



Principi di Bioingegneria

A.A. 2022/23

Lezione 4

Intro MATLAB 2

Vincenzo Catrambone, PhD

vincenzo.catrambone@unipi.it

Basi di programmazione in MATLAB

Figure

A TRUE reminder

x	y	~x	x y	x && y	xor(x,y)
true	true	false	true	true	false
true	false	false	true	false	true
false	false	true	false	false	false

Operators	Precedence
parentheses: ()	highest
transpose and power ', ^	
unary: negation (—), not (~)	
multiplication, division *, /\	
addition, subtraction +, —	
colon operator :	
relational <, <=, >, >=, ==, ~=	
and &&	
or	
assignment =	lowest

Assume that there is a variable x that has been initialized. What would be the value of the following expression?

$3 < x < 5$

If the value of x is 4? What if the value of x is 7?

What about $5 < x > 0$

Think about what would be produced by the following expressions, and then type them in to verify your answers.

$4 > 3 + 1$

$'e' == 'd' + 1$

$3 < 9 - 2$

$(3 < 9) - 2$

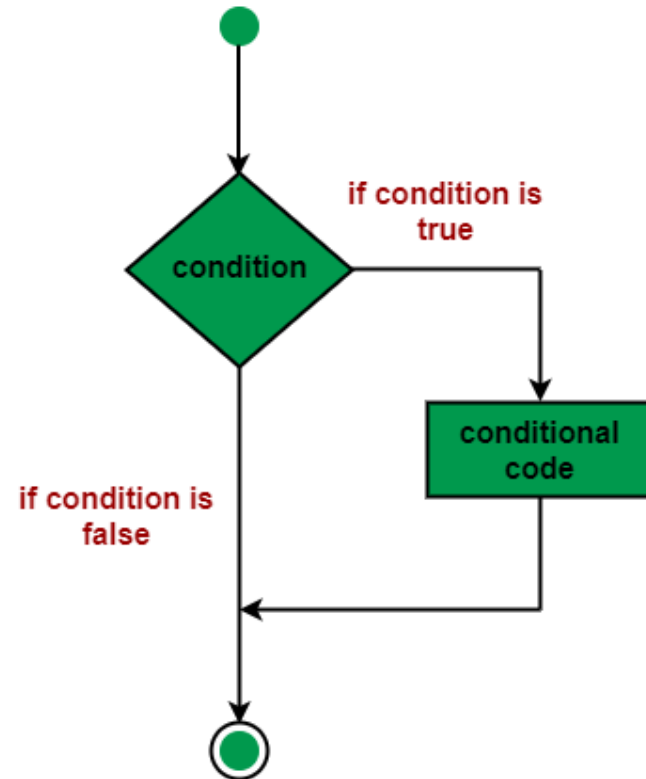
$4 == 3 + 1 \ \&\& \ 'd' > 'c'$

$3 >= 2 \ || \ 'x' == 'y'$

$\text{xor}(3 >= 2, 'x' == 'y')$

Condizioni di controllo: if ... end

Flow Diagram of If....end Statement



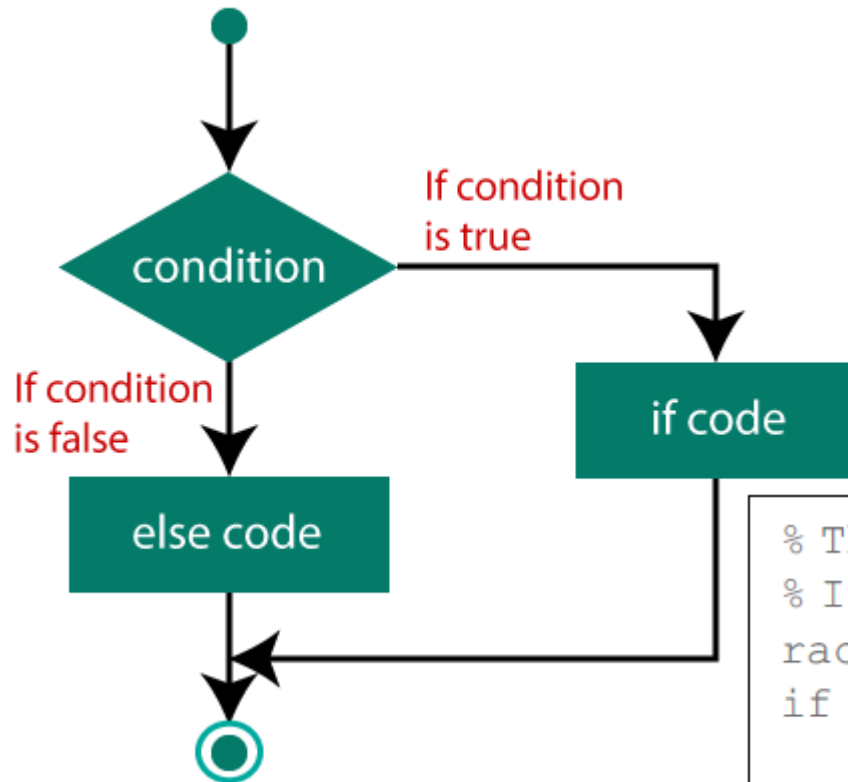
```
>> num = -4;  
>> if num < 0  
    num = abs(num)  
end  
num =  
4
```

```
>> num = 5;  
>> if num < 0  
    num = abs(num)  
end  
>>
```

```
% Prompt the user for a number and print its sqrt  
num = input('Please enter a number: ');  
% If the user entered a negative number, change it  
if num < 0  
    num = abs(num);  
end  
fprintf('The sqrt of %.1f is %.1f\n', num, sqrt(num))
```

```
>> sqrtifexamp  
Please enter a number: -4.2  
The sqrt of 4.2 is 2.0
```

Condizioni di controllo: if ... else ... end



```
% program to check the number is even or odd
```

```
a = randi(100,1);
```

```
if rem(a,2) == 0  
    disp('an even number')
```

```
else  
    disp('an odd number')
```

```
end
```

```
if rand < 0.5  
    disp('It was less than .5!')
```

```
else  
    disp('It was not less than .5!')
```

```
end
```

```
% This script calculates the area of a circle
```

```
% It error-checks the user's radius
```

```
radius = input('Please enter the radius: ');
```

```
if radius <= 0
```

```
    fprintf('Sorry; %.2f is not a valid radius\n',radius)
```

```
else
```

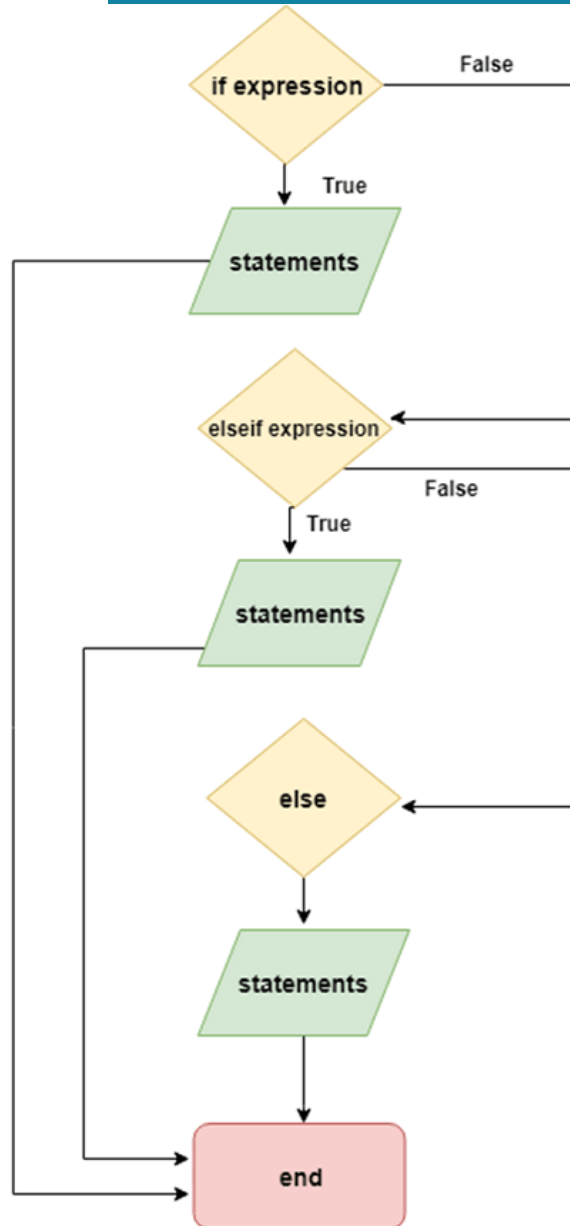
```
    area = calcarea(radius);
```

```
    fprintf('For a circle with a radius of %.2f,',radius)
```

```
    fprintf(' the area is %.2f\n',area)
```

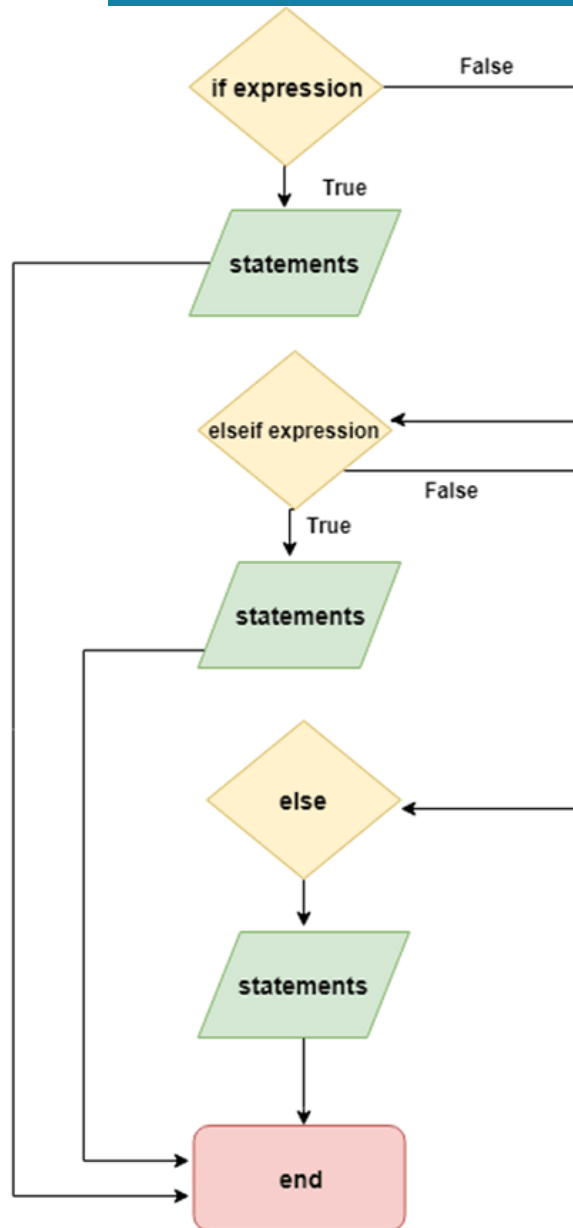
```
end
```

Condizioni di controllo: if ... elseif ... else ... end



```
% program to compare two numbers
% generate random number for your age
n = randi(100,1);
age = 23;
% check the number is greater than your age
if n > age
    disp('i am younger')
elseif n < age
    disp('you are younger')
else
    disp('we are of same age')
end
```

Condizioni di controllo: if ... elseif ... else ... end



In quanti modi diversi pensate di poter scrivere questo sistema?

