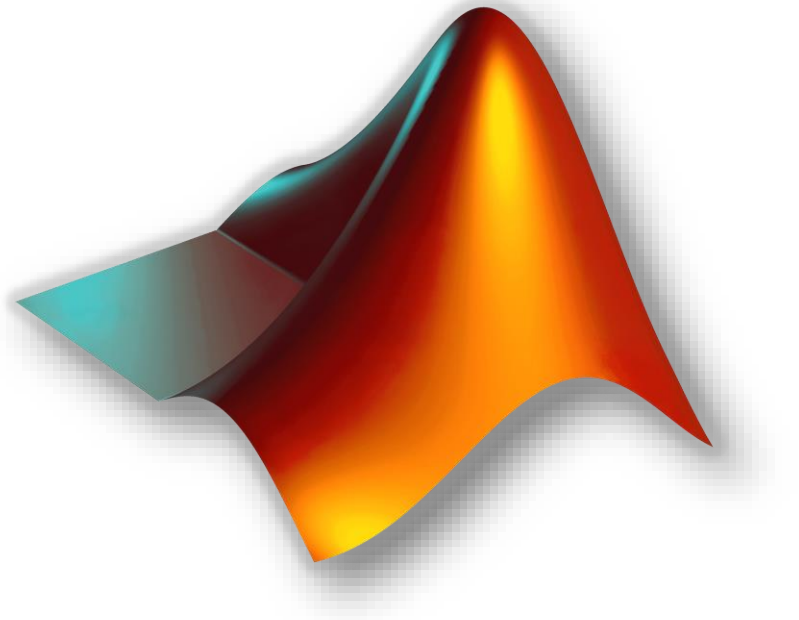


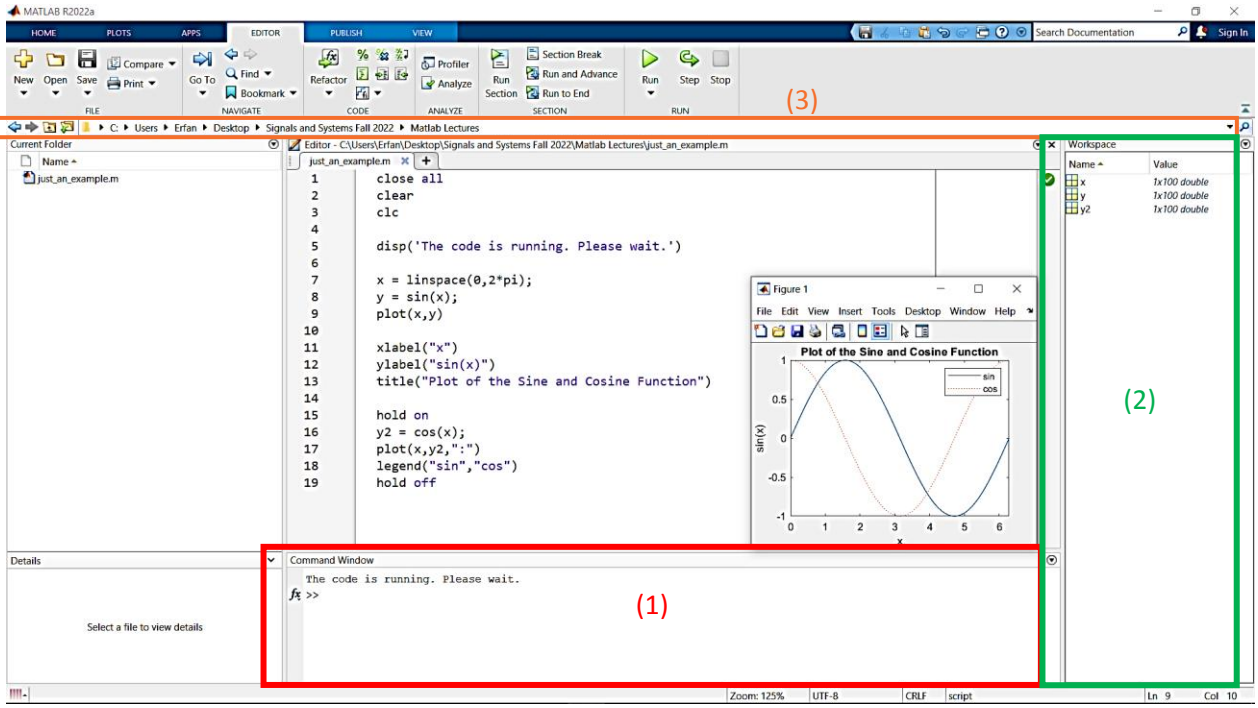
MATLAB

ارائه دهنده و استاد درس: دکتر هادی عزمی

با همکاری: علی اکبر سامانی، پارسا وطن پور، سارینا زیرک سیما، عرفان یعقوبی و عرفان احمدی

آشنایی با محیط نرم افزار MATLAB

هنگامی که نرم افزار **MATLAB** را باز می‌کنید با چنین پنجره‌ای روبه‌رو خواهید شد.



۱. Command Window:

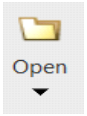
پنجره دستور یا فرمان محیطی هست که در آن میتوان دستورات مربوطه را وارد کرد و آن ها را اجرا کرد. از این پنجره بیشتر بعنوان محیطی برای اجرای کد استفاده می شود و کد های متلب را در file.m ها که مزایای بیشتری نسبت به Command window دارند نوشته می شوند.

۲. Workspace:

در این پنجره متغیرهای که تا کنون در Command window تعریف شده اند به همراه مقادیر، نوع، سایز و... نمایش داده می شوند.

کار با m فایل ها

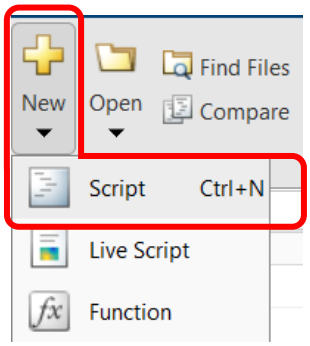
کد های متلب در **Editor** نوشته می شوند. پسوند فایل های حاوی کد **MATLAB** از نوع **.m** بوده و این فایل ها، به **m** فایل معروف هستند.

_ از طریق گزینه 

در ســـربرگ **home** می‌توان از دایرکتوری مورد نظر، **m** فایل کد خود را انتخاب نموده و آن در نرم افزار **MATLAB** باز کرد.

۳. **Direction line** :

نوار محل باز شدن نرم افزار را نشان می‌دهد. (که بسته به این که فایل مورد نظر شما در کدام مسیر قرار گرفته است متفاوت خواهد بود.)

