FIZ 272E
Comp. Meth. in Phys.

MATLAB/GNU Octave Class Notes

Tolga Birkandan
(birkandant@itu.edu.tr)

MATLAB

A Scientific Calculator

&

A Programming Language

MATLAB

- (+): Very easy to learn, widely used in engineering, fast in matrix calculations (MATrix LABoratory)

- (-) : Expensive*, occupies too much hard disc space (~ 5 GB)

- A free MATLAB-like program: **GNU Octave**

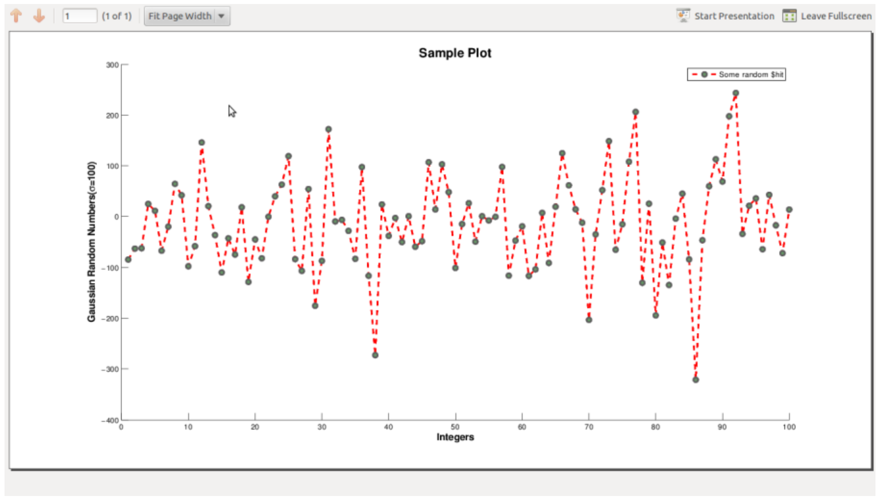
MATLAB can be used for…

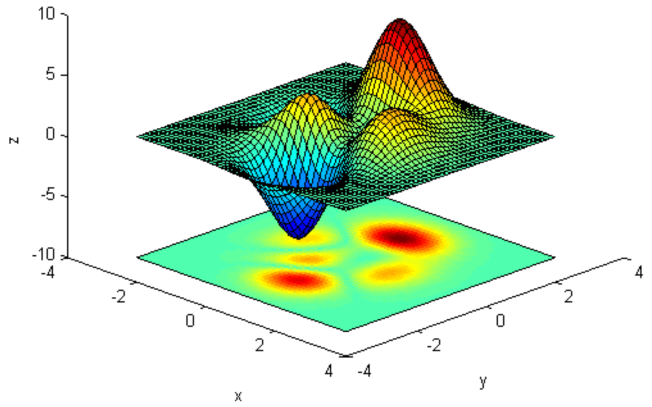
- As a very developed scientific calculator:

```
>> sin(pi/2)

ans =
     1
```

```
>> exp(log(10)) - 10

ans =
   1.7764e-15
```