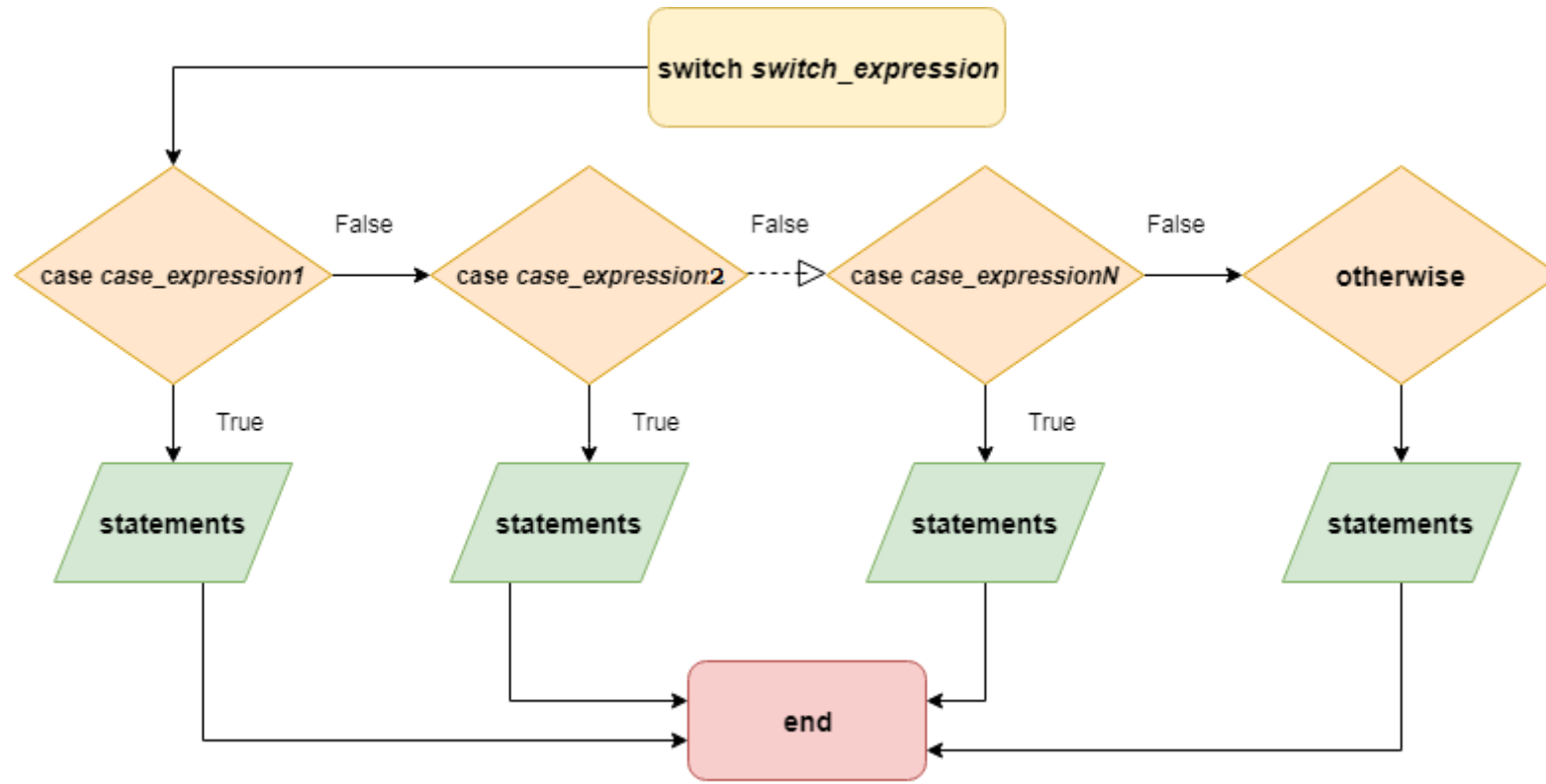
```
y = 1 if x < -1  
y = x^2 if -1 ≤ x ≤ 2  
y = 4 if x > 2
```

```
if x < -1  
    y = 1;  
else  
    if x ≤ 2  
        y = x^2;  
    else  
        % No need to check  
        % If we are here, x must be > 2  
        y = 4;  
    end  
end
```

```
if x < -1  
    y = 1;  
end  
if x ≥ -1 && x ≤ 2  
    y = x^2;  
end  
if x > 2  
    y = 4;  
end
```

```
if x < -1  
    y = 1;  
elseif x ≤ 2  
    y = x^2;  
else  
    y = 4;  
end
```

Condizioni di controllo: switch ... case ... end

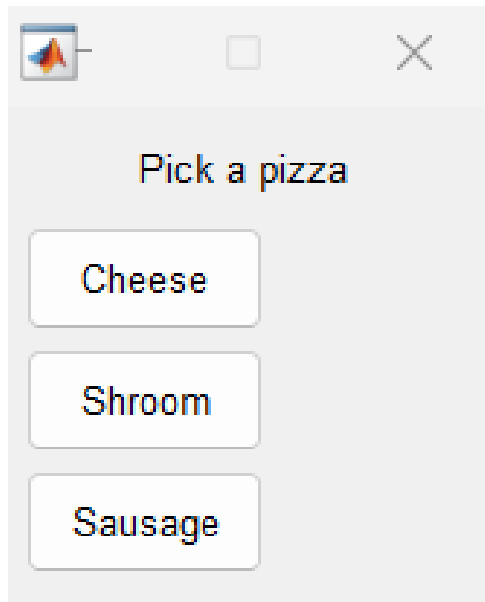


```
% First, error-check
if quiz < 0 || quiz > 10
    grade = 'X';
else
    % If here, it is valid so figure out the
    % corresponding letter grade using a switch
    switch quiz
        case 10
            grade = 'A';
        case 9
            grade = 'A';
        case 8
            grade = 'B';
        case 7
            grade = 'C';
        case 6
            grade = 'D';
        otherwise
            grade = 'F';
    end
end
end
```


Menu option

MATLAB has a built-in function called `menu` that will display a Figure Window with pushbuttons for the options. The first string passed to the `menu` function is the heading (an instruction), and the rest are labels that appear on the pushbuttons. The function returns the number of the button that is pushed.

```
>> mypick = menu('Pick a pizza', 'Cheese', 'Shroom', 'Sausage');
```



```
mypick = menu('Pick a pizza', 'Cheese', 'Shroom', 'Sausage');  
switch mypick  
    case 1  
        disp('Order a cheese pizza')  
    case 2  
        disp('Order a mushroom pizza')  
    case 3  
        disp('Order a sausage pizza')  
    otherwise  
        disp('No pizza for us today')  
end
```

How could the otherwise action get executed in this switch statement?

Loops: for ... end

```
% program to print multiples of first prime number between 1000 and 2000
% using for loop
pr = 0;
for k = 1000:2000
    if isprime(k)
        pr = k;
        disp(['The first prime number is : ', num2str(pr)])
        for m = pr:pr:pr*10
            disp(m)
        end

        break
    end
end
```

```
runprod = 1;
for i = 1:n
    runprod = runprod * i;
end
```

```
>> factorial(5)
ans =
    120
```

Loops: while ... end

```
a = 10;  
% while loop execution  
while( a < 20 )  
    fprintf('value of a: %d\n', a);  
    a = a + 1;  
end
```

Output:

```
value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19
```

```
% Prompts the user and echo prints the numbers entered  
% until the user enters a negative number  
  
inputnum=input('Enter a positive number: ');  
while inputnum >= 0  
    fprintf('You entered a %d.\n\n',inputnum)  
    inputnum = input('Enter a positive number: ');  
end  
fprintf('OK!\n')
```

Nested loops

We can use the nested for loop to display all the prime numbers from 1 to 100.

```
for i=2:100
    for j=2:100
        if(~mod(i, j))
            break; % if factor found, not prime
        end
    end
    if(j > (i/j))
        fprintf('%d is prime\n', i);
    end
end
```

Suppose *mat* is generic matrix, What is this code doing?