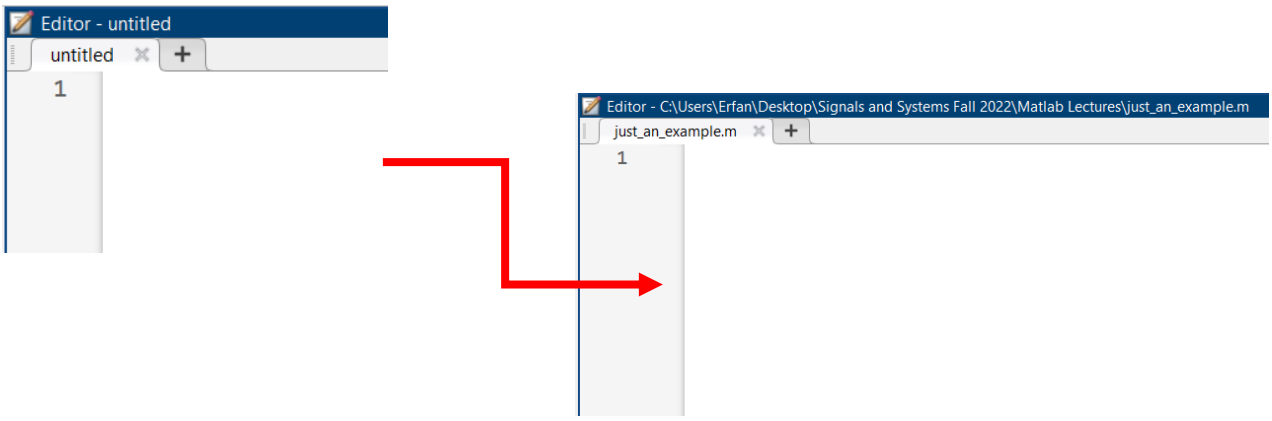


- برای ایجاد یک **m** فایل جدید کافیست از سربرگ home گزینه‌ی 

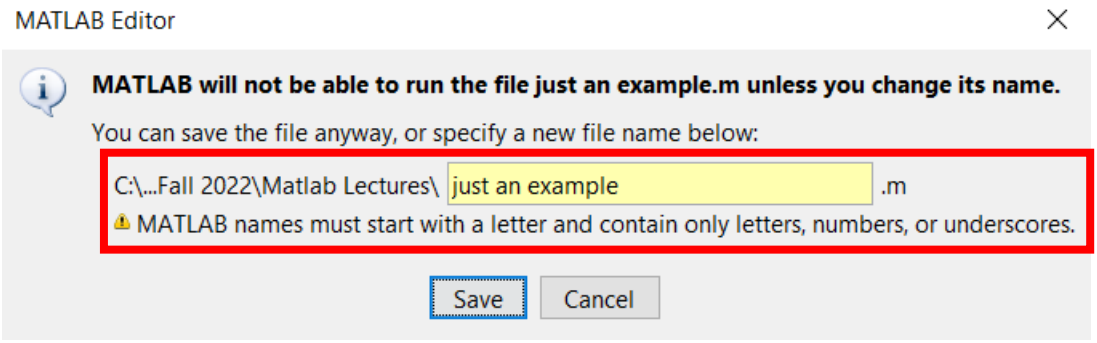
را انتخاب و سپس از منوی باز شده گزینه‌ی **Script** انتخاب نمایید. همچنین برای راحتی و سرعت بیشتر میتوانید از میانبر **Ctrl + N** استفاده نمایید.

پس از ایجاد **script** جدید، پنجره‌ی ادیتور نرم افزار باز شده و می‌توانید کد های مورد نظر خود را در آن بنویسید. اما قبل از آن نیاز دارید که فایل خود را از طریق گزینه 

به فرمت **'.m'** ذخیره نمایید.



نکته: توجه کنید که برای تخصیص نام فایل مورد نظر، حتما باید از یک اسم ۶۳ کاراکتری که با حروف انگلیسی شروع شده و شامل اعداد و حروف انگلیسی و کاراکتر **underscore** (_) باشد استفاده کنید.



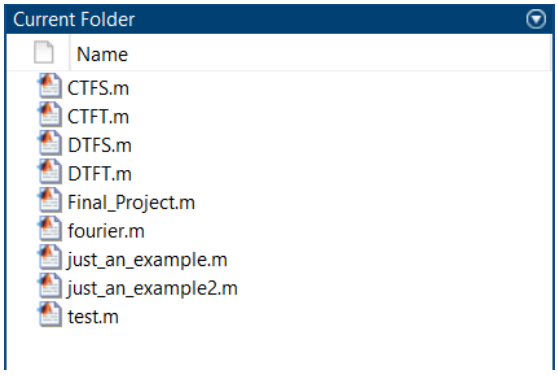
Example.m ✓ **e.x.ample.m** ✗

test_.m ✓ **فایل_چهارم.m** ✗

final_48.m ✓ **04test.m** ✗

f_i_n_a_l.m ✓ **_final.m** ✗

- از قسمت Current Folder میتوانید فایل هایی را که در دایرکتوری مورد نظر خود قرار گرفته انتخاب و باز کنید.

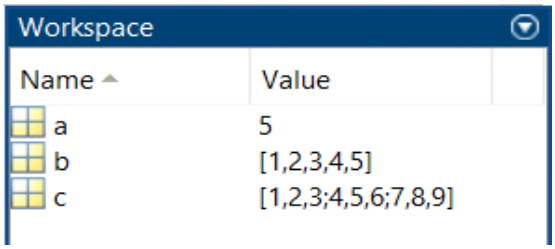


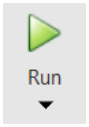
نکته: به صورت پیش فرض بعضی ازپنجره های متلب نمایش داده نمی شوند ، یا برخی از این پنجره ها نیاز به حذف شدن دارند. برای تنظیمات پنجره های متلب از سربرگ **Home** دکمه باز شونده ای **ENVIRONMENT** را انتخاب کنید و وارد **Layout** شوید، از این قسمت می توانید براحتی پنجره های مورد نیاز متلب را اضافه یا کم کنید.

- در پنجره **Workspace** متغیرهای که تا کنون در **Command window** تعریف شده اند به همراه مقادیر، نوع ، سایز متغیر و... نمایش داده می شوند.

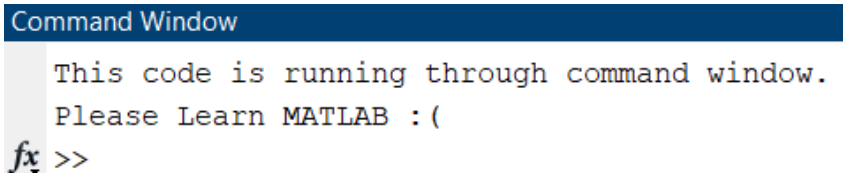
```
close all
clear
clc

a = 5;
b = [1 2 3 4 5];
c = [1 2 3 ; 4 5 6 ; 7 8 9];
```



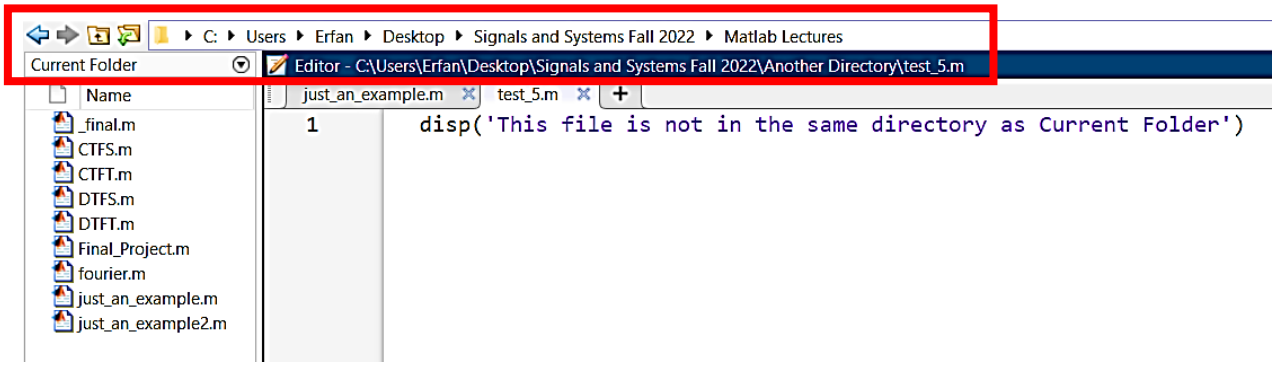
✓ برای اجرای کد خود کافیست بر روی گزینه ی 

کلیک کرده تا کد شما در قسمت **Command Window** اجرا شود.



نکته مهم :

گاهی اوقات شـــما فایلی را قســـمتی از کامپیوتر خود باز میکنید که در **Current Folder** موجود نیست.



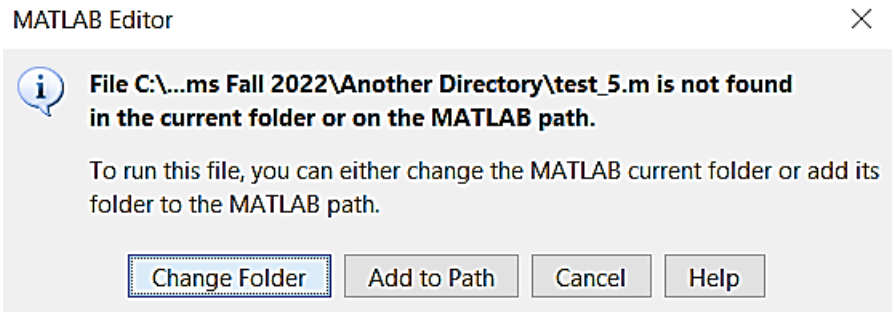
دایرکتوری **Current Folder**:

C:\Users\Erfan\Desktop\Signals and Systems Fall 2022\Matlab Lectures

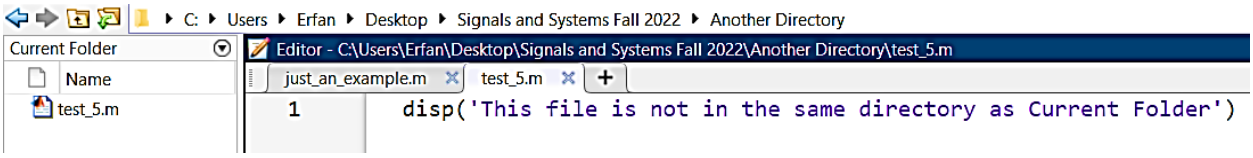
دایرکتوری فایل باز شده:

C:\Users\Erfan\Desktop\Signals and Systems Fall 2022\Another Directory

همانطور که مشـــاهده می‌کنید، فایل just_an_example.m در Current Folder **r** وجود دارد اما فایل test_5.m این گونه نیست زیرا در دایرکتوری دیگری از کامپیوتر باز شده است. در این هنگام، اگر شما کد خود را اجرا کنید یک error با این عنوان که "**دایرکتوری فایلی که می‌خواهید اجرا کنید در مسیر Current Folder یا مسیر MATLAB وجود ندارد**" مشاهده خواهید کرد.