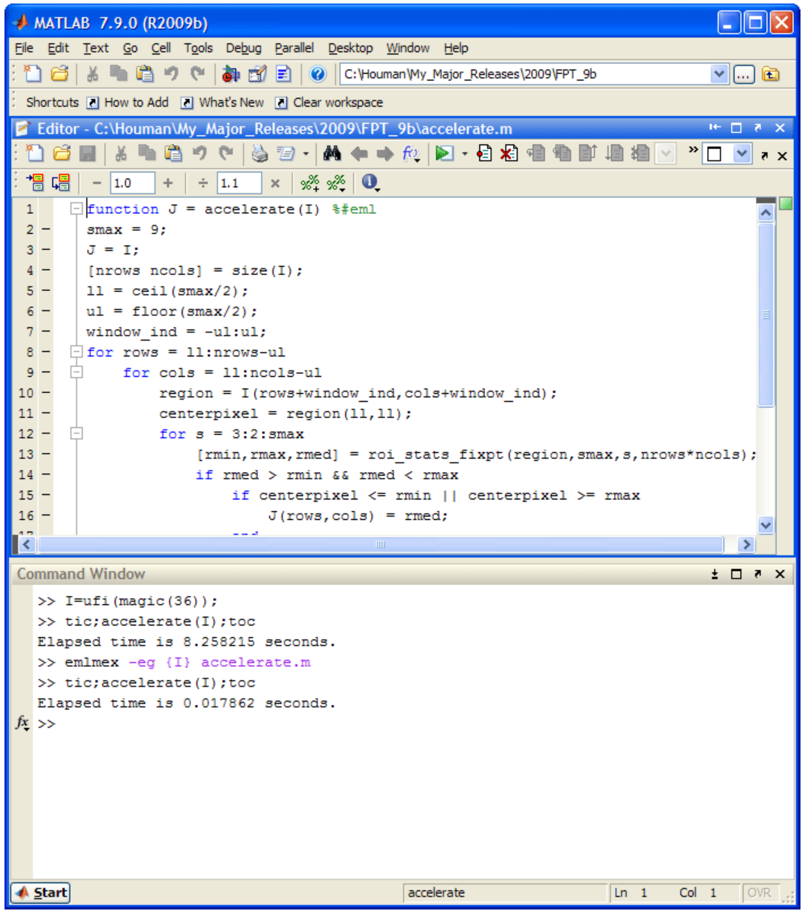
- Plotting graphics:





and…

- As a programming language:

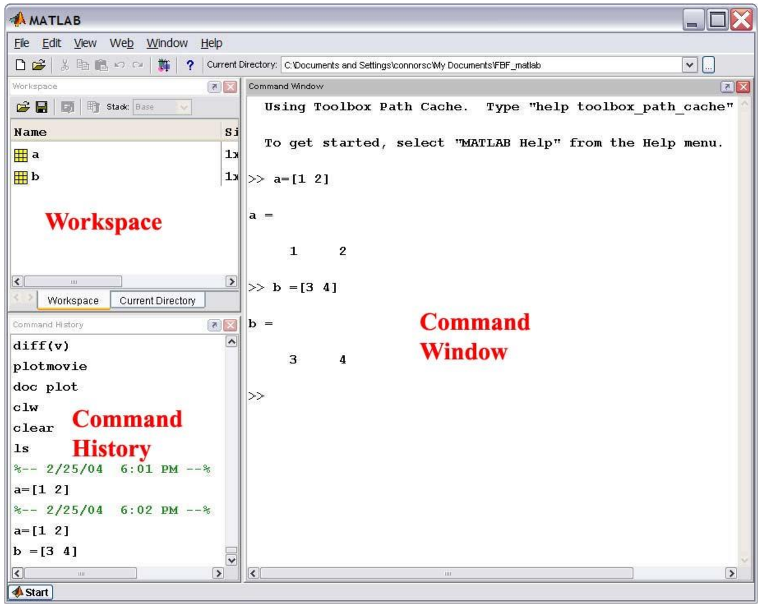

MATLAB 7.9.0 (R2009b)
File Edit Text Go Cell Tools Debug Parallel Desktop Window Help
C:\Houman\My_Major_Releases\2009\FPT_9b
Shortcuts How to Add What's New Clear workspace
Editor - C:\Houman\My_Major_Releases\2009\FPT_9b\accelerate.m
Command Window
Start
accelerate
Ln 1 Col 1
OVR

```matlab
function J = accelerate(I) %#eml
smax = 9;
J = I;
[nrows ncols] = size(I);
ll = ceil(smax/2);
ul = floor(smax/2);
window_ind = -ul:ul;
for rows = ll:nrows-ul
    for cols = ll:ncols-ul
        region = I(rows+window_ind,cols+window_ind);
        centerpixel = region(ll,ll);
        for s = 3:2:smax
            [rmin,rmax,rmed] = roi_stats_fixpt(region,smax,s,nrows*ncols);
            if rmed > rmin && rmed < rmax
                if centerpixel <= rmin || centerpixel >= rmax
                    J(rows,cols) = rmed;
```

```
>> I=ufi(magic(36));
>> tic;accelerate(I);toc
Elapsed time is 8.258215 seconds.
>> emlmex -eg {I} accelerate.m
>> tic;accelerate(I);toc
Elapsed time is 0.017862 seconds.
>>
```