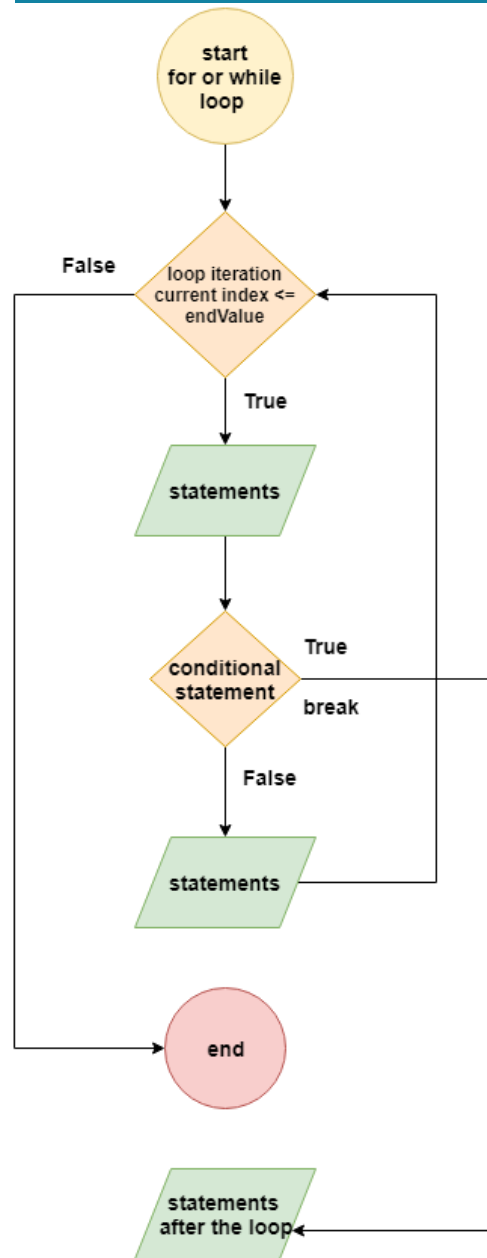
```
[ row col] = size(mat);
outsum = 0;

% The outer loop is over the columns
for i = 1:col
    for j = 1:row
        outsum = outsum + mat(j, i);
    end
end
end
```

Try

```
sum(mat(:))
sum(sum(mat))
sum(mat)
sum(mat, 2)
```

Break a loop



% program to terminate the execution on finding negative input

```
a = randn(4)
```

```
k = 1;
```

```
while k < numel(a)
```

```
if a(k) < 0
```

```
break
```

```
end
```

```
k = k + 1;
```

```
end
```

```
disp(['negative number :', num2str(a(k)), ',found at index: ', num2str(k),'hence the program terminated'])
```

```
a = 4x4
```

```
0.2398    -1.6118    0.8617    0.5812
```

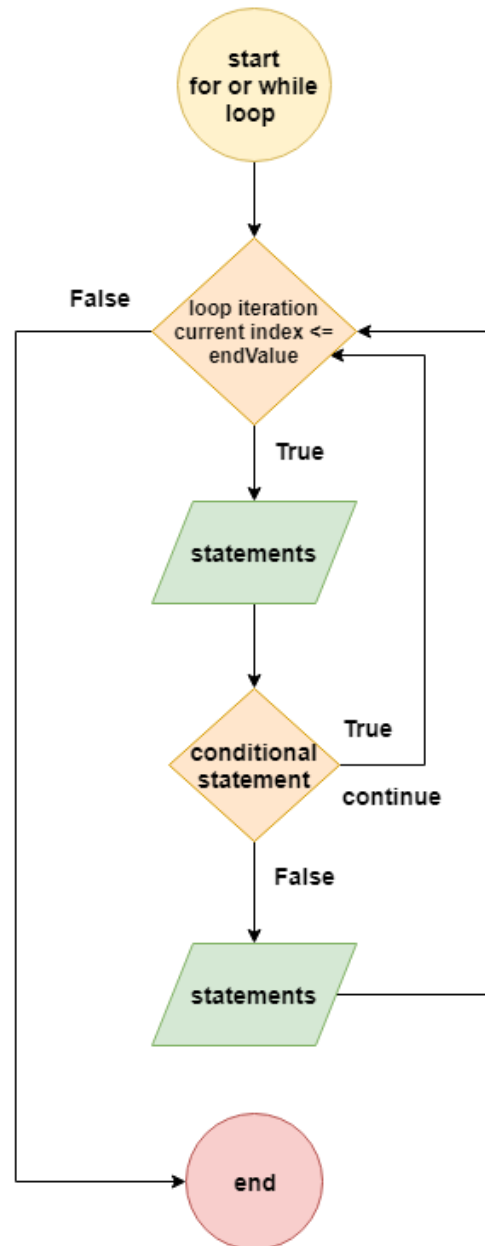
```
-0.6904   -0.0245    0.0012   -2.1924
```

```
-0.6516   -1.9488   -0.0708   -2.3193
```

```
1.1921    1.0205   -2.4863    0.0799
```

```
negative number :-0.69036,found at index: 2,hence the program terminated
```

Continue a loop

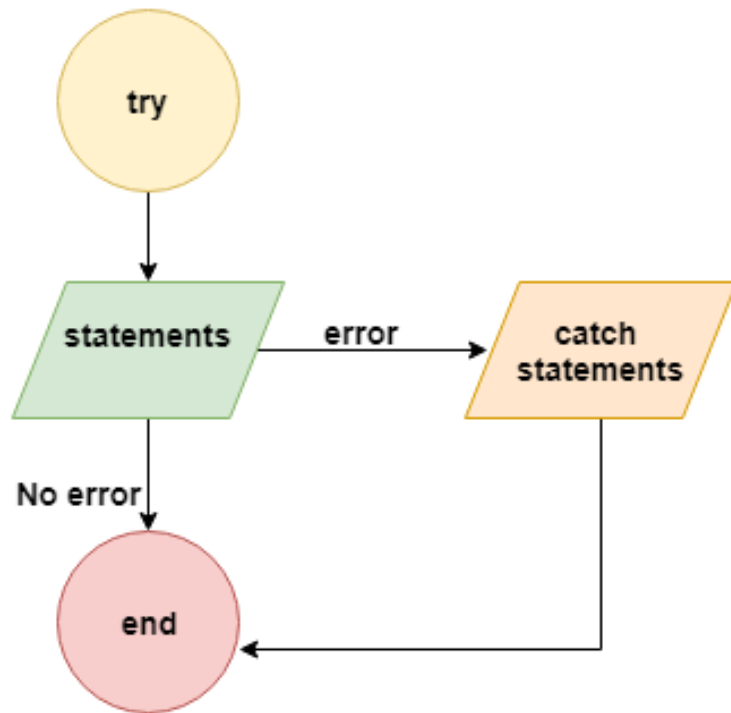


```
% program to print all numbers divisible by 3 and skip remaining  
a = (1:4:50); % creates row vector from 1 to 50 with a step of 4  
for k = 1:numel(a)  
    if rem(a(k),3)  
        continue  
    end  
    disp(a(k))  
end
```

```
9  
21  
33  
45
```

Try to program to find numbers
between 1 and 10000 which are
divisible by all numbers from 2 to 9

Managing errors



```
a = ones(4);  
b = zeros(3);  
try  
    c = [a;b];  
catch ME  
    disp(ME)  
end
```

Whenever any error or exception occurs in the try block, the MATLAB constructs an instance of the **MException** class and returns the object in the catch statement. The MException class object can be accessed with the variable **ME**. The MException class object has five properties-**identifier**, **message**, **stack**, **cause**, and **Correction**. These properties describe details about the occurred exception. We can't use multiple catch block, only one catch block within a try block is allowed, but we can use nested try/catch blocks if required.

MException with properties:

```
identifier: 'MATLAB:catenate:dimensionMismatch'  
message: 'Dimensions of arrays being concatenated are not consistent.'  
cause: {0x1 cell}  
stack: [3x1 struct]  
Correction: []
```

The *end* of this part

As you've probably already noticed, the *end* keyword in MATLAB serves two main purposes:

1. It terminates a block of code.
2. It indicates the last array index.

```
a = ones(4)
for k = 1:length(a)
    if a(k) == 1
        a(k) = 0;
    end
end
disp('.....')
disp(a)
disp('...end')
```

```
a = 4x4
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
.....
    0    1    1    1
    0    1    1    1
    0    1    1    1
    0    1    1    1
...end
```

```
a = randi(100,4,4)
b = a(end,2:end) % here first end argument indicates the last row,
%and the second indicates the columns number from 2 to the last
a = 4x4
    43    66    68    66
    92     4    76    18
    80    85    75    71
    96    94    40     4
b = 1x3
    94    40     4
```

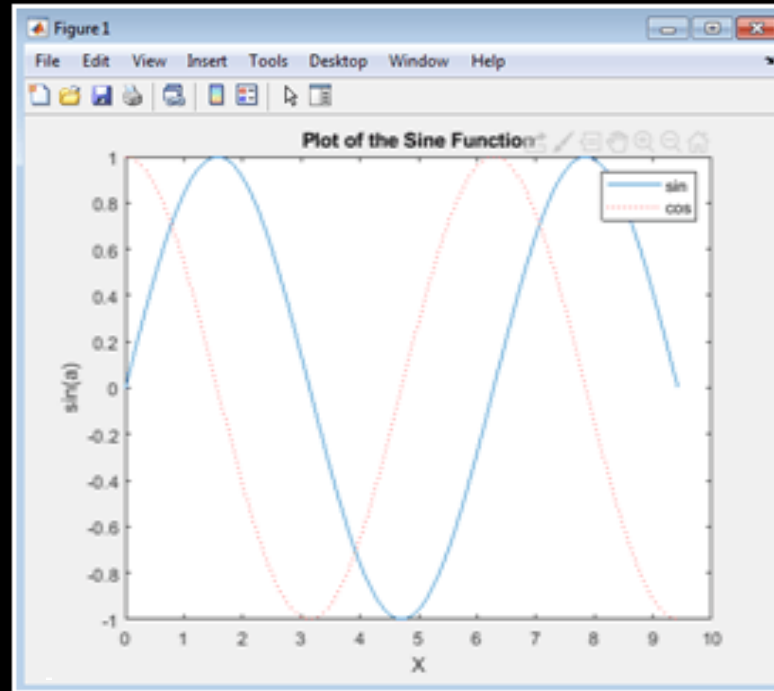

Figures: plot()

Come altrimenti potremmo definire il vettore a ?

```
>> a = linspace(0,3*pi,301)
```

```
>> a = 0:pi/100:3*pi;  
>> b = sin(a);  
>> plot(a,b)
```

The output of this plot() function will output in a new window, where we can edit input values to manipulate the plot.



```
>> c = cos(a);  
>> hold on  
>> plot(a,c,'r:');  
>> legend('sin','cos')
```

- xlabel() - labels the x axis.
- ylabel() - labels the y axis.
- title() - gives a title to the plot

```
>> xlabel('X')  
>> ylabel('sin(X)')  
>> title('Plot of the Sine Function')
```

- A new variable 'c' that holds the result of **cos(a)**.
- **hold on** function to hold the current plot to edit.
- A new plot line of a & c.
- The third argument in plot function, i.e., 'r:' describes the color of the new plotline (r for red color) & ':' for the dotted line.
- Legend function takes the argument, indicating plot lines with their name.