با انتخاب گزینه **Change Folder** مسیر **Current Folder** به مسیر دایرکتوری فایل مورد نظر تغییر می‌کند؛ به طوری که محتویات قسمت **Current Folder** شامل فایل هایی می‌شود که در دایرکتوری فایل در حال اجرا وجود دارند.



همانطور که مشاهده می‌کنید دایرکتوری **Current Folder** تغییر کرده است. با انتخاب گزینه **Add to Path** نیز می‌توان دایرکتوری فایل مورد نظر را به مسیر **MATLAB** اضافه کرد.

شروع کد نویسی در متلب

تا اینجا با محیط متلب و برخی از پنجره های آن آشنا شدیم و همچنین یاد گرفتیم که با **m** فایل ها چگونه کار کنیم. دیدیم که پنجره های **Command window**، **Workspace** و... چه کاربرد های دارند. به عنوان مثال **Workspace**تمام متغیر های تعریف شده در طول اجرای یک برنامه را در خود ذخیره می کند. این ویژگی متلب برای ما مفید است اما وقتی بخواهیم یک برنامه دیگری در متلب اجرا کنیم، مسلما برنامه‌ی جدید با متغیرهایی که از قبل ذخیره شده‌اند تداخل داشته و نتیجه مطلوب حاصل نمی‌شود. یک برنامه خوب برای شروع به سه دستور زیر نیاز دارد تا با اطلاعات ذخیره شده قبلی تداخلی نداشته باشد.

close all	این دستور پنجره های شکلی که قبلا در متلب باز شده اند را می بندد.
clear	این دستور تمامی متغیرهایی که قبلا در متلب تعریف شده اند را پاک می کند. (**Workspace** را خالی می کند)
clc	این دستور تمامی اطلاعاتی که قبلا در پنجره **command** نمایش داده شده است را پاک می کند

نکته : بنابر کاربرد این سه دستور در ترتیب وارد کردن آن ها بهتر است دستور **clc** به عنوان سومین دستور وارد شود تا **Command Window** بطور کلی پاک شود.

just_an_example.m

```
1   close all
2   clear
3   clc
```

در مهندسی برق، سیگنال را کمیتی متغییر با زمان تعریف کردیم که اطلاعاتی درباره رفتار یا ویژگی یک پدیده در بر دارد ؛ روش های متعددی برای نمایش سیگنال ها معرفی شدند که دو مورد از آن ها را میخواهیم با برنامه نویسی متلب بررسی کنیم.

الف) روابط ریاضی

یکی از راه های نمایش یک سیگنال، استفاده از ضابطه ریاضی توصیف کننده‌ی آن است. به این منظور بهتر است ابتدا با انواع متغیر ها در متلب آشنا شویم:

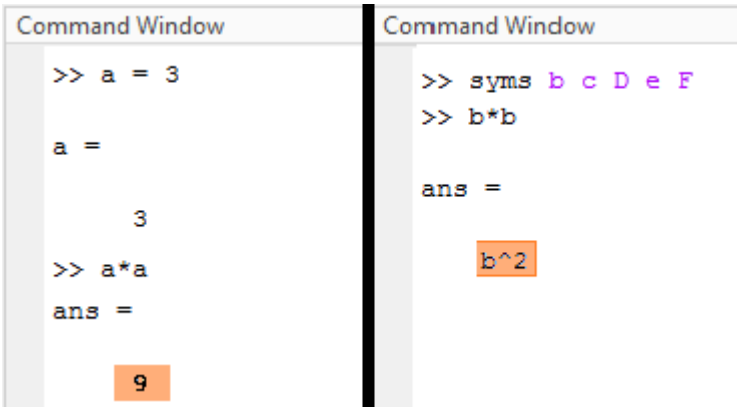
در زبان برنامه نویسی متلب، بر خلاف برخی از زبان ها نیاز نیست که نوع متغیر را از قبل مشخص کنیم؛ بلکه با مقادیری که در طول کد نویسی به متغیر داده می شود نوع آن تعیین می شود.

در متلب انواع متغیر ها را داریم به عنوان مثال:

- متغیر های عددی
- متغیر های رشته ای
- متغیر های سمبلیک

متغییر های نوع اول و دوم را می‌توان متغیر **معمولی** نام برد. **متغیر های معمولی** در واقع محل ذخیره‌ای برای اطلاعات هستند و با قرار دادن داده‌ای در یک متغیر معمولی و با فراخوانی اسم آن متغیر می‌توان از داده های ذخیره شده در آن استفاده کرد.

متغیر های سمبلیک: به این متغیر ها بر خلاف متغیر های معمولی، عدد یا مقدار خاصی نسبت داده نمی شود و فقط از نماد آن ها بصورت سمبلیک استفاده می شود. این متغیر ها همان **x** و **y** در ریاضی هستند که در دوران دبیرستان از آن ها برای تشکیل و حل معادلات ریاضی استفاده می کردیم. برای تعریف چنین متغیر هایی، از دستور **syms** بصورت زیر استفاده می شود:



پس از اینکه با انواع متغیر و نحوه تعریف و استفاده ازآن ها آشنا شدیم، می‌توانیم توابع ریاضی که توصیف کننده‌ی یک سیگنال هستند را در متلب تعریف کنیم و از آنها بهره ببریم.

در اینج با دو روش تعریف توابع ریاضی در متلب اشنا می شویم:

۱- تعریف تابع ریاضی با استفاده از **symfun**

```
f = symfun( 1 , 2 )
```

نکته: در این روش ابتدا باید متغیر یا آرگمان های ورودی تابع، به صورت **syms** تعریف شوند.

- **قسمت 1** : در این قسمت، ضابطه تابع مورد نظر را می نویسیم.
- **قسمت 2** : در این قسمت، آرگمان ورودی تابع را می نویسیم.

نکته: اگر تابع ما چند متغیره باشد در قسمت دوم دستور از یک ماتریس استفاده کرده و ارگمان های ورودی را در درایه های ماتریس قرار می دهیم.

✓ حال تابع ما در متغیر **f** ذخیره شده و میتوان انواع اعمال ریاضی(مقدار دهی، مشتق گیری، انتگرال گیری و...) بر روی آن انجام داد.

Command Window

```
>> syms t
>> f = symfun(t^2 + 3*t + 4 , t )

f(t) =

t^2 + 3*t + 4

>> f(2)

ans =

14
```

Command Window

```
>> syms x y
>> g = symfun (x^2 + y^2 ,[x,y])

g(x, y) =

x^2 + y^2

>> g(3,2)

ans =

13
```

۲- تعریف تابع ریاضی به روش فانکشن هندل (**Function handle**) :

```
f = @( 1 )( 2 )
```

نکته: در این روش نیازی نیست متغیر یا آرگمان های ورودی تابع به صورت **syms** تعریف شوند.

توجه: علامت @ یک اپراتور است که برای ایجاد تابع بصورت فانکشن هندل استفاده می‌شود.

- قسمت 1 : در این قسمت، آرگمان ورودی تابع را می نویسیم.
- قسمت 2 : در این قسمت، ضابطه تابع مورد نظر را می نویسیم.

✓ حال تابع ما در متغیر **f** ذخیره شده و میتوان انواع اعمال ریاضی(مقدار دهی , مشتق گیری, انتگرال گیری و.) بر روی آن انجام داد.