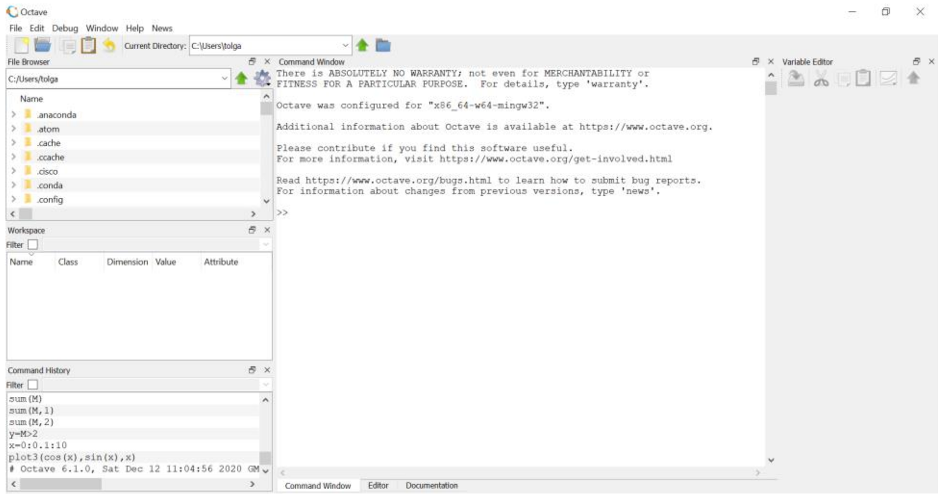
Let’s start!

- Start MATLAB / GNU Octave.
- The first component that we will use is in this screen: **Command window**



```
Using Toolbox Path Cache.  Type "help toolbox_path_cache"

To get started, select "MATLAB Help" from the Help menu.
>> a=[1 2]

a =

     1     2

>> b =[3 4]

b =

     3     4

>>
```

```
diff(v)
plotmovie
doc plot
clw
clear
ls
%-- 2/25/04   6:01 PM --%
a=[1 2]
%-- 2/25/04   6:02 PM --%
a=[1 2]
b =[3 4]
```



```
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>>
```

```
sum(M)
sum(M,1)
sum(M,2)
y=M>2
x=0:0.1:10
plot3(cos(x),sin(x),x)
# Octave 6.1.0, Sat Dec 12 11:04:56 2020 GM
```

Basic commands -1

```
>> 2+2

ans =
     4

>> 4^2

ans =
    16

>> sin(pi/2)

ans =
     1
```

Basic commands -2

```
>> 1/0

 ans =

      Inf


>> exp(i*pi)

ans =
  -1.0000 + 0.0000i


>> Inf/Inf

ans =
    NaN
```

Basic commands -3

```
>> x = sqrt(3)

x =
    1.7321

>> atan(x)

ans =
    1.0472

>> pi/ans

ans =
     3
```

Basic commands -4

```
 >> exp(log(10)) - 10

 ans =
    1.7764e-15
```

```
>> % Anything after a % sign is a comment.
>> x = rand(100,100); % ; means "don't print out result"
>> s = 'Hello world';  % single quotes enclose a string
>> t = 1 + 2 + 3 + ...
4 + 5 + 6               % ... continues a line

t =
    21
```

Arrays and matrices -1

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =
     1     2     3
     4     5     6
     7     8     9

>> b = [0;1;0]

b =
     0
     1
     0
```

```
>> size(A)

ans =
     3     3

>> size(b)

ans =
     3     1
```

Arrays and matrices -2

Arrays can be built out of other arrays, as long as the sizes are compatible.

```
>> [A b]

ans =
     1     2     3     0
     4     5     6     1
     7     8     9     0
```

`eye`	identity matrix
`zeros`	all zeros
`ones`	all ones
`diag`	diagonal matrix (or, extract a diagonal)
`toeplitz`	constant on each diagonal
`triu`	upper triangle
`tril`	lower triangle
`rand`, `randn`	random entries
`linspace`	evenly spaced entries
`cat`	concatenate along a given dimension
`repmat`	duplicate vector across a dimension

Arrays and matrices -3

An especially important array constructor is the **colon** operator.