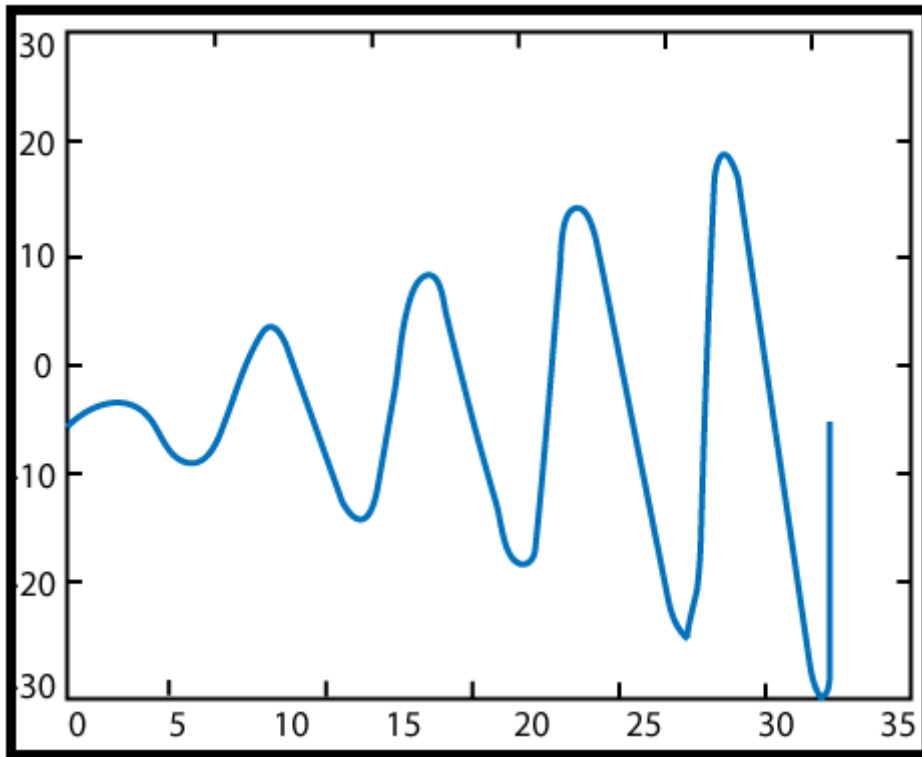
Figures: options

Color Style-option	Line Style option	Marker Style-option
y yellow	- Solid	+ plus sign
m magenta	-- dashed	o circle
c cyan	: dotted	* asterisk
r red	-. dash-dot	x x-mark
g green	none no line	. point
b blue		^ up triangle
w white		s square
k black		d diamond, etc.

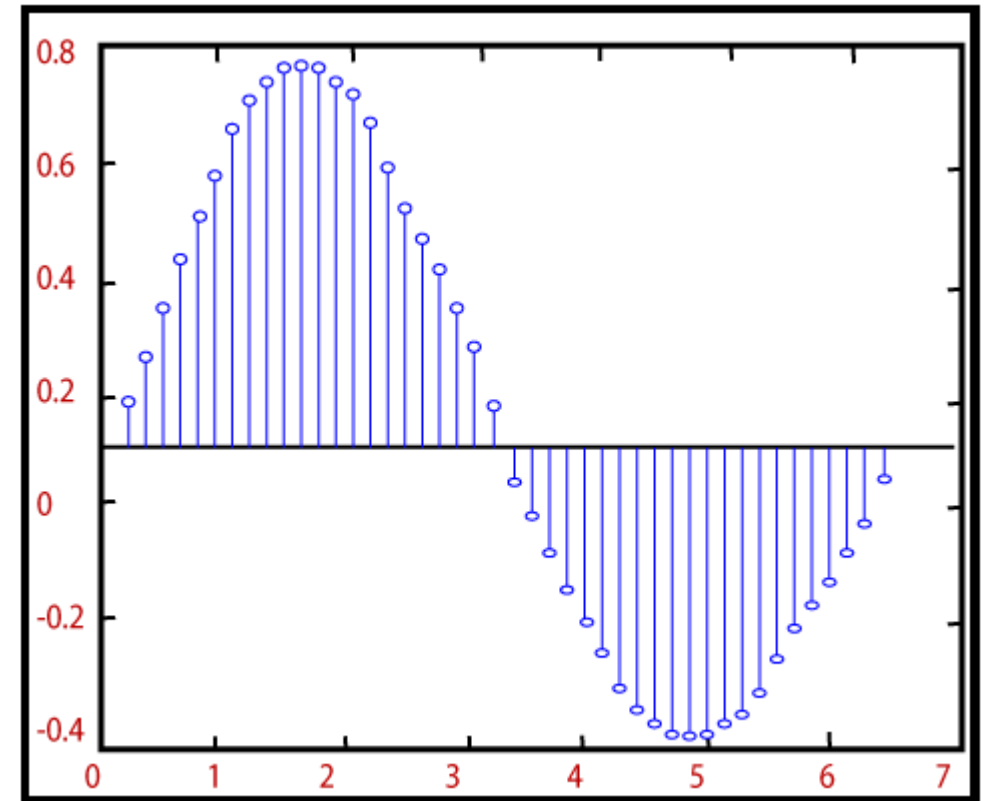
function	description
title	title of the graph
xlabel, ylabel, zlabel	label axes
grid on, grid off	turns grid on / off
hold on	enables to add another graphical elements while keeping the existing ones
xlim, ylim, zlim	set axes' range
legend	display legend
subplot	create more axes in one figure
yyaxis	create chart with two y-axes
box on	display axes outline
text	adds text to graph
and others	

Figures

```
f(t)=t sin t,  $0 \leq t \leq 10\pi$   
fplot('x.*sin(x)',[0 10*pi])
```

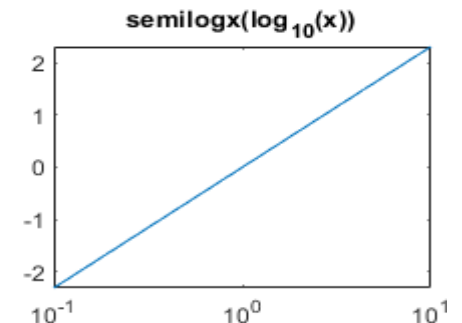
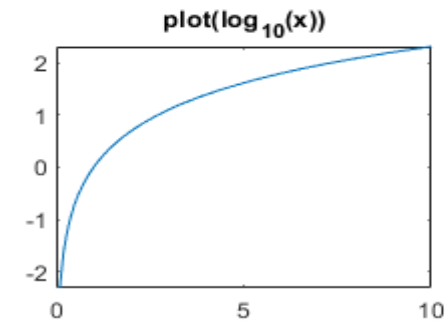
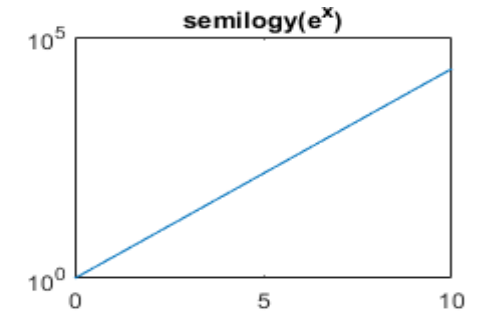
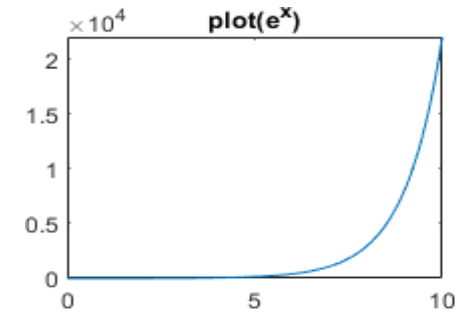


```
f=e-t/5 sin t,  $0 \leq t \leq 2\pi$   
t=linspace(0, 2*pi, 200);  
f=exp(-.2*t).*sin(t);  
stem(t, f)
```



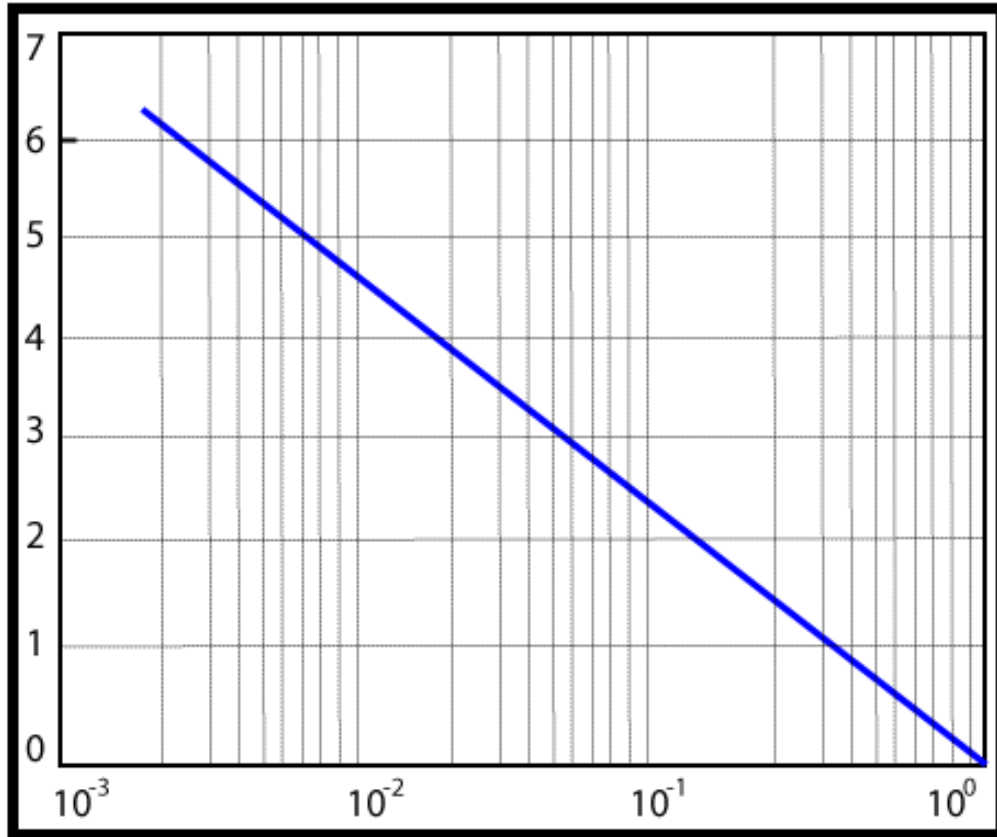
Figures

```
x = 0:0.1:10;  
y1 = exp(x); y2 =  
log(x);  
  
figure('color', 'w');  
subplot(2, 2, 1); plot(x, y1);  
title('plot(e^x)');  
  
subplot(2, 2, 2); semilogy(x, y1);  
title('semilogy(e^x)');  
  
subplot(2, 2, 3); plot(x, y2);  
title('plot(log_1_0(x))');  
  
subplot(2, 2, 4); semilogx(x, y2);  
title('semilogx(log_1_0(x))');
```