Command Window

```
>> f = @(t) (1/3*t^3)

f =

  function_handle with value:

    @(t) (1/3*t^3)

>> f(3)

ans =

     9
```

Command Window

```
>> g = @(w,z) [z*w + 2]

g =

  function_handle with value:

    @(w,z) [z*w+2]

>> g(5,2)

ans =

    12
```

دستور input

دستور input راه ارتباطی کاربر با برنامه است. وظیفه‌ی این دستور، دریافت ورودی از کاربر است.

نکته: این دستور دو نوع داده‌ی عددی و رشته‌ای را از کاربر دریافت می‌کند.

۱- **دریافت رشته از کاربر**

```
a = input( 1 , 2 )
```

- **قسمت 1** : در این قسمت رشته ای وارد می شود که پس از اجرای دستور این رشته به کاربر نمایش داده می شود. بهتر است این رشته توضیحات یا یک راهنمایی راجب داده ورودی برای کاربر باشد.
- **قسمت 2** : در این قسمت عبارت **<< "s" >>** نوشته میشود. **s** مخفف **string** یا رشته است.

نکته: در این حالت، برنامه هر کاراکتری (اعم از عدد و حروف و فاصله و # و : و ...) که کاربر وارد می‌کند را بعنوان رشته می‌شناسد.

- ✓ حال رشته از کاربر دریافت شده و در متغیر a ذخیره شده و قابل استفاده است.

۲- **دریافت عدد از کاربر**

```
a = input( 1 )
```

- ✓ در این حالت قسمت یک را مانند حالت قبل داریم وتنها تفاوت این است که باید قسمت دوم را خالی بگذاریم, این گونه میتوانیم عددی از کاربر دریافت کرده و از آن استفاده کنیم.

Command Window

```
>> a = input ('enter your name :','s');
enter your name :erfan
>> b = input ('enter your last name :','s');
enter your last name :yagoubi
>> c = input ('enter your age :');
enter your age :20
```

Command Window

```
>> a
a =
    'erfan'
>> b
b =
    'yagoubi'
>> c
c =
    20
```

- **دستور** disp

این دستور مقدار یک متغیر را در خروجی نمایش می‌دهد. این متغییر می‌تواند عددی یا رشته‌ای باشد .

```
Command Window
>> Full_name = 'erfan yagoubi';
>> disp(Full_name)
erfan yagoubi
>> age = 20;
>> disp(age)
    20
```

توجه: این دستور فقط یک ورودی دارد، یعنی در حالت عادی می‌تواند فقط مقدار یک متغیر را چاپ کند.

? **برای چاپ مقادیر چندین متغیر چه باید کرد؟؟؟**

پاســخ: برای این کار ماتریســی را به عنوان ورودی **disp** در نظر می‌گیریم و متغیرها را در درایه های ماتریس قرار می‌دهیم؛ بدین صورت می‌توان مقادیر چند متغیر را در کنار هم چاپ کرد.

```
Command Window
>> a = 'Hi ';
>> b = ',';
>> c = 'class!!';
>> disp(['----',a,b,c,'----'])
----Hi ,class!!----
```

تذکر: توجه کنید که جنس درایه های ماتریس ورودی **disp** باید از یک نوع باشد.

? **برای چاپ مقدار یک متغیر عددی در کنار رشته چه باید کرد؟؟؟**

پا سخ: برای چاپ مقدار متغیر عددی در کنار ر شته از د ستور **num2str** ا ستفاده کرده و عدد را به رشته تبدیل میکنیم.

```
s = num2str( 1 )
```

➢ **قسمت 1** : در این قسمت مقدار عدد یا اسم متغیری که عدد در آن ذخیره شده وارد می‌شود.

✓ حال عدد وارد شده بصورت یک رشته در s ذخیره شده و قابل استفاده است.

```
Command Window
>> a = input ('enter your date of birth :');
enter your date of birth :2002
>> age = 2022 - a ;
>> age_s = num2str(age);
>> disp(['You are, ' , age_s , ' years old.'])
You are, 20 years old.
>>
```

- **ایده ای برای گرفتن سیگنال از کاربر با استفاده از دستور** str2func

ابتدا با دستور **str2func** آشنا شویم:

```
f = str2func([ '@( 1 )', 2 ])
```

➢ **قسمت 1** : در این قسمت آرگمان ورودی تابع را می نویسیم.

➢ **قسمت 2** : در این قسمت ضابطه تابع به صورت رشته وارد میشود.

➢ حال این دستور ضابطه‌ی تابعی که به صورت رشته در قسمت **2** وارد شده را بر حسب آرگمان های ورودی مشخص شده در قسمت **1** بصورت فانکشن هندل به یک تابع ریاضی تبدیل کرده و در متغیر **f** ذخیره می‌کند.

✓ از آنجا که می‌دانیم چگونه می‌توان با دستور **input** یک رشته را از کاربر دریافت کرد، ضابطه‌ی سیگنال را از کاربر دریافت و با استفاده از دستور **str2func** آن را به یک فانکشن تبدیل می‌کنیم.

```
Command Window
>> f_s= input('please enter your signal: ','s');
please enter your signal: t^2 + 3
>> f=str2func(['@(t)',f_s]);
>> f(3)

ans =

    12

>> f(2)

ans =

     7
```

دستور if, elseif, else

در بسیاری از مواقع، ما نیاز داریم برای ورودی هایی را که از کاربر دریافت می‌کنیم شرط ایجاد کنیم تا برنامه بر اساس ورودی کاربر، عملکرد متفاوتی داشته باشد. در این صورت از دستور **if** استفاده می‌کنیم. برای بررسی شرط های بیشتر می‌توانیم از دستور های **elseif** و **else** استفاده کنیم و در پایان حتما باید دستور **end** را بنویسیم تا برنامه متوجه شود که دستورات شرطی ما به پایان رسیده است.

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

نحوه کار شرط ها بدین صورت است که اگر عبارت شرطی درست باشد(اصطلاحا مقدار True داشته باشد)، دستورات بعدی اجرا می‌شوند؛ ولی اگر عبارت شرطی نادرست باشد(اصطلاحا مقدار False داشته باشد)، دستورات بعد اجرا نشده و برنامه به صورت خودکار به ادامه‌ی کار میپردازد.

```
identify_number.m
1   close all
2   clear
3   clc
4
5   num = input('Enter a number: ');
6
7   if num == 5
8       disp('The number you entered is 5')
9   elseif num > 5
10      disp('The number you entered is greater than 5')
11  else
12      disp('The number you entered is less than 5')
13  end
```

```
Command Window
Enter a number: 1
The number you entered is less than 5
------------------------------------------------
Enter a number: 9
The number you entered is greater than 5
------------------------------------------------
Enter a number: 5
The number you entered is 5
fx >>
```

نحوه عملکرد برنامه:

همانطور که قبلا گفته شـــد، در خط ۱ تا ۳ به ترتیب پنجره های اضـــافی بســـته و مقادیر ذخیره شـــده در Workspace و مطالب از قبل نوشته در Command Window پاک می‌شوند.

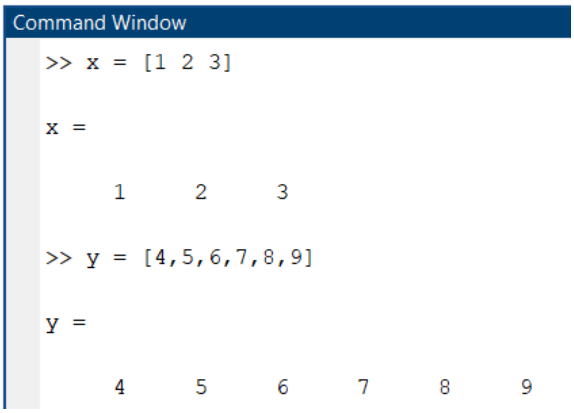
در خط ۵، از کاربر می‌خواهیم یک عدد بعنوان ورودی ارســـال کند. در خط ۷ پردازش بر روی عددی که کاربر بعنوان ورودی ارسال کرده آغاز می‌شود. بدین صورت که اگر عدد وارد شده برابر با 5 بود به کاربر بگو "عددی که وارد کردی 5 اســت."(خط ۷ و ۸)؛ در غیر این صـورت اگر عددی وارد شـده بزرگتر از 5 بود به کاربر بگو "عددی که وارد کردی بزرگتر از 5 ا ست."(خط ۹ و ۱۰) و در غیر این صورت به کاربر بگو"عددی که وارد کردی کوچکتر از 5 است."