```
>> 1:8

ans =
     1     2     3     4     5     6     7     8

>> 0:2:10

ans =
     0     2     4     6     8    10

>> 1:-.5:-1

ans =
    1.0000    0.5000         0   -0.5000   -1.0000
```

The format is `first:step:last`. The result is always a row vector, or the empty matrix if `last` < `first`.

Arrays and matrices -4

Referencing elements

```
>> A(2,3)

ans =
     6
```

```
A =
        1         2         3
        4         5         6
        7         8         9
```

```
>> b(2)

ans =
     1
```

```
b =
        0
        1
        0
```

```
>> B(:,3)

ans =
       5
       6
```

```
B =
        1         2         5
        3         4         6
```

Matrix operations -5

```
>> A+A

ans =
     2     4     6
     8    10    12
    14    16    18

>> ans-1

ans =
     1     3     5
     7     9    11
    13    15    17

>> 3*B

ans =
     3     6     9
     9    12    18
```

```
A =
     1     2     3
     4     5     6
     7     8     9
```

```
B =
     1     2     5
     3     4     6
```

Matrix operations -6

```
>> A*b

ans =
     2
     5
     8

>> B*A

ans =
    30    36    42
    61    74    87

>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.
>> A^2

ans =
    30    36    42
    66    81    96
   102   126   150
```

Matrix operations -7

The apostrophe ' produces the complex-conjugate transpose (i. e, the hermitian or adjoint) of a matrix.

```
>> b'*b

ans =
       1
```

A special operator, \ (backslash), is used to solve linear systems of equations.

Matrix operations -8

$$\begin{aligned}1x_1 + 3x_2 + 5x_3 &= 3\\ 2x_1 + 1x_2 + 1x_3 &= 2\\ 4x_1 + 3x_2 + 6x_3 &= 1\end{aligned}$$

```
>> M =[ 1 3 5;
                2 1 1;
                4 3 6]
M =
     1     3     5
     2     1     1
     4     3     6
>>              b =[3;
                     2;
                     1 ]
b =


     3
     2
     1
>>              x =M\b
x =
   -0.0909
    3.9091
   -1.7273
```

Matrix operations -9

```
>> A.*C

ans =
     1     6    -3
     8    20     0
    42     0     9

>> A*C

ans =
    23    11     2
    50    32     2
    77    53     2
```

```
A =
     1     2     3
     4     5     6
     7     8     9
```

```
C =
     1     3    -1
     2     4     0
     6     0     1
```

Matrix operations -10

```
>> C

C =
     1     3    -1
     2     4     0
     6     0     1

>> sum(C,1)

ans =
     9     7     0

>> sum(C,2)

ans =
     3
     6
     7
```

Matrix operations -11

```
>>          y =linspace(0, 0.5, 4)
y =
     0    0.1667    0.3333    0.5000
```

```
>>          x = [ 1  2  3  4  5];
>>          y  = exp(x)
y =
   2.7183    7.3891   20.0855   54.5982  148.4132
```

```
>>          x = [ 1 2 3 4 5];
>>          x >3
ans =
     0     0     0     1     1
```

```
>>          x = [ 5 6 3 8 7 9];
>>          x(1:3)
ans =
     5     6     3
```