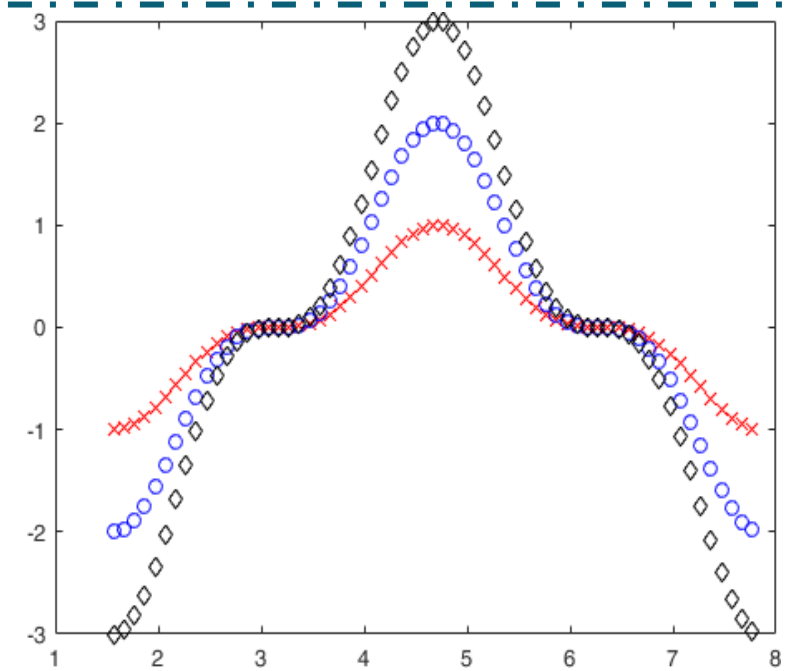
Figures

Che grafico è questo?



```
>> y = linspace(0,2*pi,200);  
>> semilogx(exp(-y),y); grid;
```

Date le x ed fx
seguenti, come si
potrebbe tracciare
questo grafico?



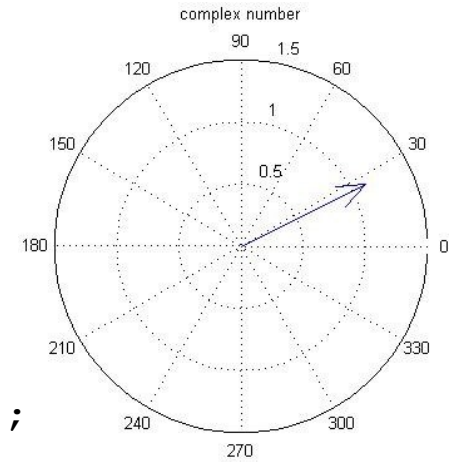
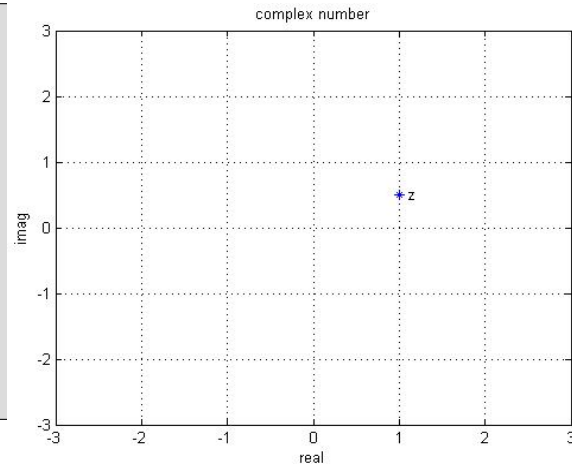
```
>> x = (0:0.1:2*pi) + pi/2;  
>> fx = -[1 2 3].'*sin(x).^3;
```

```
figure;  
plot(x, fx(1, :), 'xr');  
hold on;  
plot(x, fx(2, :), 'ob');  
hold on;  
plot(x, fx(3, :), 'dk');
```

Figures: plotting complex data

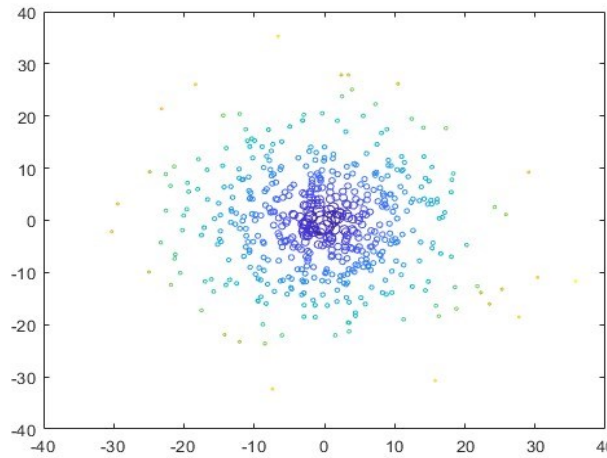
```
z = 1 + .5j;
```

```
plot(z, 'b*'),  
axis([-3 3 -3 3], grid on,  
  
text(real(z)+.1, imag(z), 'z'),  
xlabel('real'), ylabel('imag');  
title('complex number');
```



```
z = 1 + .5j;  
compass(z),  
title('Complex number')
```

```
x = 10*randn(500, 1);  
y = 10*randn(500, 1);  
c = hypot(x, y);  
  
figure('color', 'w');  
scatter(x, y, 100./c);  
box on;
```



```
z = 1 + .5j;  
teta = angle(z);  
r = abs(z);  
polar(teta, r, 'b*'),  
title('Complex number')
```

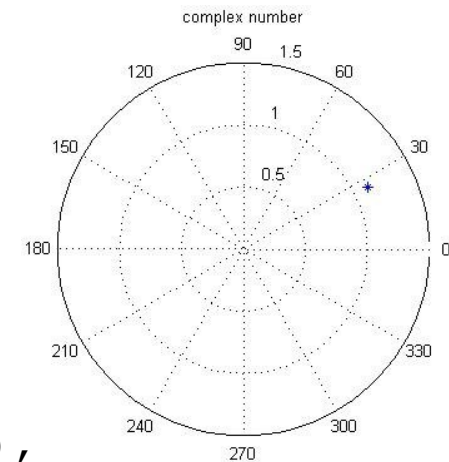
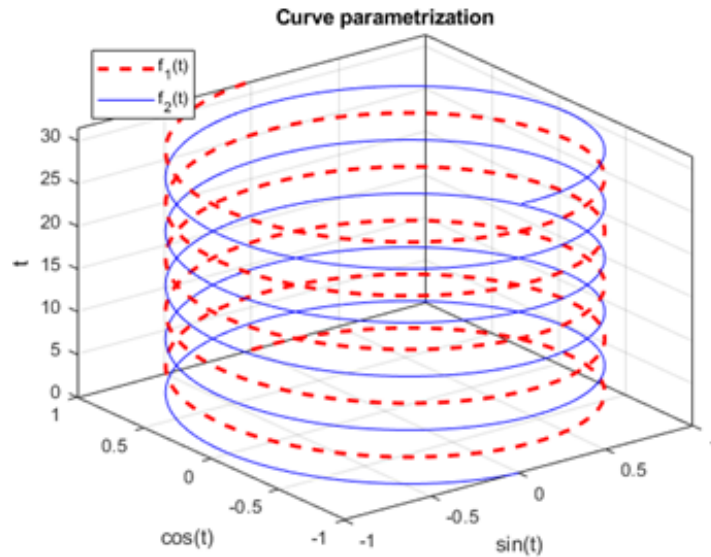


Figure: plot3



```
figure('color', 'w');  
t = 0:0.05:10*pi;  
plot3(sin(t), cos(t), t, 'r--', ...  
      'LineWidth', 2);  
hold on;  
plot3(-sin(t), -cos(t), t, 'b')  
box on;  
grid on;  
xlabel('sin(t)');  
ylabel('cos(t)');  
zlabel('t');  
title('Curve parametrization');  
legend('f_1(t)', 'f_2(t)', ...  
      'Location', 'northwest');
```

Figures: meshgrid

```
[X,Y] = meshgrid(x,y)
```

Dati i vettori x e y , di lunghezza n , la funzione Matlab *meshgrid* costruisce 2 matrici, X e Y di dimensioni $n \times n$, così fatte:

$$X = \begin{bmatrix} x_1 & x_2 & \dots & \dots & x_n \\ x_1 & x_2 & \dots & \dots & x_n \\ x_1 & x_2 & \dots & \dots & x_n \\ \dots & \dots & \dots & \dots & \dots \\ x_1 & x_2 & \dots & \dots & x_n \end{bmatrix} \quad Y = \begin{bmatrix} y_1 & y_1 & \dots & \dots & y_1 \\ y_2 & y_2 & \dots & \dots & y_2 \\ y_3 & y_3 & \dots & \dots & y_3 \\ \dots & \dots & \dots & \dots & \dots \\ y_n & y_n & \dots & \dots & y_n \end{bmatrix}$$

```
x = [-2:0.1:2]; % plot 3D di dati matriciali  
y = [-5:0.2:5]; % N.B. i vettori x e y hanno le stesse dimensioni  
[X,Y] = meshgrid(x,y); % per poter valutare tutte le combinazioni di valori x e y  
Z = X.*exp(-X.^2-Y.^2);  
plot3(X,Y,Z), grid on, title('grafico 3D con 'plot3''), xlabel('X'), ylabel('Y'), zlabel('Z');
```

grafico 3D con 'plot3'