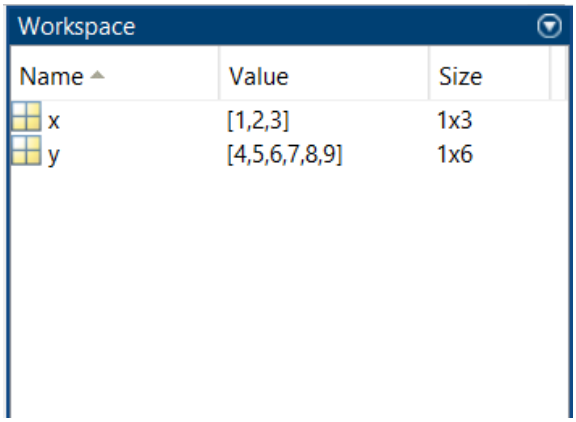
تعریف ماتریس و آرایه:

آرایه و ماتریس همانند متغیر های معمولی بوده اما با این تفاوت که آرایه و ماتریس ابعاد بیشتری نسبت به متغیر های معمولی دارند.

برای ایجاد یک آرایه به روش زیر عمل می‌کنیم:

- نوشتن اسم آرایه و قرار دادن علامت =
- آغاز آرایه با علامت براکت باز [
- تعیین ستون بعدی با علامت , یا فاصله
- پایان آرایه با علامت براکت بسته]





بدین ترتیب، یک ماتریس تک سطری خواهیم داشت که اصطلاحا به آن بردار یا آرایه عددی(آرایه) گفته می‌شود. همانطور که ملاحظه می‌کنید بردار x، دارای ۱ سطر و ۳ ستون و همچنین بردار y، دارای ۱ سطر و ۶ ستون می‌باشد.

نکته: برخلاف دیگر زبان های برنامه نویسی، در متلب اندیس آرایه ها از ۱ شروع می‌شود.

```
Command Window
>> x = [3 5 9 6]

x =

     3     5     9     6

>> x(0)
Array indices must be positive integers or logical values.

>> x(1)

ans =

     3
```

برای ایجاد یک ماتریس چند سطری به روش زیر عمل می‌کنیم:

- نوشتن اسم آرایه و قرار دادن علامت =
- آغاز آرایه با علامت براکت باز [
- تعیین ستون بعدی با علامت , یا فاصله
- تعیین سطر بعدی با ;
- پایان آرایه با علامت براکت بسته]

```
Command Window
>> x = [1 2 3;4 5 6;7 8 9;]

x =

     1     2     3
     4     5     6
     7     8     9

>> y = [1 2 3 4 5;6 7 8 9 10]

y =

     1     2     3     4     5
     6     7     8     9    10
```

Workspace

Name	Value	Size
x	[1,2,3;4,5,6;7,8,9]	3x3
y	[1,2,3,4,5;6,7,8,9,10]	2x5

همانطور که ملاحظه می‌کنید ماتریس x، یک ماتریس 3×3(دارای ۳ سطر و ۳ ستون) و همچنین ماتریس y، یک ماتریس 2×5(دارای ۲ سطر و ۵ ستون) می‌باشد.

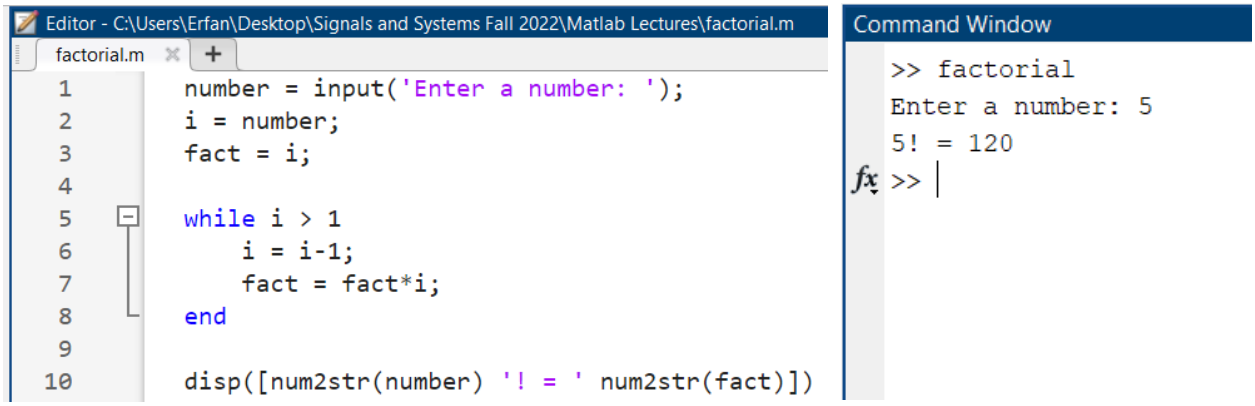
حلقه ها:

در همه‌ی زبان های برنامه نویسی از حلقه ها برای تکرار انجام یک کار معین ا ستفاده می شود. زبان برنامه نویسی متلب نیز از این قاعده مستثنا نیست شامل دو حلقه‌ی بسیار پرکاربرد `while` و `for` می‌باشد.

حلقه‌ی `while`:

پیش از این با شرط ها آ شنا شدیم و دیدم که در صورتی که عبارت شرطی مقدار `True` دا شته با شد، برنامه د ستورات وابسته به شرط را اجرا می‌کند. در حلقه‌ی `while` هم تا زمانی که شرط حلقه مقدار `True` دا شته باشد، دستورات زیرمجموعه‌ی حلقه اجرا می‌شوند.

```
while expression
    statements
end
```



برای درک بهتر این موضوع نگاهی به برنامه‌ی محاسبه‌ی فاکتوریل می‌اندازیم:

در ابتدا از کاربر می‌خواهیم عددی را که خوا هان محاسبه‌ی فاکتور یل آن است به عنوان ورودی وارد نماید(`number`). سپس متغیر شرط حلقه(`i`) و متغیری که در آن جواب نهایی ذخیره می‌شود(`fact`) را ایجاد می‌کنیم. د ستورات متعلق به حلقه‌ی `while` تنها در زمانی اجرا می شوند که `i > 1` با شد. در ابتدا `i = number` هست(عدد ورودی کاربر) و در حلقه‌ی `while` پس از هربار اجرا شدن دستورات حلقه، ۱ واحد از مقدار `i` کم شده تا در نهایت `i = 1` شود. در این هنگام دیگر شرط حلقه درست نیست(دیگر `i > 1` نیست) و دستورات متعلق به حلقه دیگر اجرا نمی‌شوند.

حلقه for:

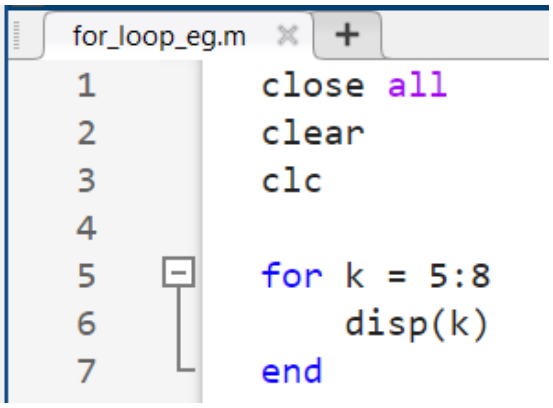
این نوع حلقه زمانی استفاده می‌شود که ما میخواهیم یک دستور به تعداد معینی اجرا شود. سینتکس آن به روش زیر است:

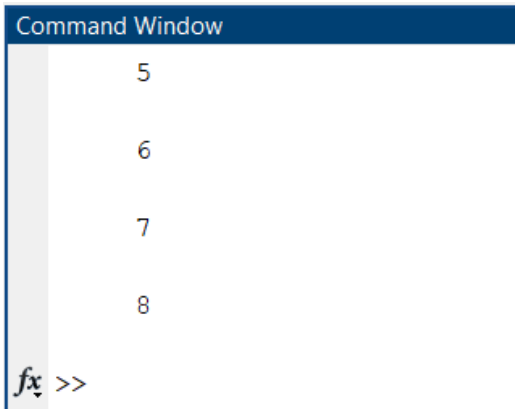
```
for index = values
   statements
end
```

نحوه مقدار دهی index

- **index = initVal:endVal**

در این روش شما اس متغیر حلقه‌ی خود را به جای index، مقدار اولیه‌ی و نهایی آن را به ترتیب بجای initVal و endVal قرار می‌دهید. گام پیمایش حلقه در این روش ۱ بوده و تا زمانی که مقدار index بزرگتر از مقدار initVal نشده، تکرار حلقه انجام می‌شود.