```
>>          x(2:4)
ans =
     6     3     8
```

```
>>          x(3:end)
ans =
3     8     7     9
```

Matrix operations -12

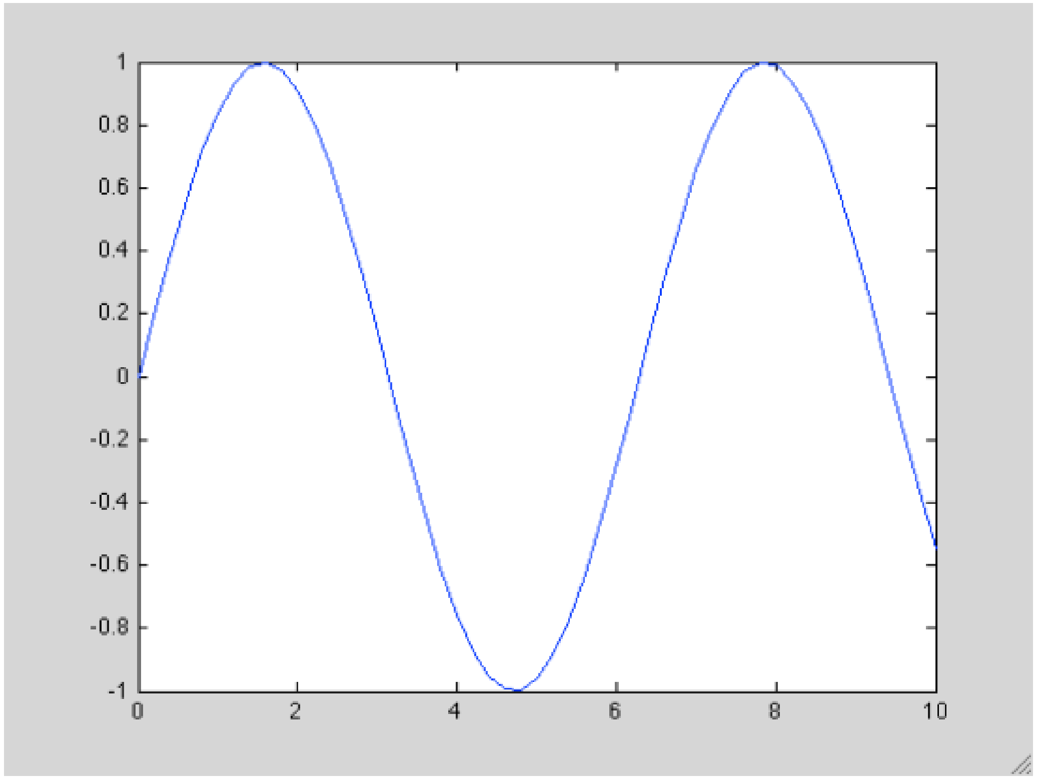
```
>>                 zeros(3,3)
ans =
     0     0     0
     0     0     0
     0     0     0
```

```
>>                 ones(3,3)
ans =
     1     1     1
     1     1     1
     1     1     1
```

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

Plots -1

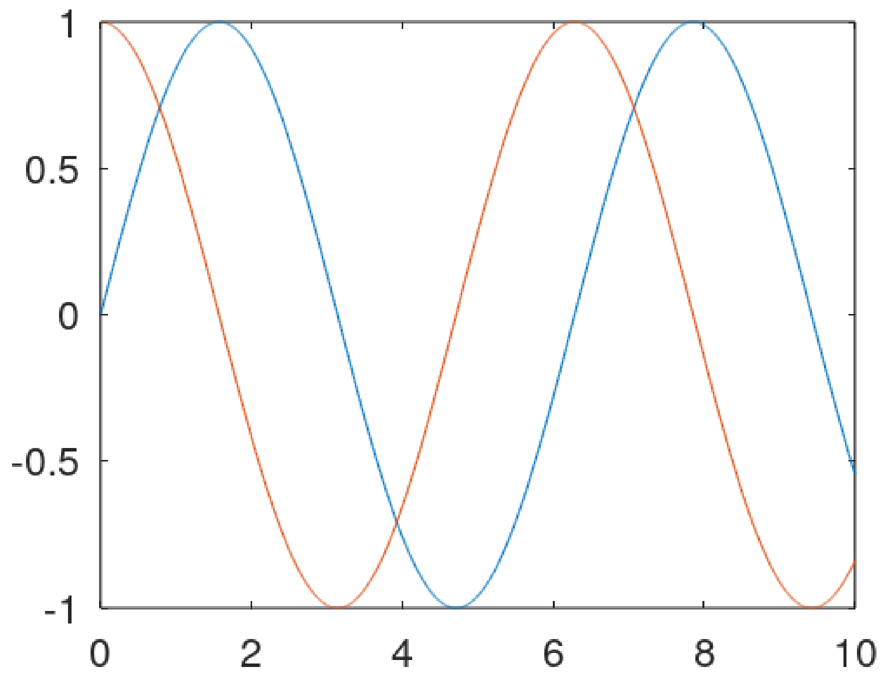
```
>> x=0:0.2:10;
>> y=sin(x);
>> plot(x,y)
```



Plots -2