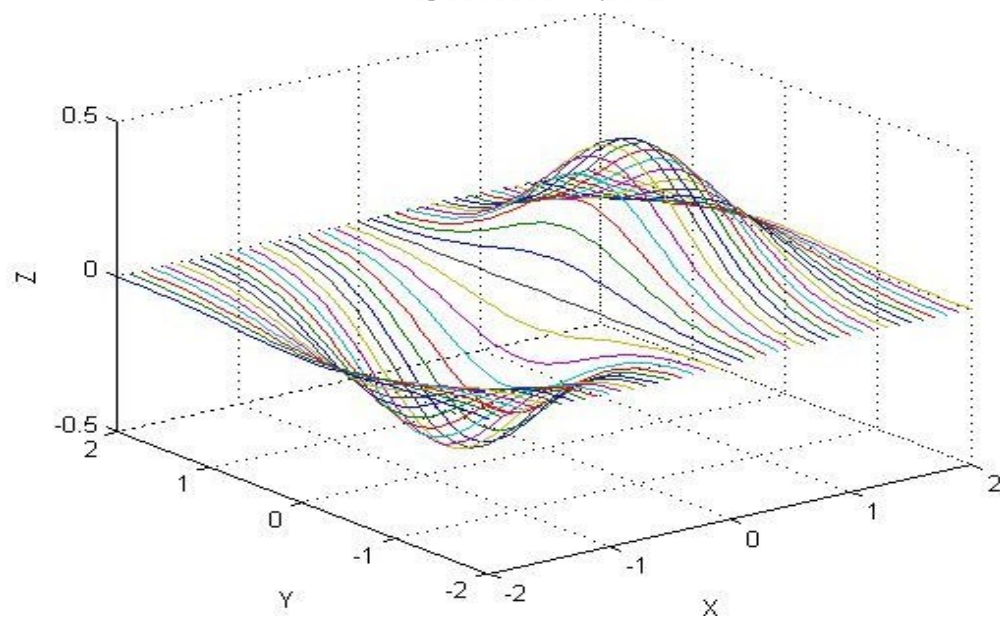


Figure: altri grafici 3D

`mesh(X,Y,Z)` (oppure: `mesh(Z)`) % genera una superficie con griglia colorata

`surf(X,Y,Z)` (oppure: `surf(Z)`) % genera una superficie colorata

`colormap` per modificare la mappa dei colori per (vedi “help colormap”)

`colorbar` mostrare la mappa dei colori (vedi “help colorbar”)

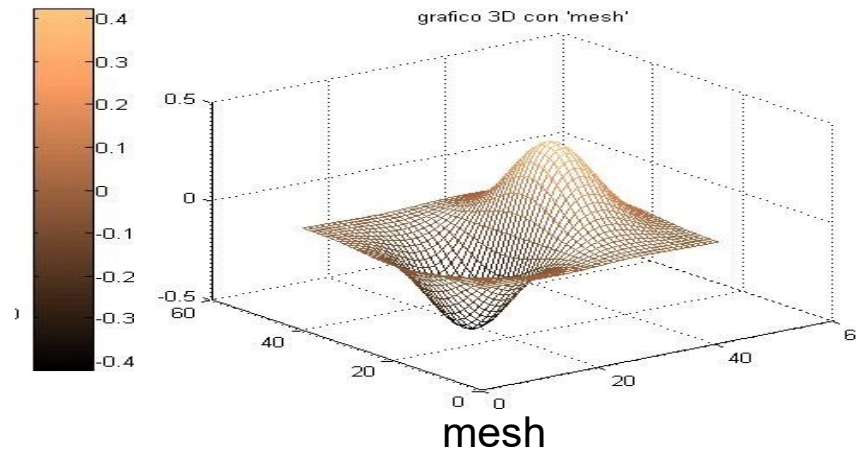
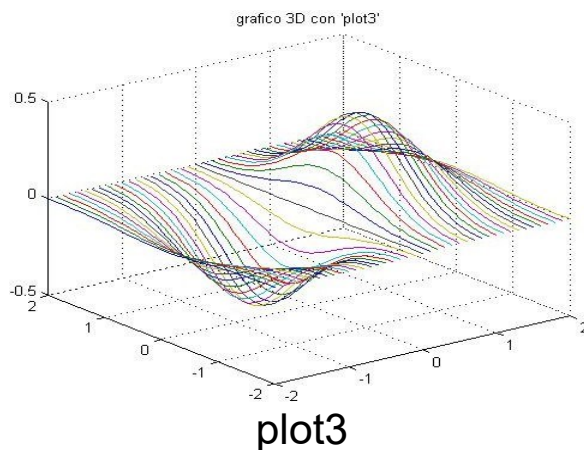
% plot 3D di dati matriciali

```
[X,Y] = meshgrid([-2:0.1:2]); Z = X.*exp(-X.^2-Y.^2);
```

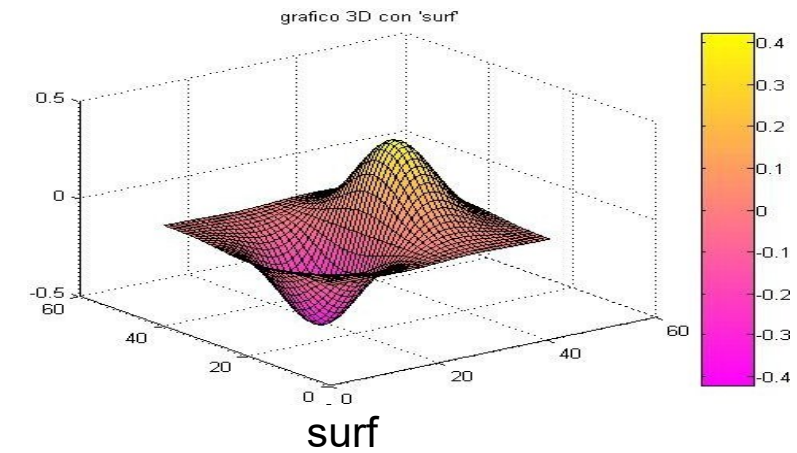
```
plot3(X,Y,Z), grid on, title('grafico 3D con 'plot3'');
```

```
figure,mesh(Z), grid on, title('grafico 3D con 'mesh''); colormap('copper'), colorbar;
```

```
figure,surf(Z), grid on, title('grafico 3D con 'surf''); colormap('spring'), colorbar;
```



Colormap 'copper'



Colormap 'spring'

Figures: mesh and subplots

- ✓ Create a vector

```
>> x = 0:pi/5:3*pi;
```

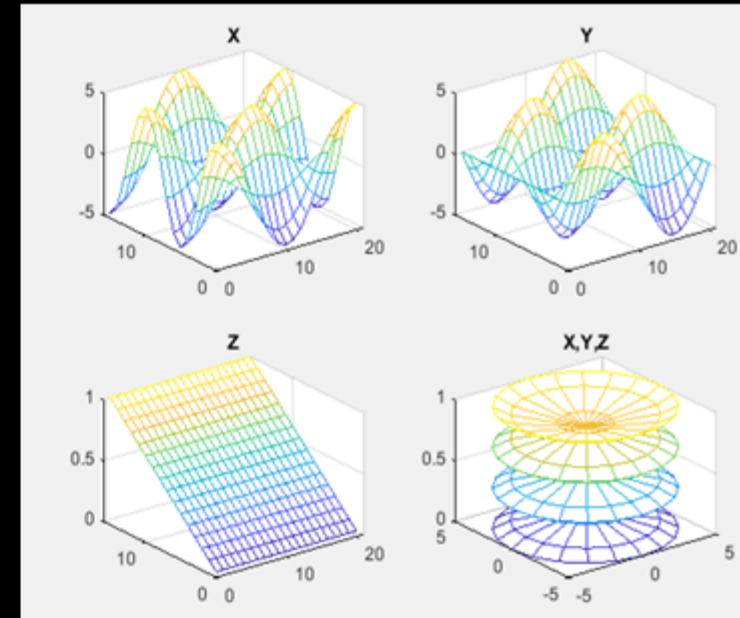
- ✓ create a set of matrices using **cylinder** function that returns coordinates for cylinder unit.

```
>> [X,Y,Z] = cylinder(5*cos(x));
```

- ✓ Now enter **subplot** command by defining the position of each subplot in the same region using matrix indexing.

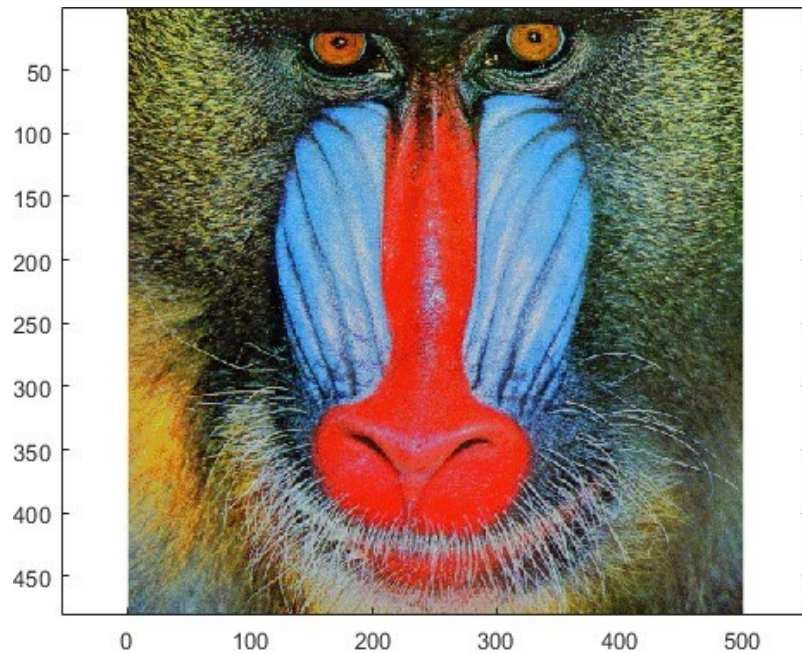
```
>> subplot(2,2,1); mesh(X); title('X');  
>> subplot(2,2,2); mesh(Y); title('Y');  
>> subplot(2,2,3); mesh(Z); title('Z');  
>> subplot(2,2,4); mesh(X,Y,Z); title('X,Y,Z');
```

- ✓ There are three inputs provided in the **subplot** function.
- ✓ First input defines row, second input defines column, and the third input defines the index where the subplot to be positioned.
- ✓ Use **mesh** function to draw 3-D wireframe.
- ✓ And a **title** function to give the title to the subplot.