همانطور که مشاهده می‌کنید، متغیر k از 5 تا 8 مقدار گرفته و حلقه با گام یک تکرار شده تا جایی که دیگر مقدار k>8 شده و تکرار حلقه خاتمه یافته است.

- **index = initVal:step:endVal**

for_loop_eg.m

```
1  close all
2  clear
3  clc
4
5  for k = 1:0.2:2
6      disp(k)
7  end
```

Command Window

```
    1

    1.2000

    1.4000

    1.6000

    1.8000

    2
```

- **index = valArray**:

در این روش کافی ست برداری از اعداد را به متغیر حلقه مقداردهی کنیم. در این حالت، متغیر حلقه در هر بار پیمایش برابر با عنصر nم بردار خواهد شد(n شماره اندیس عناصر بردار است).

for_loop_eg.m

```
1  close all
2  clear
3  clc
4
5  for k = [5 9 1 6]
6      disp(k)
7  end
```

Command Window

```
    5

    9

    1

    6
```

توابع:

در بسیاری از مواقع نیاز داریم برای تمیز نویسی و جلوگیری از شلوغی کد، آن را بصورت فانکشنال یا به زبان دیگر با استفاده از توابع بنویسیم. در واقع توابع به ما کمک می‌کنند تا از بازنویسی دستورات تکراری پرهیز کنیم و کدی ساده تر و با تعداد خط کمتر داشته باشیم.

برای تعریف توابع به روش زیر عمل می‌کنیم:

```
function [y1,...,yN] = myfun(x1,...,xM)
        statement
end
```

- **function**: برای تعریف یک تابع، لازم است حتما از کلمه‌ی کلیدی function استفاده کنیم تا به برنامه بگوییم دستوراتی که از این پس نوشته می‌شوند مربوط به بدنه‌ی تابع خواهند بود.
- **[y1,...,yN]**: در این قسمت متغیر های خروجی تابع را بصورت یک بردار تعریف می‌کنیم.

نکته: اگر تابع ما فقط یک خروجی داشت نیازی به براکت ها نیست.

- **Myfun**: در این قسمت کافیست نام دلخواهی برای تابع مورد نظرتان بگذارید.

نکته ۱: نام تابع باید حتما با حروف انگلیسی شروع شده و می‌تواند شامل حروف انگلیسی، اعداد و کاراکتر underscore(_) باشد.

نکته ۲: در تعریف نام تابع از فاصله استفاده نکنید!

نکته ۳: از اسامی توابع داخلی متلب و همچنین دستورات متلب برای اسم تابع استفاده نکنید!

- **(x1,...,xM)**: در این قسمت باید تمامی متغیر های مورد نیاز که تابع به عنوان ورودی دریافت می‌کند را تعریف کنید.

نحوه فراخوانی توابع:

برای فراخوانی توابع کافیست متغیری تعریف کرده و مقدار آن را برابر با اسم تابع به همراه ورودی های مورد نیاز آن قرار داد.

نحوه ذخیره سازی توابع:

در برنامه نویسی متلب، توابع را می‌توان به ۲ روش ذخیره کرد. روش اول زمانی استفاده می‌شود که شما می‌خواهید از تابع فقط در یک اسکریپت(m فایل) استفاده کنید. در این صورت باید توابع را در انتهای کد تعریف کرده و در خط های بالاتر آن ها را فراخوانی کنید.

روش دوم زمانی استفاده می‌شود که شما نیاز دارید از توابعی که تعریف کرده‌اید در چندین اسکریپت استفاده کنید. در این حالت شما نیاز دارید یک اسکریپت جداگانه ایجاد کرده و تابع خود را در آن تعریف کنید. سپس فایل اسکریپت آن را در محل قرارگیری کد ها متلبی که از این تابع استفاه می‌کنند ذخیره کنید.

نکته: در هنگام ذخیره کردن فایل اسکریپت تابع از همان اسم تابع برای اسم فایل اسکریپت استفاده کنید. در غیر این صورت با خطا مواجه خواهید شد.

روش اول:

```
func_test.m
1   close all
2   clear
3   clc
4
5   number = add_two_nums(2,3);
6   disp(number)
7
8   function a = add_two_nums(x, y)
9   a = x + y;
10  end
```

```
Command Window
     5
fx >>
```

همانطور که ملاحظه می‌کنید تابع `add_two_nums` که شامل ۲ ورودی به نام های `x` و `y` و یک خروجی به نام `a` هست در پایین ترین قسمت کد قرار گرفته و در خط های بالاتر(خط ۵) فراخوانی شده و مقدار خروجی تابع در متغیری به نام `number` قرار گرفته است.

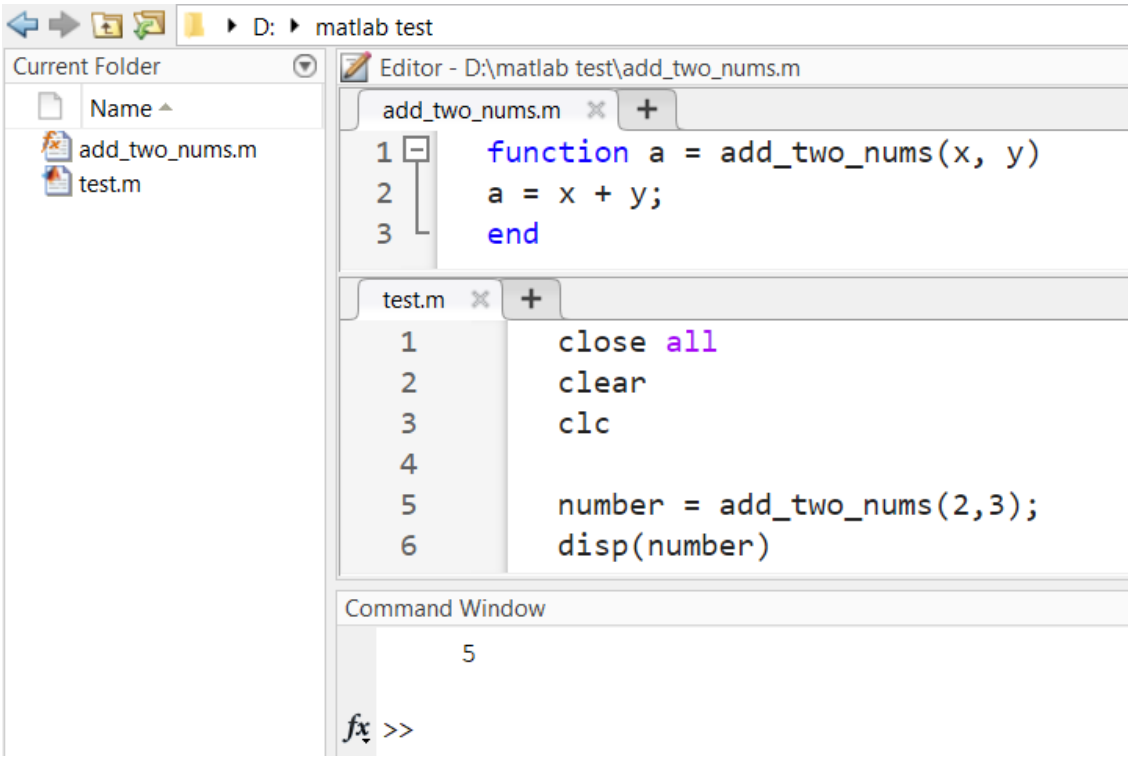
چنانچه به تعداد بیشتری از خروجی نیاز داشتیم به این صورت عمل می‌کنیم:

```
func_test.m
1   close all
2   clear
3   clc
4
5   [add, sub] = add_and_sub(5,3);
6   disp(add)
7   disp(sub)
8
9   function [a, s] = add_and_sub(x, y)
10  a = x + y;
11  s = x - y;
12  end
```

```
Command Window
     8
     2
fx >>
```

روش دوم:

هر دو فایل در یک مسیر قرار گرفته‌اند.