```
plot(xaxis1,yaxis1,xaxis2,yaxis2,…)
```

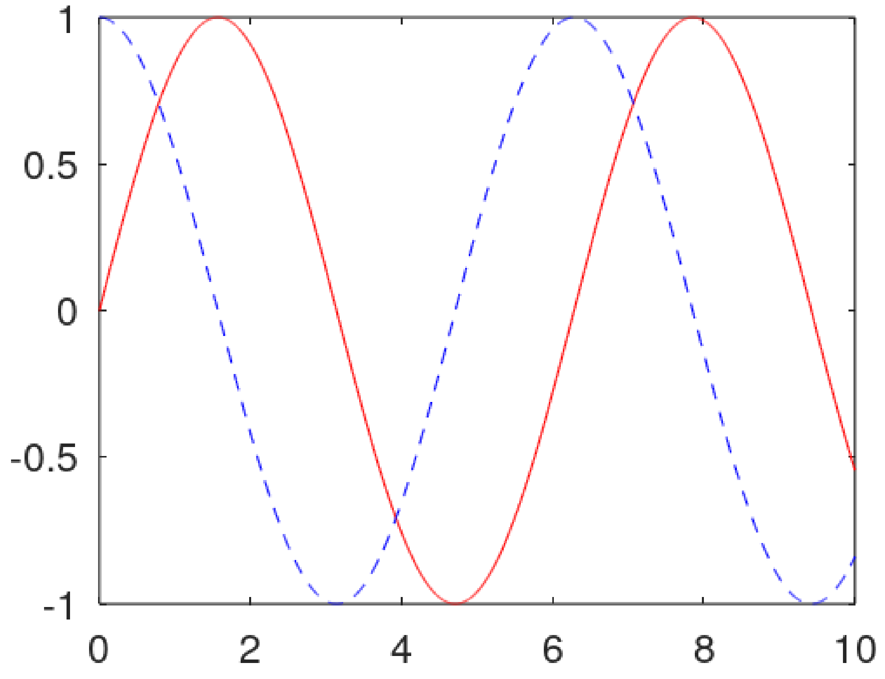
```
x=0:0.01:10;
y=sin(x);
z=cos(x);
plot(x,y,x,z)
```



Plots -2.5

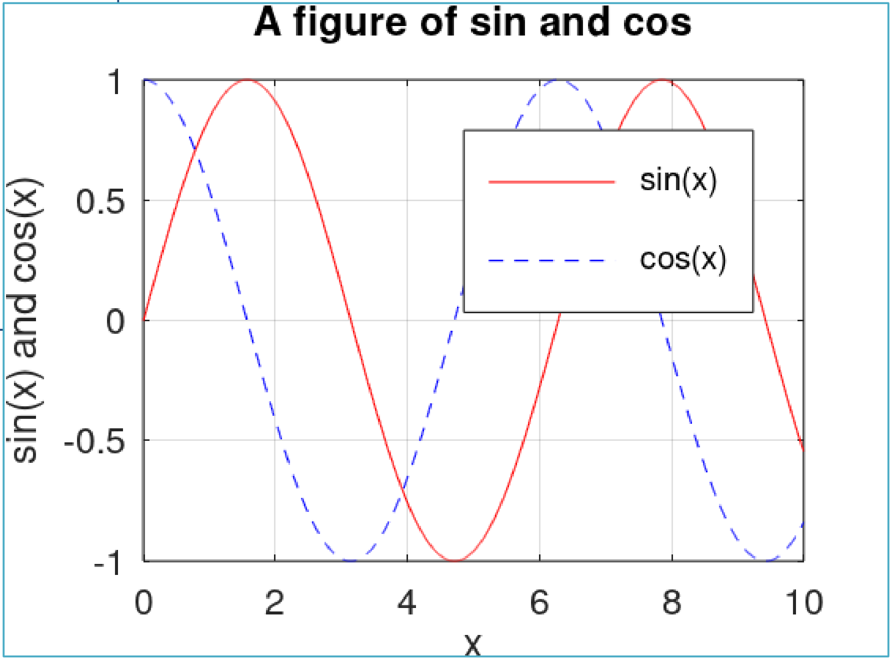
```
plot(xaxis1,yaxis1,xaxis2,yaxis2,...)
```

