    -1     0     0     1     1     1
```

از دیگر توابع ریاضی که در همین بخش به آن اشاره میکنیم sign است. این تابع به ازای اعداد مثبت عدد 1 و به ازای اعداد منفی عدد 1- را بر می گرداند. به ازای 0 همان عدد 0 خروجی تابع خواهد بود. این تابع را روی ماتریس A اعمال میکنیم:

```
>>sign(A)
ans=
     -1    -1    -1    -1     1     1     1     1
```

د) توابع اعداد طبیعی: این توابع تنها برای اعداد طبیعی مفهوم درستی دارد و برای سایر اعداد یا اساساً عمل نمی کند و یا مفهوم متفاوتی خواهد داشت. برخی توابع اعداد طبیعی عبارتند از:

نام M-File	عملکرد
gcd(m,n)	بزرگترین مخرج مشترک(ب م م) n و m را محاسبه می کند.
lcm(m,n)	کوچکترین مضرب مشترک(ک م م) n و m را محاسبه می کند.
primes(n)	اعداد اول طبیعی از ۲ تا n را در قالب یک ماتریس سطری بر می گرداند.
factorial(n)	n! را محاسبه می کند.
factor(n)	n را به عوامل اول تجزیه، و این عوامل را در قالب یک ماتریس سطری بر می گرداند.
rem(m,n)	باقیمانده ی تقسیم عدد m بر n را بدست می آورد.

در اینجا مثال هایی برای توابع اعداد طبیعی ارائه شده است :

```
>> gcd(30,40)
ans=
     10
>> lcm(12,15)
ans=
    60
>> primes(20)
ans=
    2     3     5     7    11    13    17    19
>> factorial(5)
ans=
   120
>> factor(156)
ans=
    2     2     3    13
>> rem(11,3)
ans=
     2
```

هـ) توابع اعداد مختلط: این توابع همواره بر روی اعداد مختلط کار می کنند، هر چند ممکن است در مورد سـایر اعـداد نیز به کار روند که در اینصورت معنایی متناسب با آن نوع اعداد خواهند داشت. چند مورد پر کاربرد این توابع عبارتند از:

نام M-File	عملکرد
abs(Xc)	اندازه ی عدد مختلط را بدست می آورد.اندازه یXc=x+i*y برابر است باx^2+y^2√
angle(Xc)	زاویه ی عدد مختلط را بر حسب رادیان بدست می آورد.زاویه یXc=x+i*y برابر است atan(y/x)
imag(Xc)	قسمت موهومی Xc را بدست می آورد.
real(Xc)	قسمت حقیقی Xc را بدست می آورد.
atan2(imag(Xc),real(Xc))	قسمت حقیقی و موهومی یک عدد مختلط را گرفته و زاویه آن در ربع اول را بر حسب رادیان پیدا میکند.
conj(Xc)	مزدوج عدد مختلط Xc را بر می گرداند.

به عنوان مثال عدد مختلط Yc را تعریف نموده و توابع فوق را روی آن اعمال می کنیم. به چگونگی تعریف یک عدد مختلط نیز توجه کنید.

```
>>Yc=3+4*i
Yc=
    3.0000  +  4.0000i
>>abs(Yc)    %(3^2+4^2)^0.5
ans=
     5
>>real(Yc)
ans=
     3
>>imag(Yc)
ans=
    4
>>angle(Yc)
ans=
    0.9273
>>atan2(4,3)
ans=
    0.9273
>>conj(Yc)
ans=
    3.0000  -  4.0000i
```

Yc را می توانستیم به صورت Yc=3+4*j نیز تعریف کیم که البته نتیجه کاملاً یکسان خواهد بود.

**توابع ریاضی MATLAB بسیار گسترده و متنوع است. من در اینجا به همین چند تابع پر کاربرد اکتفا می کنم.

عبارات و توابع پارامتری یا Symbolic:

در ریاضیات ما اغلب نیاز داریم عبارات چند جمله ای و یا متغیرهای پارامتری ایجاد کنیم و قبل از هر گونه مقدار دهی و حل عددی، روی آنها به صورت پارامتریک کار کنیم. مشتق گیری، انتگرال گیری، حد، سری، تبدیلات لاپلاس ، فوریه و Z و حل دستگاههای معادلات چند جمله ای و معادلات دیفرانسیل مباحثی هستند که در همین قسمت مورد بررسی قرار میگیرند. در این بخش به این موارد خواهیم پرداخت و رسم نمودار در این حوزه را به بخش رسم نمودار موکول می کنیم.

الف) ساخت متغیرهای Symbolic : ما می توانیم یک حرف یا یک کلمه را به عنوان یک متغیر غیر عددی و پارامتریک معرفی کنیم. به عنوان مثال متغیری که همیشه در ریاضیات با آن سر و کار داشته ایم، x را می خواهیم یک متغیر سیمبلیک در نظر بگیریم. دستور زیر این کار را برای ما انجام می دهد:

```
>> x=sym('x')
x=
x
```

گاهی متغیرهای پارامتری ما بیش از یکی است. چه برای یک متغیر و چه چند متغیر می تـوان از دسـتور syms اسـتفاده کرد. در اینجا تنها کافیست نام متغیرهای پارامتری خود را در مقابل دستور تایپ کنید:

```
>> syms x y z
```

با نوشتن این دستور متغیرهای x ، y و z به عنوان متغیرهای پارامتری در workspace قرار می گیرند. هر متغیری که به عنوان تابعی از یک متغیر پارامتری تعریف شود به طور اتوماتیک به عنوان یک متغیر سیمبلیک در نظر گرفته می شود. مثلاً در مثال زیر f چون تابعی از x است، یک متغیر پارامتریک در نظر گرفته شده است:

```
>> syms x
>> f=sin(x)
f=
sin(x)
>>whos f
  Name      Size            Bytes    Class
  f         1x1               136    sym object
Grand total is 7 elements using 136 bytes
```

شما می توانید یک متغیر را تابعی از چند متغیر پارامتری دیگر قرار دهید. دستور findsym نشان می دهد که هر متغیر به چه متغیرهای پارامتری دیگری وابسته است.البته غالباً حاصل اعمال این تابع متغیرهای پارامتری اولیه هستند.مثال:

```
>>syms x y z
>>f=sin(x)+cos(y);
>>g=f*z;
>>findsym(g)
ans=
x, y, z
```

شما می توانید با استفاده از دستور subs در یک عبارت پارامتری یک متغیر پارامتری را با یک عدد و یا یک پارامتر جدید و یا حتّی یک ماتریس جایگزین کنید. به عنوان مثال برای تابع f در مثال قبل داریم:

```
>> subs(f,'x',1)
ans=
sin(1)+cos(y)
>> subs(f,y,'theta')
ans=
sin(x)+cos(theta)
>> subs(f,{x,y},{'theta','alfa'})
ans=
sin(theta)+cos(alfa)
>> subs(f,{x,y},{pascal(3),magic(3)})
ans=
    0.6960    1.3818    1.8016
   -0.1485    1.1930    0.8950
    0.1878   -0.7700   -0.6956
>> subs(f,{x,y},{sym(pascal(4)),magic(4)})
ans=
  [ sin(1)+cos(16),   sin(1)+cos(2),   sin(1)+cos(3),   sin(1)+cos(13) ]
  [  sin(1)+cos(5),  sin(2)+cos(11),  sin(3)+cos(10),    sin(4)+cos(8) ]
  [  sin(1)+cos(9),   sin(3)+cos(7),   sin(6)+cos(6),  sin(10)+cos(12) ]
  [  sin(1)+cos(4),  sin(4)+cos(14), sin(10)+cos(15),   sin(20)+cos(1)]
```

همانطور که در دو دستور اخیر مشاهده می کنید، اگر تمامی متغیرهای یک عبارت سیمبلیک با عدد یا ماتریسهای عددی جایگزین شوند، عبارت سیمبلیک به یک عبارت عددی تبدیل می شود. اما اگر حتّی یک متغیر به صورت سیمبلیک باقی بماند، عبارت ما سیمبلیک خواهد بود. تفاوت این دو دستور آخر آنست که در اولی به جای x ماتریس پاسکال به عنوان یک ماتریس عددی قرار گرفته است و در دومی به عنوان یک ماتریس سیمبلیک و تفاوت این دو شکل دستور در نتایج اجرای آنها مشهود می باشد. ضمناً به چگونگی استفاده از علائم ' ، {} ، [] و () در دستورات دقّت فرمایید.

اگر پس از اینکه یک تابع پارامتری را تعریف نمودیم، متغیرهای آن به عدد و یا ماتریس عددی تغییر یابند و بخواهیم مقدار تابع را به ازای متغیرهای عددی داشته باشیم، کافی است نام تابع را به تنهایی به تابع subs دهیم و به نتیجه ی دلخواه برسیم. برای مثال قبل داریم:

```
>>x=1;
>>y=2;
>>subs(f)
ans=
    0.4253
```

تابع دیگری که می تواند بخشهای قابل تبدیل یک عبارت سیمبلیک را به عدد تبدیل کند،تابع vpa است. با اعمال این تابع، هر بخشی از عبارت سیمبلیک که ارزش عددی و محاسباتی دارد، با مقدار عددی جایگزین می شـود. آرگومـان دوم ورودی این تابع تعیین می کند که اعداد با چه دقّتی محاسبه و نمایش داده شوند. مثال:

```
>>a=subs(f,'x',2)
a=
sin(2)+cos(y)
>>vpa(a)
ans=
.90929742682568169539601986591174+cos(y)
>>vpa(a,5)
ans=
.90930+cos(y)
```

ب) مشتق: برای مشتق گرفتن از یک عبارت جبری و پارامتری از تابع diff استفاده می شود. نوع و تعداد ورودی های این تابع بسته به انتظاری که از اعمال آن روی عبارت پارامتریمان داریم، متفاوت است. به صورت پیش فرض، اگر پارامتری را که میخواهیم مشتق گیری بر اساس آن انجام گردد ذکر نکنیم، مشتق گیری برمبنای x و اگر تابع وابسته بـه پـارامتری غیـر از x باشد بر مبنای نزدیکترین متغیر از نظر الفبایی به x صورت می گیرد و در این حالت حرفی که از نظر الفبایی بعد از x است به حرفی که قبل از x و با همان فاصله قرار گرفته، ترجیح دارد. در سایر توابع این بخش نیز این قرارداد صادق است .

به مثال های زیر که در هیچکدام ذکر نشده است که مشتق گیری بر اساس کدام پارامتر صورت گیرد توجه کنید:

```
>> syms u v w x y z
>> f1=sin(x)+cos(y);
>> f2=sin(y)+cos(z);
>> f3=sin(w)+cos(y);
>> f4=sin(w)+cos(z);
>> f5=sin(u)+cos(v);
>> diff(f1)
ans=
cos(x)
>> diff(f2)
ans=
cos(y)
>> diff(f3)
ans=
-sin(y)
>> diff(f4)
ans=
cos(w)
>> diff(f5)
ans=
-sin(v)
```

بهترین راه برای آنکه از مبنای مشتق گیری مطمئن باشیم و روند مشتق گیری را کاملاً در اختیار داشته باشیم، آنستکه نام متغیر مورد نظرمان را پس از نام تابع ذکر کنیم. مثال:

```
>> syms w x y
>> f=sin(x)+cos(y)+tan(w);
>> diff(f,x)
ans=
cos(x)
>> diff(f,y)
ans=
-sin(y)
>> diff(f,w)
ans=
1+tan(w)^2
```

سومین آرگومانی که می تواند به عنوان ورودی به تابع diff داده شود، مرتبه ی مشتق گیری است. مثلاً اگر شما بخواهید از تابعی بر مبنای x مشتق مرتبه ۲ بگیرید، کافیست سومین ورودی تابع diff را 2 در نظر بگیرید. به مثالهای زیر توجه فرمایید:

```
>>syms x y z
>>g=sin(x*y)*z;
>>diff(g,x,2)
ans=
-sin(x*y)*y^2*z
>>diff(g,y,3)
ans=
-cos(x*y)*x^3*z
>>diff(diff(g,z),x,4)
ans=
sin(x*y)*y^4
```

ج) انتگرال: تابع int برای گرفتن انتگرال معین و نامعین به کار می رود. قرارداد عملکرد انتگرال در مورد ورودی ها، مشابه مشتق است، با این تفاوت که اینجا پس از ورودی دوم که نشان می دهد انتگرال بر مبنای کدام متغیر گرفته می شود، اگر انتگرال نامعین باشد، دیگر آرگومانی وارد نمی کنیم و اگر معین باشد اول حدّ پایین انتگرال و سپس حد بالای آن را وارد میکنیم. در اینجا نیز اگر متغیر مبنا نوشته نشود، x یا در صورت عدم وجود آن متغیری که از نظر الفبایی به x نزدیکتر است، به عنوان متغیر مبنای انتگرال در نظر گرفته می شود. برای گرفتن انتگرال های چندگانه لازم است از تابع int به صورت تودرتو استفاده گردد.

مثال های متعدد زیر، چگونگی عملکرد تابع int را نشان می دهد؛ به چگونگی وارد کردن آرگومان ها توجّه کنید:

```
>> syms x y z t
>> w=sin(x)+cos(y)+exp(z)+t^2;
>> int(w)
ans=
-cos(x)+cos(y)*x+exp(z)*x+t^2*x
>> int(w,y)
sin(x)*y+sin(y)+exp(z)*y+t^2*y
>> int(w,-pi,pi)
ans=
2*cos(y)*pi+2*exp(z)*pi+2*t^2*pi
>> int(w,t,1,10)
ans=
9*sin(x)+9*cos(y)+9*exp(z)+333
>> int(w,z,-inf,0)
ans=
signum(sin(x)+cos(y)+t^2)*Inf
>> vpa(int(int(int(w,x),t),z,0,pi/2),3)
ans=
-1.57*cos(x)*t+1.57*cos(y)*x*t+3.81*x*t+.524*t^3*x
```

*تابع signum همان تابع sign است که اینجا نام آن در قسمت پاسخ کامل آورده شده ، اما در دستورات حتماً باید از خود تابع sign استفاده کنیم.

د) سری: منظور از سری در اینجا بدست آوردن مجموع مقادیر یک تابع به ازای بازه ای از مقادیر مختلف برای یکی از متغیرهای آن است. در اینجا سری در واقع نوعی انتگرال برای مقادیر گسسته و صحیح است و با این تعریف دو نوع معین و نامعین خواهد داشت که برای هر دو نوع از تابع symsum استفاده می کنیم. در اینجا نیز مشابه با انتگرال عمل میکنیم، با این تفاوت که در سری معین غالباً حدّ پایین و بالای سری عدد صحیح و گام آن یک است. در سری های نامعین هم از قوائد محاسبه ی سری های نامعین استفاده می گردد. مثال های متعدد زیر عملکردهای گوناگون این تابع را به خوبی آشکار میسازد:

```
>> syms a b c x y z
>> f1=a*b;
>> f2=a*x^2+b*y-exp(-c*z);
>> symsum(f1)
ans=
1/2*a*b^2-1/2*a*b
>> symsum(f1,a)
ans=
1/2*b*a^2-1/2*a*b
```

```
>> symsum(f1,0,3)
ans=
6*a
>> symsum(f1,a,0,3)
ans=
6*b
>> symsum(f2)
ans=
1/3*a*x^3-1/2*a*x^2+1/6*a*x+b*y*x-1/exp(c*z)*x
>> symsum(f2,y)
ans=
a*x^2*y+1/2*b*y^2-1/2*b*y-1/exp(c*z)*y
>> symsum(f2,c,1,3)
ans=
3*a*x^2+3*b*y-exp(-z)-exp(-2*z)-exp(-3*z)
>> vpa(symsum(symsum(symsum(f2,x,0,5),z,0,1),c,1,3),5)
ans=
36.*b*y-21.318+330.*a
>> symsum(symsum(symsum(f2,x,0,5),z,0,1),c,1,3)
ans=
36*b*y-18+330*a-6*exp(-1)-6*exp(-2)-6*exp(-3)
```

هـ) حد: تابع limit برای بدست آوردن حدّ تابع مورد استفاده قرار می گیرد و قابلیت محاسبه ی حدّ تابع در یک نقطه و حدّ چپ و راست تابع در آن نقطه را دارد. اگر تنها نام تابع را به عنوان ورودی به تابع limit بدهید، به طور پیش فرض حدّ تابع را به ازای x→0 محاسبه می کند و اگر تابع ما به x وابسته باشد طبق قرارداد به سراغ نزدیکترین متغیر موجود از نظر الفبایی به x می رود. اگر خواستید به ازای میل دادن متغیر دیگری به سمت صفر حدّ تابع را حساب کنید، نام آن متغیر را به عنوان دومین آرگومان ورودی تابع وارد کنید و عددی را که می خواهید به سمت آن میل کند را پس از آن حتماً بنویسید(حتی اگر آن عدد صفر باشد). در صورتی که می خواهید x (یا متغیر نزدیک به آن، زمانی که x نباشد)را به سمت عددی میل دهید، میتوانید پارامتر دوم را همان عدد قرار دهید. حالت بعدی زمانی پیش می آید که شما مایلید حدّ چپ یا راست تابع را بدست آورید. در این حالت که حتماً باید نام متغیر و عددی که به سمت آن میل می کند ذکر شود (حتی به ازای x) اگر پس از عدد در جایگاه چهارم عبارت 'left' را بنویسید ، حدّ چپ و اگر 'right' بنویسد ، حدّ راست تابع در آن نقطه بدست می آید. مثال های زیر را در مورد دو تابع f و g به دقّت ملاحظه نمایید:

```
>> syms x y z
>> f=x/sin(x);
>> g=y+ceil(z);
```

```
>> limit(f)
ans=
1
>> limit(f,1)
ans=
1/sin(1)
>> limit(g)
ans=
ceil(z)
>> limit(g,1)
ans=
1+ceil(z)
>> limit(g,z,0)
ans=
NaN
>> limit(g,z,0,'right')
ans=
y+1
>> limit(g,z,0,'left')
ans=
y
```

همانطور که ملاحظه فرمودید این تابع در مورد توابعی که حدّ آنها دارای ابهام است، خود رفع ابهام می کند و در مورد توابعی که حدّ چپ و راستِ برابری در یک نقطه ندارند و یا به هر دلیلی حد نداشته باشند، عبارت NaN را به معنای عدم وجود پاسخ برمیگرداند.

و) تبدیلات(فوریه، لاپلاس و Z) : تبدیلاتی که در حوزه ی عبارات symbolic مطرح می گردد عبارتند از تبدیل فوریه و معکوس آن، تبدیل لاپلاس و معکوس آن، و تبدیل Z و معکوس آن. این تبدیلات با تعاریف زیر بیان می شوند.

تبدیل فوریه ی تابع f :

$$f = f(x) \Rightarrow F = F(w) \qquad F(w) = \int_{-\infty}^{\infty} f(x) e^{-iwx} dx$$

این تابع با این تعریف به صورت F=fourier(f) بدست می آید. اگر بخواهم F تابعی از پارامتری غیر از w باشد، پارامتر مورد نظر را به عنوان آرگومان دوم به تابع می دهیم. در این صورت در واقع در تعریف تبدیل فوریه، پارامتر جدید به جای w قرار می گیرد. حال اگر تابع f وابسته به متغیر دیگری نیز باشد و بخواهیم تبدیل فوریه نسبت به آن متغیر انجام گیرد، باید آن متغیر آرگومان دوم ورودی و w یا هر متغیر دیگری که مایلیم در کنار این متغیر در توان عدد نپر(با توجه به تعریف تبدیل فوریه) باشد به عنوان متغیر سوم وارد شود. مثلاً F=fourier(f,y,w) موجب انتگرال گیری بر اساس y خواهد شد و در

توان e در کنار y ، w قرار می گیرد. در حالی که اگر در این مورد بنویسیم F=fouier(f,y)، انتگرال گیری بر اساس x انجام میشود و y جایگزین w می گردد. جهت فهم دقیق موضوع مثال های زیر را دقیق بررسی کنید:

```
>> syms x y u
>> f=sin(x);
>> g=sin(x)+sin(y);
>> fourier(f)
ans=
 i*pi*(dirac(w+1)-dirac(w-1))
>> fourier(f,u)
ans=
i*pi*(dirac(u+1)-dirac(u-1))
>> fourier(g)
ans=
pi*(i*dirac(w+1)+2*sin(y)*dirac(w)-i*dirac(w-1))
>> fourier(g,y)
ans=
pi*i*(dirac(y+1)-dirac(y-1))
>> fourier(g,y,u)
ans=
pi*(-i*dirac(u-1)+i*dirac(u+1)+2*sin(x)*dirac(u))
```

در محاسبه fourier(g,y)، در واقع y جایگزین w شده و انتگرال بر حسب x گرفته شده و چون 2*sin(y)*dirac(y) برابر صفر است ، مقدار آن در نتیجه ی اجرای دستور نیامده است. تابع dirac همان تابع ضربه است که به شکل زیرتعریف میشود:

```
>>int(heaviside(x),x,-inf,inf)
ans=
dirac(x)
```

تابع heaviside(x) همان تابع پله است که به ازای x>0 مقدار آن یک، به ازای x<0 مقدار آن صفر و به ازای x=0 تعریف نشده است.

تابع ifourier برای محاسبه ی فوریه معکوس یک تابع به کار می رود. فوریه ی معکوس تابع F طبق تعریف زیر بدست می آید.

$$F = F(w) \Rightarrow f = f(x) \qquad f(x) = 1/(2\pi)\int_{-\infty}^{\infty} F(w)e^{iwx}dw$$

همان شرایطی که برای تابع fourier وجود داشت، برای معکوس آن نیز صادق است، با این تفاوت که اینجا همانطور که در تعریف نیز مشهود است، تمام چیزهایی که در مورد x و w گفتیم ، با هم جا به جا شده و برعکس تبدیل فوریه است.

به مثال های زیر توجه کنید:

```
>> syms x t u w
>> f=1/w;
>> g=u+w;
>> ifourier(f)
ans=
1/2*i*(2*heaviside(x)-1)
>> ifourier(f,t)
ans=
1/2*i*(2*heaviside(t)-1)
>> ifourier(g)
ans=
 u*dirac(x)-i*dirac(1,x)
>> ifourier(g,u,x)
ans=
 -i*dirac(1,x)+w*dirac(x)
```

تبدیل لاپلاس : تبدیل دیگری که غالباً در بسیاری مباحث به آن نیاز داریم، تبدیل لاپلاس است. کلیه ی اتّفاقاتی که در تبدیل فوریه، در مورد x و w رخ می داد، اینجا برای t و s رخ می دهد، البته در چهارچوب تعریف زیر:

$$F = F(t) \Rightarrow L = L(s) \qquad L(s) = \int_0^{\infty} F(t)e^{-st}dt$$

اینجا L=laplace(F) این تعریف را در نظر می گیرد. بدیهی است که این تعریف زمانی صدق می کند که F تابعی از t باشد. اگر F تابع x باشد چطور؟ در اینجا اگر وابستگی تابع هم به x باشد و هم t ، ارجحیت با t خواهد بود، اما اگر x تنها یا با متغیری غیر از t باشد، انتگرال گیری بر مبنای x انجام میگیرد. کمی دقت در مثال های زیر، این مسأله را روشن می کند.

```
>> syms x y t s r
>> f=heaviside(x);
>> g=x+sin(t);
>> h=x+sin(y);
>> laplace(f)
ans=
1/s
>> laplace(g)
ans=
x/s+1/(s^2+1)
>> laplace(h)
ans=
1/s^2+sin(y)/s
>> laplace(f,t)
ans=
1/t
>> laplace(g,x,r)
ans=
  1/r^2+sin(t)/r
```

معکوس لاپلاس که از تعریف زیر پیروی می کند نیز با استفاده از تابع ilaplace قابل محاسبه است:

$$L = L(s) \Rightarrow F = F(t) \qquad F(t) = \int_{c-i\infty}^{c+i\infty} L(s)e^{st}ds$$

مثال های زیر را بررسی کنید تا به چگونگی عملکرد تابع به ازای ورودی های مختلف بهتر پی ببرید:

```
>> syms s t a x y
>> f=1/s^2;
>> g = 1/(t-a)^2;
>> ilaplace(f)
ans=
t
>> ilaplace(g)
ans=
x*exp(a*x)
>> ilaplace(g,y)
ans=
y*exp(a*y)
>> ilaplace(g,y,x)
ans=
 1/(t-a)^2*dirac(x)
>> ilaplace(g,x,a)
ans=
1/(t-a)^2*dirac(a)
>> ilaplace(g,a,x)
ans=
x*exp(t*x)
```

تبدیل Z : از دیگر تبدیلات ریاضی است که همتای زمان گسستة تبدیل لاپلاس می باشد و در قالب تعریف زیر شناخته میشود:

$$f = f(n) \Rightarrow F = F(z) \qquad F(z) = \sum_{0}^{\infty} \frac{f(n)}{z^n}$$

این تعریف زمانی صادق است که f تابعی از n باشد و ما بخواهیم تبدیل Z آن تابعی از z باشد که با دستور ztrans(f) این تبدیل با این متغیرهای پیش فرض انجام می گیرد.

اگر f خود تابعی از z باشد و دستور را به صورت ztrans(f) بنویسیم، z در تعریف اولیه ی تبدیل، جایگزین n شده و خروجی به جای z، به صورت پیش فرض تابعی از w خواهد بود. رابطه ی زیر این مطلب را بهتر بیان می کند:

$$f = f(z) \Rightarrow F = F(w) \qquad F(w) = \sum_{0}^{\infty} \frac{f(z)}{w^{z}}$$

واضح است که اگر f جز n یا z تابع متغیر دیگری نیز باشد و ما بخواهیم بر اساس آن متغیر تبـدیل Z بگیریم،بـا اجـرای دستور به شکل ztrans(f) ، به نتیجه ای متفاوت با انتظارمان از اجرای تبدیل Z خواهیم رسید. در این حالت اگر به فرض f تابعی از k باشد و ما بخواهیم k جایگزین n در تعریف شود، دستور ztrans(f,k,z) ما را به مطلوبمان خواهد رساند و اگـر مثلاً بخواهیم همین تبدیل تابعی از w را منتج گردد، دستورztrans(f,k,w) مطلوب خواهد بود. روابط زیر بیان ریاضی این دو شکل از تبدیل Z است:

ztrans(f,k,z)→

$$f = f(k) \Rightarrow F = F(z) \qquad F(z) = \sum_{0}^{\infty} \frac{f(k)}{z^{k}}$$

ztrans(f,k,w)→

$$f = f(k) \Rightarrow F = F(w) \qquad F(w) = \sum_{0}^{\infty} \frac{f(k)}{w^{k}}$$

اگر f تابعی از n باشد یا اینکه به دلیلی بخواهید تبدیل بر اساس n انجام شود ، اما نتیجه تابع پارامتری غیر از z باشد، دیگر نیازی نیست سه ورودی داشته باشید. بلکه ورودی دوم در تعریف اولیه خود جایگزین z خواهد شد. مثلاً اگر بخواهیم حاصل تبدیل تابع w باشد اجرای دستورztrans(f,w) این کار را انجام خواهد داد. این دستور به زبان ریاضی عبارتست از:

$$f = f(n) \Rightarrow F = F(w) \qquad F(w) = \sum_{0}^{\infty} \frac{f(n)}{w^{n}}$$

مثال های زیر این روابط را روشن تر خواهد ساخت:

```
>> syms n k w z u
>> f=n^2;
>> ztrans(f)
ans=
z*(z+1)/(z-1)^3
>> ztrans(f,u)
ans=
u*(u+1)/(u-1)^3
>> f=z^2;
>> ztrans(f)
ans=
w*(w+1)/(w-1)^3
>> f=(k+n)^2;
```

```
>> ztrans(f)
ans=
k^2*z/(z-1)+2*k*z/(z-1)^2+z*(z+1)/(z-1)^3
>> ztrans(f,k,z)
ans=
z*(z+1)/(z-1)^3+2*n*z/(z-1)^2+n^2*z/(z-1)
>> ztrans(f,k,w)
ans=
z*(w+1)/(w-1)^3+2*n*w/(w-1)^2+n^2*w/(w-1)
```

تابع iztrans نیز وجود دارد که با تعریف زیر معکوس تبدیل Z را بدست می آورد.

$$F = F(z) \Rightarrow f = f(n) \qquad f(n) = \frac{1}{2\pi i} \oint_{|z| = R} F(z) z^{n-1} dz,\ n = 1, 2, \ldots$$

همانطور که از این تعریف بر می آید، F تابعی از z و حاصل اعمال تبدیل معکوس Z روی آن تابع f می باشد که تابعی از n است. همانند موارد پیشین این متغیرها پیش فرض تعریف این تبدیل در MATLAB هستند که با همان شرایط مشابه موارد قبلی، قابل جایگزنی با متغیرهای دیگر میباشند. در اینجا برای بیان موضوع تنها به چند مثال بسنده می کنم :

```
>> syms z w n k
>> f = 2*z/(z-2)^2;
>> iztrans(f)
ans=
2^n*n
>> g = n*(n+1)/(n^2+2*n+1);
>> iztrans(g)
ans=
(-1)^k
>> f=z/(z-w);
>> iztrans(f)
ans=
w^n
>> iztrans(f,k)
ans=
w^k
>> f=w/(w-z);
>> iztrans(f,w,n)
ans=
z^n
>> iztrans(f,w,k)
ans=
z^k
```

ز) توابع ترکیبی : توابع ترکیبی در MATLAB با استفاده از تابع compose ساخته می شوند. چگونگی قرار گرفتن اسامی توابع و متغیرها در بخش ورودی تابع، تعیین کننده ی نحوه ی عملکرد تابع compose است. در مثال های زیر پس از تعریف چند تابع ، به ساختن توابع ترکیبی از آنها پرداخته ایم:

```
>> syms x y z t u
>> f=1/(1+x^2);          %f=f(x)
>> g=sin(y);             %g=g(y)
>> h=x^t;                %h=h(x,t)
>> p=exp(-y/u);          %p=p(y,u)
>> compose(f,g)          %f(x=g(y))
ans=
1/(1+sin(y)^2)
>> compose(f,g,t)        %f(x=g(y=t))
ans=
1/(1+sin(t)^2)
>> compose(h,g,x,z)      %h(x=g(y=z),t)
ans=
sin(z)^t
>> compose(h,g,t,z)      %h(x,t=g(y=z))
ans=
x^sin(z)
>> compose(g,f,y,x,z)    %g(y=f(x=z))
ans=
sin(1/(1+z^2))
>> compose(h,p,x,u,t)    %h(x=p(y,u=t),t)
ans=
exp(-y/t)^t
```

ح) معکوس تابع : با تعریف معکوس یک تابع در دوران دبیرستان آشنا شده اید. اگر g معکوس تابع f(x) باشد آنگاه g(f(x))=x . در MATLAB تابع finverse معکوس تابعی را که به عنوان ورودی به آن می دهیم برمی گرداند. اگر تابع ما به بیش از یک متغیر وابسته باشد،لازم است متغیری که مایلیم معکوس تابع به آن وابسته باشد را به عنوان آرگومان دوم به تابع finverse دهید. البته مانند بسیاری از توابع این بخش، برای این تابع نیز x متغیر پیش فرض است . چند مثال:

```
>> syms x v t
>> finverse(exp(x))
ans=
log(x)
>> finverse(sqrt(t))
ans=
t^2
```

```
>> finverse(cos(v)+sin(x),v)
ans=
acos(-sin(x)+v)
>> finverse(cos(v)+sin(x))
ans=
-asin(cos(v)-x)
>> finverse(v^x)
ans=
log(x)/log(v)
>> finverse(v^x,v)
ans=
exp(log(v)/x)
```

ط) حل معادلات یک مجهولی و دستگاه معادلات چند مجهولی : یکی از امکانات بسیار مفید نرم افزار MATLAB حل معادلات پیچیده است. شما می توانید به راحتی معادلات خود را به تابع solve دهید تا معادلات شما را حل نموده و تمامی جوابهای ممکن آن را برای شما پیدا کند. اگر طرف راست معادله ای صفر است، کافیست شما به جای نوشتن معادله ی کامل، به نوشتن عبارت طرف چپ اکتفا کنید، اما در مورد سایر موارد نوشتن هر دو سمت معادله الزامیست. برای یافتن جواب معادلات، ابتدا هر یک از آنها را بین دو ' قرار داده و بین هر مجموعه , بگذارید. پس از نوشتن معادلات، متغیرهایی که مایلید به عنوان جواب معادله و یا دستگاه معادلات مقادیر آنها یافته شود را به همان صورت که معادلات را وارد کردید، به تابع می دهید. اگر نام متغیرها را پس از نام معادلات وارد ننمایید، مانند بسیاری توابع دیگر در حوزه ی عبارات sybolic، متغیر x و یا در صورت نبودن آن و یا وجود چند متغیر ، نزدیکترین متغیر به x در نظر گرفته می شود. اگر تعداد معادلات با تعداد متغیرها برابر باشد، تمامی پاسخ ها در صورت وجود، عددی خواهند بود. در غیر این صورت با رعایت اولویت فوق الذکر و یا بنا به متغیرهایی که شما پس از معادلات، نام آنها را ذکر نموده اید، پاسخ ها پارامتریک و وابسته به سایر متغیرهاست. مثلاً اگر معادلات دارای سه متغیر x ، y و z باشند و شما تنها دو معادله به تابع دهید، بدون ذکر نام متغیرها پس از معادلات، معادلات بر حسب x و y حل شده، و پاسخها تابع z خواهند بود.

اگر برای حل یک دستگاه معادلات چند متغیره ، تابع solve مساوی با یک متغیر قرار گیرد (مثلاً w=solve('f(x,y)','g(x,y)') آنگاه آن متغیر به عنوان یک structure حاوی پاسخ های معادلات در نظر گرفته میشود که در این صورت نمی توان مستقیماً مقادیر بدست آمده را پس از اجرای دستور مشاهده کرد. برای مشاهده ی مقادیر خروجی برای هر متغیر، باید نام متغیر حاوی پاسخ ها(مثلا w) ابتدا و پس از آن نام متغیر مورد نظر(مثلا x یا y) با واسطه ی یک نقطه(.) ذکر شود تا مقادیر بدست آمده برای آن متغیر قابل نمایش یا استفاده گردد(مثلاً در این مثال برای چاپ مقادیر بدست آمده برای x و y باید به ترتیب عبارات w.x و w.y تایپ شود).

بهتر است به جای مساوی قرار دادن تابع solve با یک متغیر، آنرا با یک آرایه از نامهای دلخواه برای پاسخهای خروجی تابع، مساوی قرار دهیم.(مثلا [x1,y1]=solve('f(x,y)','g(x,y)')) بدین ترتیب علاوه بر اینکه به ازای پاسخهای هر متغیر، یک داده با نام دلخواه به عنوان پاسخی جداگانه داریم، مقادیر آنها نیز نمایش داده می شود. اگر برای دستگاه جوابی پیدا نشود،ضمن اعلام یک پیام warning ، خروجی تابع به ازای هر متغیر،یک ماتریس تهی [ ] است.مثالهای زیر میتوانند منظور جملات اخیر را بهتر روشن سازند:

```
>> syms a b c x y z
>> solve('a*x^2+b*x+c')
ans=
1/2/a*(-b+(b^2-4*a*c)^(1/2))
1/2/a*(-b-(b^2-4*a*c)^(1/2))
>> solve('x^2+y^2=4','x+y=1')
ans=
    x:   [2x1  sym]
    y:   [2x1  sym]
>> w=solve('x^2+y^2=4','x+y=1')
w  =
    x:   [2x1  sym]
    y:   [2x1  sym]
>> w.x
ans=
  1/2-1/2*7^(1/2)
  1/2+1/2*7^(1/2)
>> w.y
ans=
  1/2+1/2*7^(1/2)
  1/2-1/2*7^(1/2)
>> [X,Y]= solve('x^2+y^2=4','x+y=1')
X=
   1/2-1/2*7^(1/2)
   1/2+1/2*7^(1/2)
Y=
   1/2+1/2*7^(1/2)
   1/2-1/2*7^(1/2)
>> [A,B,C]=solve('a*x+y','a+b+c=5','a*c*y^2=16','a','b','c')
A=
-y/x
B=
 (y^4+16*x^2+5*x*y^3)/x/y^3
C=
-16*x/y^3
>> [X,Y]=solve('x^y=0','y^x=-1')
 Warning: Explicit solution could not be found.
>In solve at 140
X=
[ empty sym ]
Y=
      [ ]
```

ی) حل معادلات دیفرانسیل : از تواناییهای ویژه ی نرم افزار MATLAB ، حل معادلات دیفرانسیل با استفاده از تابع dsolve است. معادلات دیفرانسیل همانند معادلات معمولی در ارتباط با تابع solve ، به تابع dsolve داده می شوند . عملگر D در این معادلات به صورت پیش فرض به عنوان جایگزینی برای d/dt استفاده می شود و در صورتی که t متغیر مستقل نباشد، به شرطی که نام متغیر مستقل در محل خود ذکر شود، مخرج کسر تغییر می کند. مثلا اگر تابع مقابل D ، f باشد و متغیر مستقل موجود در معادله که مشخص نموده ایم ،x باشد، آنگاه معنای Df ، df/dx خواهد بود. در این حال D2f به معنای d^2f/dx^2 می باشد. پس از نوشتن معادلات دیفرانسیل موجود، می توانید شرایط مرزی را نیز وارد کنید. در صورتی که شرایط مرزی را نداشته باشید و یا وارد نکنید، در پاسخ حل معادله ی دیفرانسیل مقادیر پارامتری ویژه ای که باC1،C2،C3و...نشان داده می شوند، ظاهر میشود. شرایط مرزی عبارتست از مقدار تابع و یا دیفرانسیل تابع به ازای مقادیر مشخصی از متغیر مستقل. مثلاً اگر مقدار تابع f در x=0 ، برابر 1 باشد، این شرایط مرزی را به صورت معادله ی 'f(0)=1' ، پس از معادلات دیفرانسیل، به تابع dsolve میدهیم. پس از وارد کردن شرایط مرزی ، آخرین ورودی تابع dsolve، نام متغیر مستقل خواهد بود که همواره یک متغیر بیشتر نمیتواند باشد.

در مورد شرایط مرزی ذکر این نکته می تواند مفید باشد که در صورت عدم وجود این شرایط، اجرای دستور میتواند جوابهای بیشتری داشته باشد و در واقع شرایط مرزی پاسخ حل معادلات دیفرانسیل را محدود و غالباً منحصر به فرد می کنند.

قبل از ارائه ی چند مثال لازم می دانم روی این نکته تأکید کنم که عدم ذکر نام متغیر مستقل مورد نظرتان در آخرین جایگاه آرگومانهای ورودی تابعdsolve از نظر نرم افزار بدان معناست که شما متغیرt را به عنوان متغیر مستقل تمامی معادلات دیفرانسیل موجود در ورودی تابع، در نظر گرفته اید و بدیهیست که عدم توجه به این نکته موجب حل نادرست معادلات دیفرانسیلی شما میگردد.

اکنون توجه شما را به چند مثال جلب می کنم:

```
>> syms t x y a b c
>> dsolve('Dy=-a*x')    %dy/dt=-a*x
ans=
-a*x*t+C1
>> dsolve('Dy=-a*x','x')    %dy/dx=-a*x
ans=
-1/2*a*x^2+C1
>> dsolve('Dy=-a*x','y(1)=5','x')
ans=
-1/2*a*x^2+1/2*a+5
>> dsolve('Df=f+sin(t)')
ans=
-1/2*cos(t)-1/2*sin(t)+exp(t)*C1
>> dsolve('D2y=-a^2*y','y(0)=1','Dy(pi/a)=0')
ans=
cos(a*t)
>> [X,Y]=dsolve('Dx = y', 'Dy = -x')
X=
C1*sin(t)+C2*cos(t)
Y=
C1*cos(t)-C2*sin(t)
```

```
>> y = dsolve('(Dy)^2 + y^2 = 1','y(0) = 0')
y=
  sin(t)
 -sin(t)
```

در مثال آخر دقت داشته باشید که (Dy)^2 کاملا با D2y متفاوت است و اگر شما در همین دستور به جای (Dy)^2 بنویسید D2y نه تنها پاسخی برای معادله ی شما نمی یابد، بلکه شما با یک پیام Error مواجه می شوید.

در پایان ذکر این نکته نیز لازم به نظر می رسد که گاهی معادله ی دیفرانسیل شما جواب قطعی و واضح(Explicit) ندارد و در صورت امکان نرم افزار برای شما یک پاسخ غیر مستقیم و مفهومی(Implicit) می دهد. این چنین پاسخی در واقع یک راه حل پیشنهادی برای شماست که می تواند شما را در حل درست و مستقیم این معادله ی دیفرانسیل که به شکل فعلی ، حل مستقیمی برای آن یافته نشده است، راهنمایی کند. پاسخ غیر مستقیم که معمولاً با یک پیامwarning همراه است، زمانی به شما داده می شود که شرایط مرزی برای مسئله قائل نشده باشید.

ک) ساده سازی و ویرایش: گاهی برخی عبارات symbolic بسیار پیچیده و یا طولانی هستند و شما مایل هستید که آنها را به عبارات ساده تر یا کوتاهتری تبدیل کنید تا بتوانید نتیجه ی بهتری از آن بگیرید. به این منظور چند تابع در MATLAB منظور شده است که شما را به ویرایش دلخواهتان از یک عبارت جبری و symbolic رهنمون می گردد. اولین و کاملترین تابع در این رابطه، تابع simple است که تمامی راههای موجود برای ویرایش یک عبارت به اشکال گوناگون را بررسی و چاپ می کند. به مثال زیر توجه کنید:

```
>> syms x y
>> simple(x^3+3*y^2*x^2+4*x*y+sin(x+y));
simplify:
x^3+3*y^2*x^2+4*x*y+sin(x+y)
radsimp:
x^3+3*y^2*x^2+4*x*y+sin(x+y)
combine(trig):
x^3+3*y^2*x^2+4*x*y+sin(x+y)
factor:
x^3+3*y^2*x^2+4*x*y+sin(x+y)
expand:
x^3+3*y^2*x^2+4*x*y+sin(x)*cos(y)+cos(x)*sin(y)
combine:
x^3+3*y^2*x^2+4*x*y+sin(x+y)
convert(exp):
x^3+3*y^2*x^2+4*x*y-1/2*i*(exp(i*(x+y))-1/exp(i*(x+y))
convert(sincos):
x^3+3*y^2*x^2+4*x*y+sin(x+y)
convert(tan):
x^3+3*y^2*x^2+4*x*y+2*tan(1/2*x+1/2*y)/(1+tan(1/2*x+1/2*y)^2)
collect(x):
x^3+3*y^2*x^2+4*x*y+sin(x+y)
mwcos2sin:
x^3+3*y^2*x^2+4*x*y+sin(x+y)
```

همانگونه که ملاحظه فرمودید، استفاده از تابع simple به این شکل، تمامی حالات ممکن برای ساده سازی یک عبارت را در مورد عبارت symbolic ورودی خود امتحان نموده و نتایج را چاپ می کند؛ اگرچه ممکن است بسیاری از این حالات برای این عبارت ناکارآمد باشد و هیچ اثری در ویرایش آن نگذارد.

از دستور simple به گونه ای دیگر نیز می توان استفاده نمود. در این حالت دستور مساوی با یک آرایه ی سطری دو متغیره قرار می گیرد. اجرای دستور موجب می شود که عبارت به یکی از روشها که خود نرم افزار مفید تشخیص می هد به کوتاهترین عبارت ممکنه ساده شده و عبارت ساده شده در متغیر اول ماتریس قرار گیرد. متغیر دوم حاوی نام روشی خواهد بود که مورد استفاده قرار گرفته است. البته در این حالت ممکن است نرم افزار راهی برای کوتاهتر نمودن عبارت پیدا نکند و در نتیجه عین عبارت را در متغیر اول و در متغیر دوم مقدار تهی را قرار دهد. چند مثال:

```
>> syms x y
>> [s,h]=simple(sin(x+y)+cos(x+y))
s=
   sin(x+y)+cos(x+y)
h=
      [ ]
>> [s,h]=simple(sin(x+y)*cos(x+y))
s=
   1/2*sin(2*x+2*y)
h=
    combine(trig)
>> [r,how]= simple(x^2+2*x+1)
r=
  (x+1)^2
how=
factor
>> [A,B]= simple(cos(x)+i*sin(x))
A=
exp(i*x)
B=
convert(exp)
```

بعضی از حالات پر کاربردی که دستور simple چاپ می کند، به عنوان توابعی مستقل در MATLAB شناخته می شوند. این موارد عبارتند از simplify ، collect ، horner ، factor و expand. simplify شکل خاصی از simple است که با استفاده از قوانین ساده سازی ریاضی عبارت موجود را ساده میسازد. به عنوان مثال :

```
>> syms x y
>> simplify(sin(x)^2 + cos(x)^2)
ans=
1
>> simplify( [(x^2+5*x+6)/(x+2),sqrt(16)])
ans=
[  x+3 ,   4 ]
```

دستور collect به صورت پیش فرض عبارت جبری داده شده به تابع را به ترتیب از بزرگترین تا کوچکترین توان x مرتب می کند. اگر x در عبارت نباشد و یا پس از دادن عبارت جبری به عنوان ورودی اول تابع، نام متغیر دیگری وارد شود، عبارت جبری بر اساس ترتیب نزولی توان آن متغیر مرتب می شود. مثال:

```
>> syms x y
>> collect((exp(x)+x)*(x+2))
ans=
x^2+(exp(x)+2)*x+2*exp(x)
>> collect((x+y)*(x^2+y^2+1), y)
ans=
y^3+x*y^2+(x^2+1)*y+x*(x^2+1)
>> collect( (x+1)*(y+1))
ans=
 (y+1)*x+y+1
```

دستور horner تمامی عبارات جبری را آنقدر ساده می کند تا هیچ عبارت توانی ای باقی نماند و تمامی عبارات به ضرایبی از متغیرها یا ترکیبات بدون توان آنها تبدیل شود؛ مگر آنکه یک متغیر توانی ضریب پارامتری نداشته باشد و به این دلیل به ضرایب کوچکتری تبدیل نشود. در اینجا نیز اولویت با x و متغیرهای نزدیک به آن است. مثال:

```
>> syms x y t
>> horner(x^3+3*y*x^2+x*y^2+x*y)
ans=
 ((y+1)*y+(3*y+x)*x)*x
>> horner(x^2+y^3+x*y)
ans=
y^3+(x+y)*x
>> horner(t^3+2*t^2*y^2+5*t*y+y^3)
ans=
t^3+(5*t+(2*t^2+y)*y)*y
```

تابع factor در حالت symbolic به عنوان تابعی برای یافتن عباراتی که می تواند به عنوان ضریبی برای سایر عبارات در یک عبارت جبری قرار گیرد و تا حدّ ممکن آنرا به ضرایبی از ترکیبات جبری تبدیل کند، مورد استفاده قرار می گیرد. مثلاً:

```
>> syms x y z
>> factor(x^3-y^3)
ans=
 (x-y)*(x^2+y^2+x*y)
>> factor(x^2+y^2-2*x*y)
ans=
 (x-y)^2
>> factor(3*t^3*y^3+9*t^2*y^4+18*y^3*t^5)
ans=
 3*y^3*t^2*(6*t^3+t+3*y)
>> factor(x*t+4*x*y+4*t*y+16*y^2)
ans=
 (x+4*y)*(t+4*y)
```

تابع آخر expand است که عبارت مقابل خود را به گسترده ترین شکل ممکن در می آورد و برای این کار از ضرب کردن عبارات در هم، تبدیل توابع سینوسی که ورودی آنها مجموع متغیرهاست به شکل ضرب عبارات مثلثاتی و یا تبدیل عبـاراتی که توان آنها مجموع چند متغیر است به ضرایب نمایی و ... استفاده می نماید. مثال:

```
>> syms a b c
>> expand(sin(a+b-c))
ans=
 sin(a)*cos(b)*cos(c)+sin(a)*sin(b)*sin(c)+cos(a)*sin(b)*cos(c)-cos(a)*cos(b)*sin(c)
>> expand(exp((a+b)^2))
ans=
exp(a^2)*exp(a*b)^2*exp(b^2)
>> expand((a+b-3*c)*(2*a-b+4*c))
ans=
 2*a^2+a*b-2*a*c-b^2+7*b*c-12*c^2
```