از تابع `add_two_nums` که آن را با همین نام و در مسیری یکسان با مسیر ذخیره سازی فایل `test` ذخیره کرده‌ایم در فایل `test` استفاده کردیم تا شلوغی از پیچیدگی کد خود بکاهیم.

ب) ترسیم شکل

یکی دیگر از راه های نمایش یک سیگنال، رسم شکل آن است. در متلب برای رسم یک سیگنال از دستور های مختلفی می‌توان استفاده کرد. اینجا دو دستور پرکاربرد زیر بررسی می شوند:

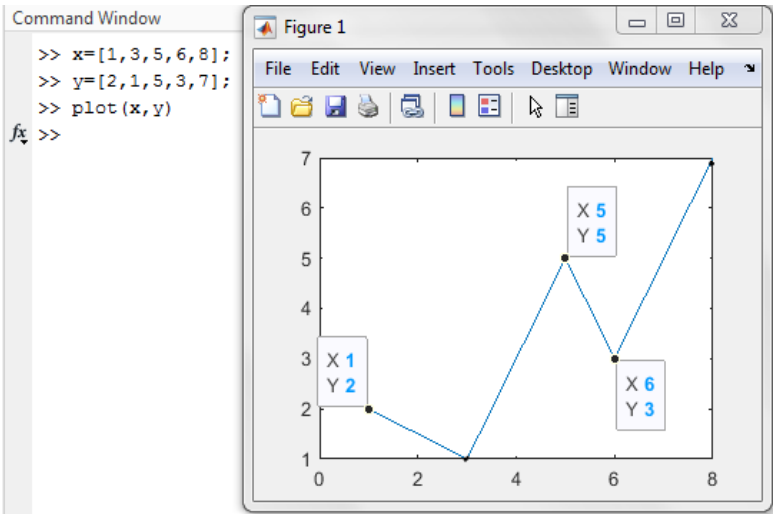
1- دستور plot 2 - دستورstem

- دستور plot

این دستور بر پایه روش درونیابی خطی کار می‌کند. بدین صورت که داده های مربوط به محور افقی و عمودی را به صورت زوج مرتب دریافت و سپس توسط یک خط شیب دار، نقطه‌ای به مختصات هر زوج مرتب را به نقطه بعدی وصل می‌کند تا نمودار مورد نظر حاصل شود.

```
plot(x , y)
```

- **ماتریس x** : این بردار حاوی داده های محور زمانی هست.
- **ماتریس y** : این بردار حاوی مقادیر سیگنال، متناظر با بردار **x** هست.



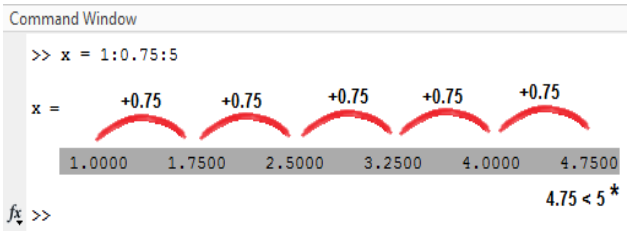
همان طور که مشاهده می شود دستور **plot** داده ها را بصورت **(1,2) (3,1) (5,5) (6,3) (8,7)** مرتب کرده و سپس به ترتیب آن ها را به یک دیگر متصل می کند.

- عملگر << : >>

برای رسم نمودار با دستور **plot** عملگر دو نقطه بسیار پر کاربرد است. یکی از کاربرد های این عملگر تعریف بازه های عددی و تصاعد های حسابی است.

```
x= a : d : b
```

- a : ابتدای دنباله عددی
- b : انتهای دنباله عددی
- d : گام یا قدر نسبت

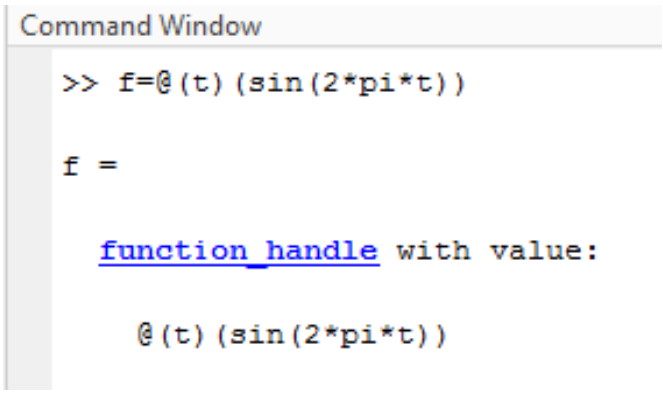


(E.g. سیگنال $f = \sin(2\pi t)$ را در بازه $-2 < t < 2$ رسم کنید.

برای رسم یک سیگنال با استفاده از دستور **plot** نیاز داریم متغیری مثل **x** را بصورت برداری از زمان بازه مورد نظر بسازیم. اما می‌دانیم که بازه مورد نظر، یک بازه پیوسته با تعداد بی نهایت عدد است؛ در این حالت، باید متغیر **x** را به گونه‌ای انتخاب کنیم که هم تعداد آن محدود باشد و هم شکل سیگنال به خوبی نمایش داده شود. برای این کار از عملگر دو نقطه استفاده می‌کنیم:

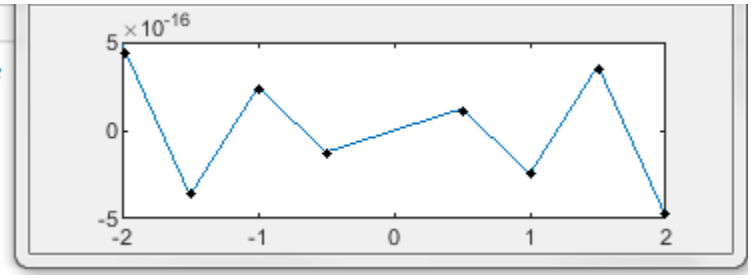
```
x = -2 : d : 2
```

ابتدا سیگنال **f** را به یکی از روش های تعریف تابع می سازیم. سپس از عملگر : استفاده کرده و **x** را برای گام های مختلف تعریف میکنیم و سیگنال را رسم می کنیم.



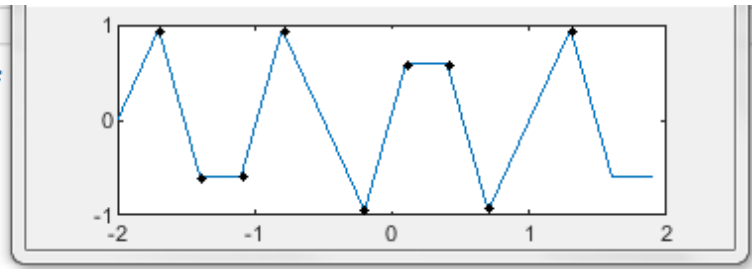
d = 0.5

```
>> x = -2:0.5:2;
>> y =f(x);
>> plot(x,y)
>>
```



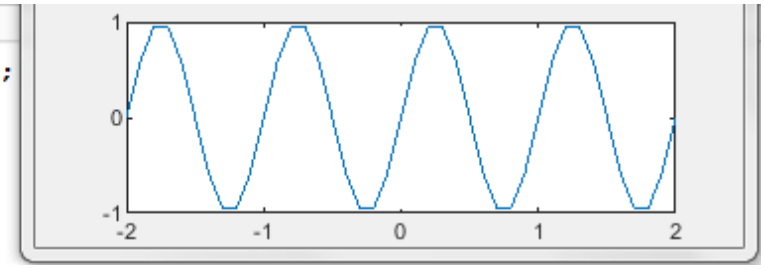
d = 0.3

```
>> x = -2:0.3:2;
>> y = f(x);
>> plot(x,y)
>>
```



d = 0.1

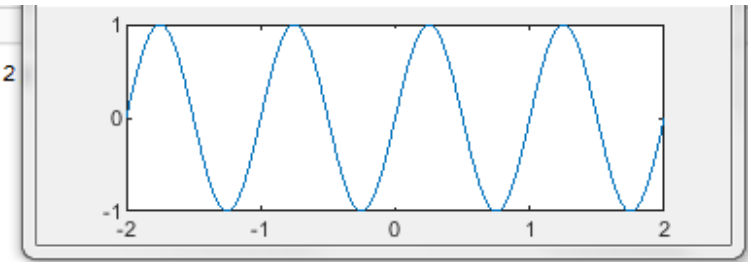
```
>> x = -2:0.1:2;
>> y = f(x);
>> plot(x,y)
>>
```