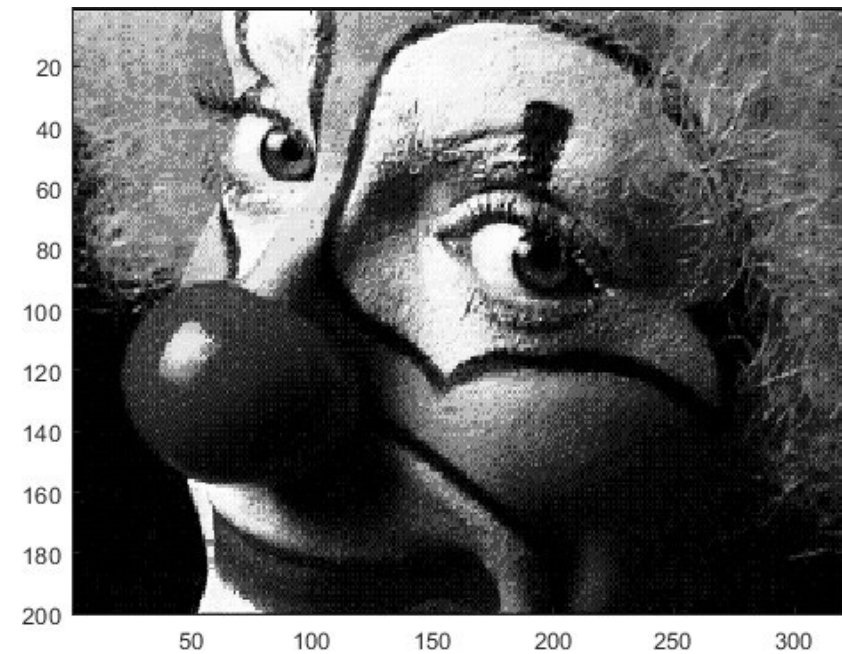


Figures: images

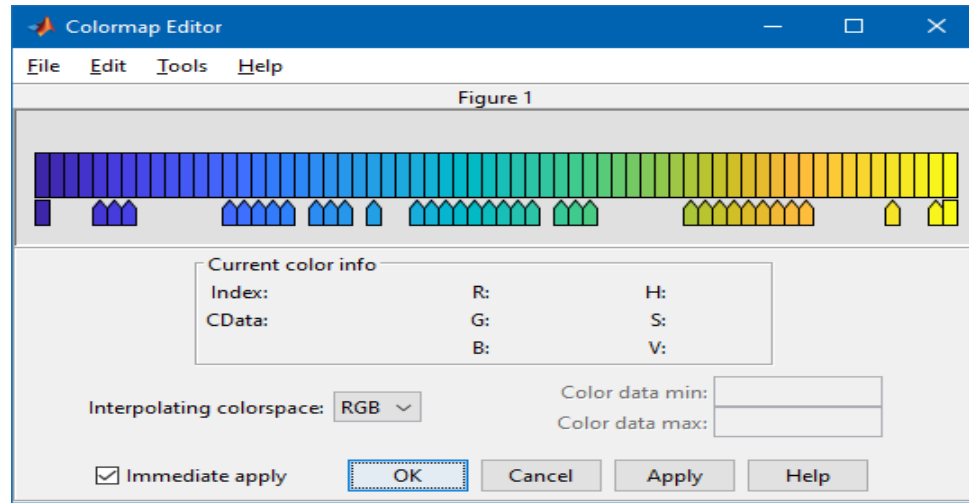
```
load mandrill  
image(X)  
colormap(map)  
axis equal
```



```
load clown  
imagesc(X)  
colormap(gray)
```



Figures: colormap



Colormap Name	Color Scale
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
gray	
bone	
copper	
pink	
lines	
colorcube	
prism	
flag	
white	

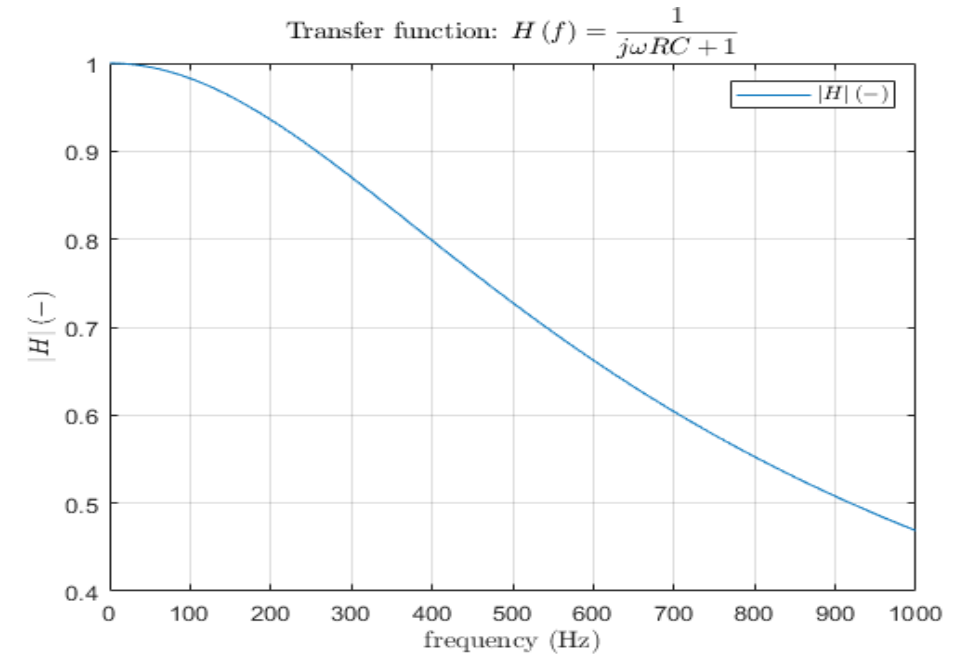
Figures: Interpreter

Labels and titles in figure have 'Interpreter' property.

Possible values are 'tex', 'latex' and 'none'.

Font is default L^AT_EX font.

```
figure;
f = 1:1e3; R = 100; C = 3e-6;
Hf = abs(1./(1j*2*pi*f*R*C + 1)); plot(f, Hf);
grid on;
xlabel('frequency (Hz)', 'Interpreter', 'latex');
ylabel('$$\left| H \right|\left( - \right)$$', ...
'Interpreter', 'latex');
title(['Transfer function: $$H\left( f \right) = \frac{1}{j\omega RC + 1}$$'], ...
'Interpreter', 'latex');
hL = legend('$$\left| H \right|\left( - \right)$$');
hL.Interpreter = 'latex';
```



Figures: Handles

Each individual graphical object has its own pointer ('handle' in Matlab terms).

these handles are practically a reference to an existing object.

Handle is always created by MaTLAB, it is up to the user to store it.

One handle can be saved to several variables but they refer to a single object.

All graphical objects inherit superclass handle.

Inherits several useful methods (set, get, delete, isvalid, ...).

```
>> hFig = figure;  
>> hAx = axes('Parent', hFig); hLine1 = line('Parent', hAx);
```

Graphical objects respect specific hierarchy.

See help for list of properties (>> doc [Figure Properties](#), >> doc [Axes Properties](#), >> doc [Line Properties](#), ...)

The way of setting handle object properties: using functions `set` and `get` (not sensitive). case

```
>> myLineObj = plot(1:10); get(myLineObj, 'color')
```

```
>> set(myLineObj, 'color', 'r')
```

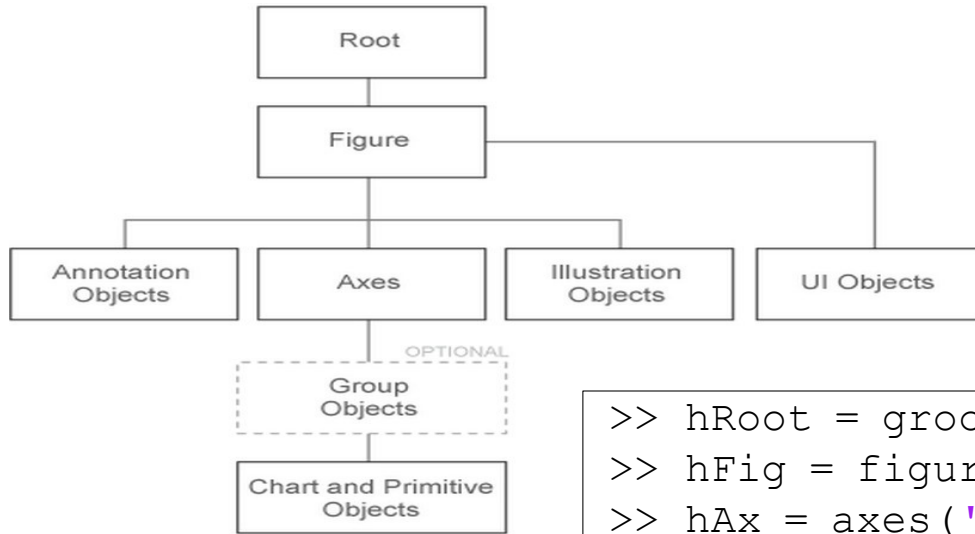
```
>> myLineObj = plot(1:10);  
>> myLineObj.Color
```

```
>> myLineObj.Color = 'r';  
>> myLineObj.Color
```

Dot notation is cAsE sEnSiTiVe.

Figures: object hierarchy

All graphical objects are connected in the hierarchy via Children and Parent properties.



```
>> hRoot = groot;  
>> hFig = figure('Parent', hRoot);  
>> hAx = axes('Parent', hFig);  
>> hLine = line('Parent', hAx, 'XData', -10:10, 'YData', (-10:10).^3);  
>> hTitle = title(hAx, 'Cubic fcn.');
```

```
>> hRoot.Children % ans = hFig  
>> hFig.Children % ans = hAx  
>> hAx.Children % ans = hLine  
>> hLine.Children % ans = 0x0 GraphicsPlaceholder  
>> hTitle.Parent % ans = hAx
```

```
>> hRoot.Children.Children.Color = 'y';
```

Figures: some properties

The `axis` command provides control over the scaling and appearance of both the horizontal and vertical axes of a plot. This command has many features, so only the most useful will be discussed here. For more complete information, refer to on-line help. The primary features are given in the following table

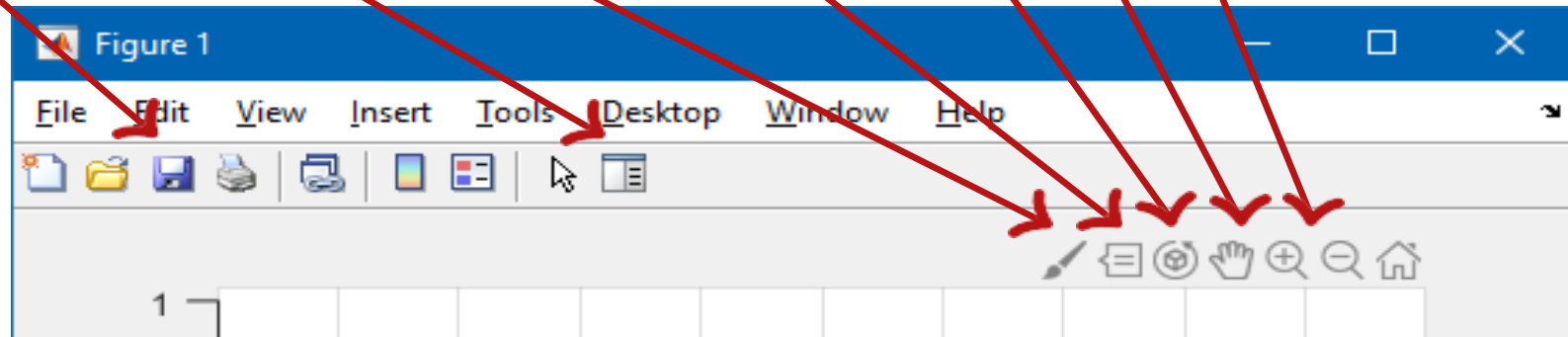
Command	Description
<code>axis([xmin xmax ymin ymax])</code>	Define minimum and maximum values of the axes
<code>axis square</code>	Produce a square plot instead of rectangular
<code>axis equal</code>	Equal scaling factors for both axes
<code>axis normal</code>	Turn off axis square, equal
<code>axis(auto)</code>	Return the axis to automatic defaults
<code>axis off</code>	Turn off axis background, labeling, grid, box, and tick marks. Leave the title and any labels placed by the <code>text</code> and <code>gtext</code> commands
<code>axis on</code>	Turn on axis background, labeling, tick marks, and, if they are enabled, box and grid

Command	Description
<code>grid on</code>	Adds dashed grid lines at the tick marks
<code>grid off</code>	Removes grid lines (default)
<code>grid</code>	Toggles grid status (off to on, or on to off)
<code>box on</code>	Adds axes box, consisting of boundary lines and tick marks on top and right of plot
<code>box off</code>	Removes axes box (default)
<code>box</code>	Toggles box status
<code>title('text')</code>	Labels top of plot with text in quotes
<code>xlabel('text')</code>	Labels horizontal (x) axis with text in quotes
<code>ylabel('text')</code>	Labels vertical (y) axis with text in quotes
<code>text(x,y,'text')</code>	Adds text in quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot
<code>gtext('text')</code>	Place text in quotes with mouse: displays the plot window, puts up a cross-hair to be positioned with the mouse, and write the text onto the plot at the selected position when the left mouse button or any keyboard key is pressed

Figures: 'handmade' properties

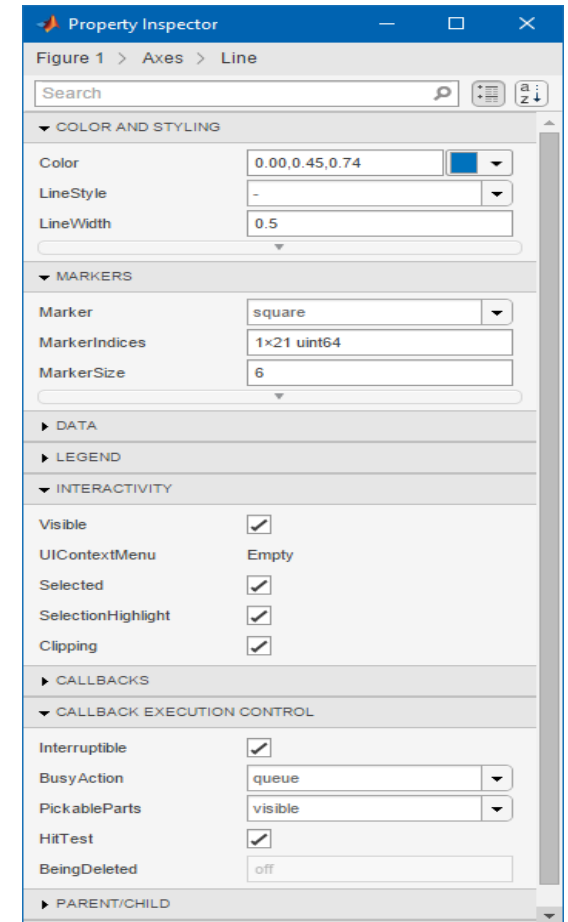
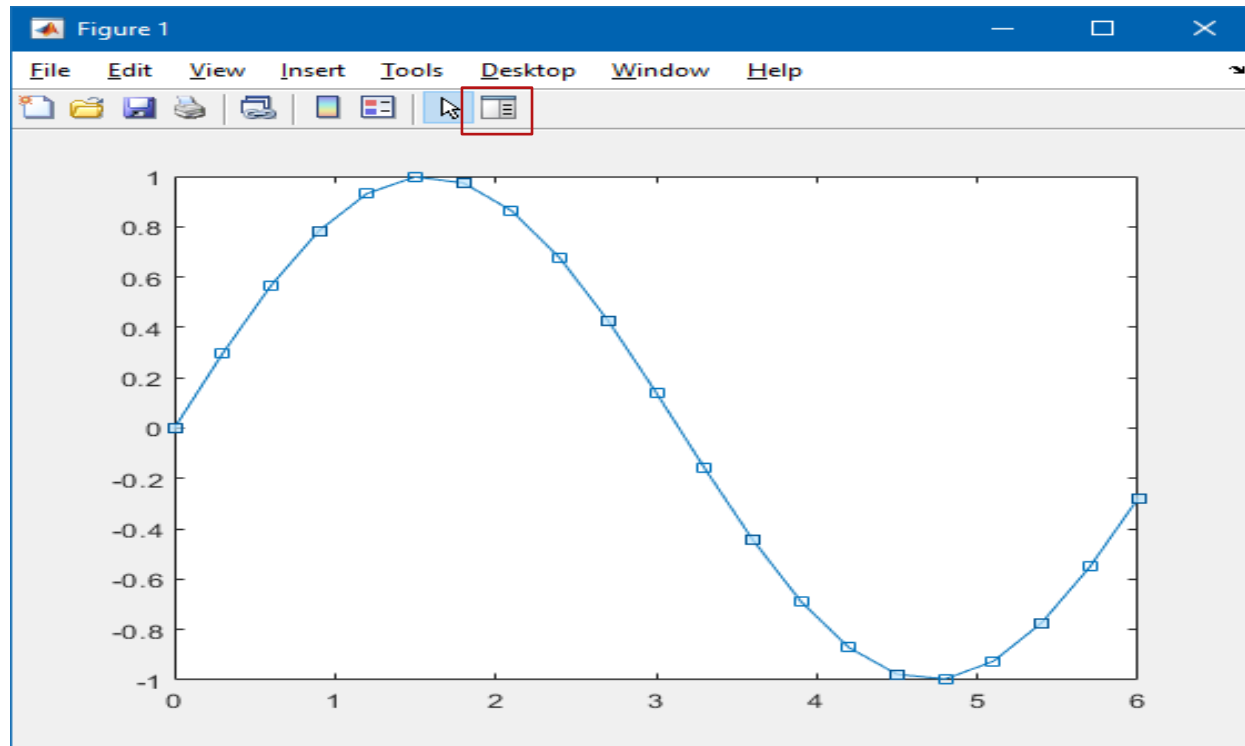
It is possible to keep on editing the graph by other means.
All operations can be carried out using MaTLAB functions.

`saveas`, `inspect`, `brush`, `datacursormode`, `rotate3d`, `pan`, `zoom`

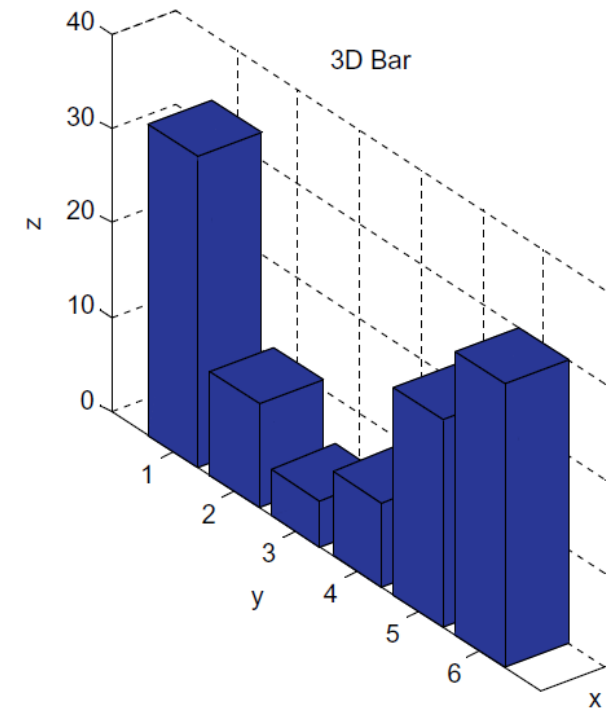
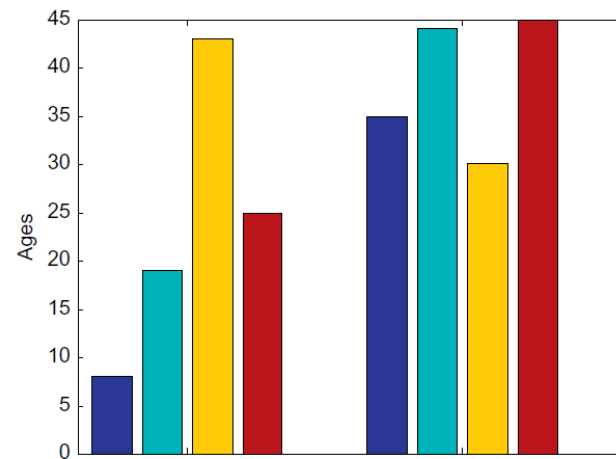
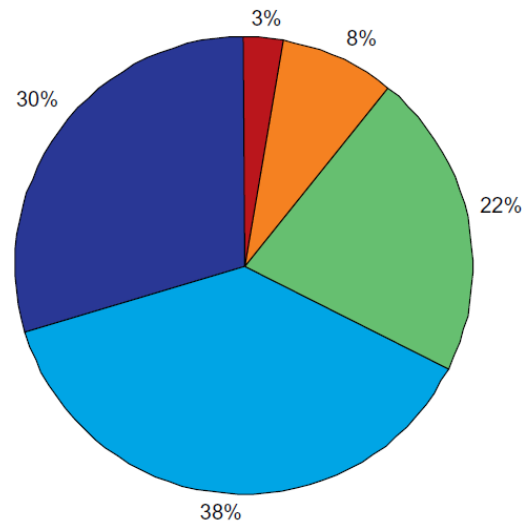


Properties of all graphical objects set programmatically are often preferred for good-looking graphs with lot of graphical features.

Figures: Property inspector (inspect).



Other figures



Other figures