به صورت شهودی دیدیم که با کوچک کردن اندازه گام برای بردار **x** ، دستور **plot** یک شکل دقیق تر و نزدیکتر به سیگنال اصلی را رسم می کند؛ اندازه یک گام مطمئن برای استفاده از دستور **plot** برای رسم سیگنال های پیوسته حدود **0.01** است. با این گام، بردار **x** را می‌توان یک بازه پیوسته فرض کرد.

d = 0.01

```
>> x = -2:0.01:2
>> y = f(x);
>> plot(x,y)
>>
```



- **دستور subplot**

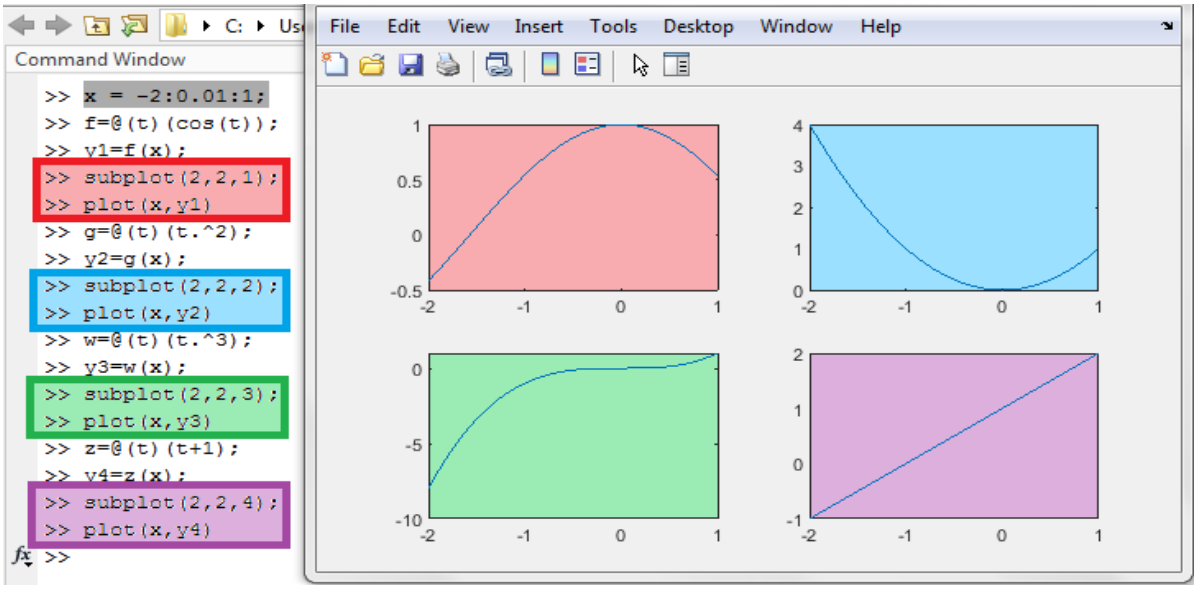
نرم افزار متلب برای اجرای هر دستور **plot** پنجره‌ای جداگانه باز می‌کند؛ مثلا اگر بخواهیم ۴ نمودار را در متلب رسم کنیم، نرم افزار هر نمودار را در پنجره‌ای جداگانه رسم می‌کند. بنابراین، ما به ۴ پنجره نیاز خواهیم داشت. این امر منجر به شلوغی و در هم ریختگی صفحه‌ی کامپیوتر شده و در نتیجه مقایسه‌ی نمودار ها با یکدیگر دشوار می‌گردد. برای جلوگیری از این درهم ریختگی و مقایسه بهتر نمودار ها می‌توان از دستور **subplot** استفاده کرد.

با استفاده از این دستور می‌توان یک **Figure** را بصورت ماتریسی به چند بخش تقسیم کرده و سپس هر نمودار را در درایه‌ای از آن رسم می‌کنیم.

```
subplot(a , b , c)
```

- a : تعداد سطر Figure
- b : تعداد ستون Figure
- c : بخشی که نمودار در آن رسم می شود.

E.g. سیگنال های **f = cos(t)، g = t^2، w = t^3، z = t+1** در یک **figure** رسم کنید. (**-2 < t < 1**)



- **دستور stem**

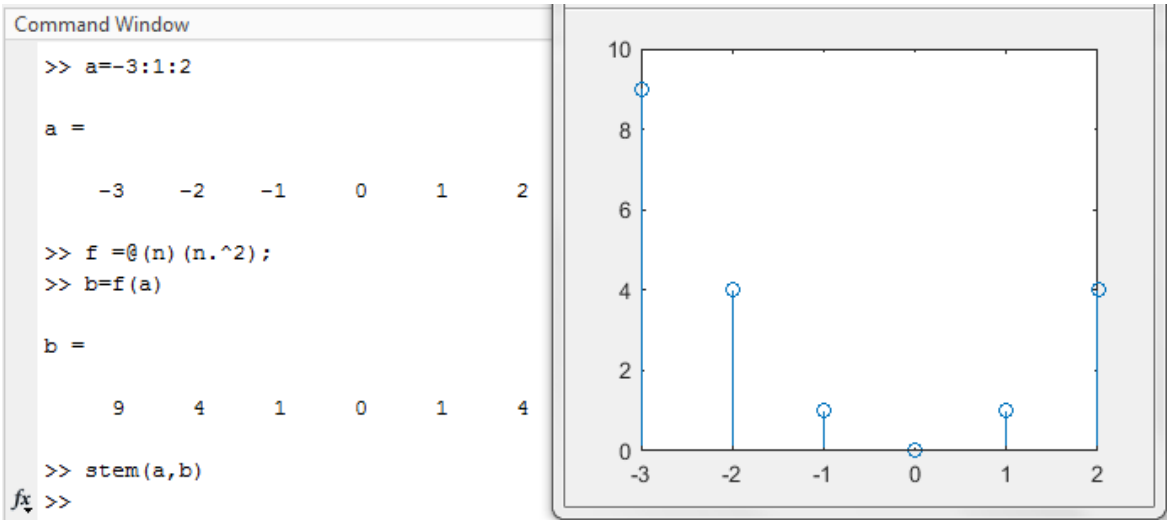
از دستور stem برای رسم سیگنال های گسسته در زمان استفاده می‌کنیم. این دستور سینتکس یکسانی با دستور plot دارد.

```
stem(a , b)
```

➢ این دستور مقادیر بردار b را به ترتیب به مقادیر بردار a نسبت می‌دهد و به صورت یک نمودار با میله و توپ نمایش میدهد.

(E.g. نمودار $f = n^2$ را برای $-3 \le n \le 2$ رسم کنید.

$D_n = -3, -2, -1, 0, 1, 2$



- **روش های درونیابی**

یکی از دستور های معروفی که برای درونیابی در متلب استفاده می شود دستور **interp** است. از دستور **interp1** نیز برای درونیابی سیگنال های تک متغیره استفاده می‌شود.

```
z = interp1 ( x , y , t ,'Method')
```

➢ **ماتریس x** : این بردار حاوی زمان های متناظر با نقاط نمونه برداری شده است.

➢ **ماتریس y** : این بردار حاوی مقادیر متناظر با نقاط نمونه برداری شده است.

- **ماتریس t** : این بردار حاوی زمان هایی است که می‌خواهیم مقادیر متناظر با آن ها را درونیابی کنیم.
- **'Method'** : در ورودی چهارم دستور، روش درونیابی را طبق جدول زیر انتخاب میکنیم.

'Method'	Description
'linear'	در این روش مقدار درونیابی شده هر زمان، برابر مقدار خط گذرنده از دو نقطه‌ی کناری است. (درونیابی خطی **LI**)
'previous'	در این روش مقدار درونیابی شده هر زمان، برابر با مقدار نقطه‌ی قبلی است. (نگهدارنده مرتبه صفر **ZOH**)

نکته۱: اگر ورودی چهارم خالی باشد متلب به صورت پیش فرض درون یابی خطی انجام می دهد.

نکته۲: درونیابی به روش نگهدارنده مرتبه یک (**FOH**) جزء روش های دستور **interp1** نیست.

- **ماتریس z** : در نهایت، مقادیر درون یابی شده در این ماتریس به صورت یک بردار ذخیره شده و قابل استفاده می‌باشد.

یادداشت:

.....

.....

.....

.....

.....