```
x=0:0.01:10;
y=sin(x);
z=cos(x);
plot(x,y,'-r',x,z,'--b')
```



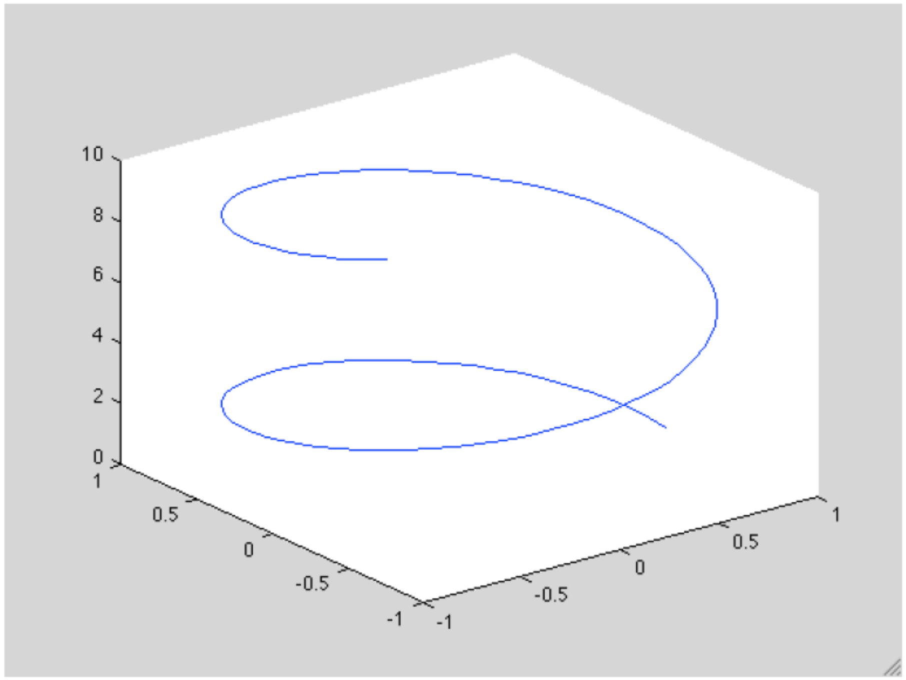
Plots -3

```
x=0:0.01:10;
y=sin(x);
z=cos(x);
plot(x,y,'-r',x,z,'--b')
legend('sin(x)','cos(x)')
title('A figure of sin and cos')
xlabel('x')
ylabel('sin(x) and cos(x)')
grid on
```



Plots -4

```
>> x=0:0.1:10;
>> plot3(cos(x),sin(x),x);
```



Plots -4

Symbol	Color	Symbol	Marker	Symbol	Linestyle
`b`	blue	`.`	point	`-`	solid line
`g`	green	`o`	circle	`:`	dotted line
`r`	red	`x`	cross	`-.`	dash-dot line
`c`	cyan	`+`	plus sign	`--`	dashed line
`m`	magenta	`*`	asterisk		
`y`	yellow	`s`	square		
`k`	black	`v`	triangle (down)		
`w`	white	`^`	triangle (up)		
		`<`	triangle (left)		
		`>`	triangle (right)		
		`p`	pentagram		
		`h`	hexagram		

Programming in MATLAB -1

- MATLAB programs are **not** compiled as in Fortran and C. They are read and run by MATLAB line by line.
- So, MATLAB is an “interpreted” language. Not a “compiled” one.
- What you write in MATLAB is a “script”.
- A MATLAB program contains:

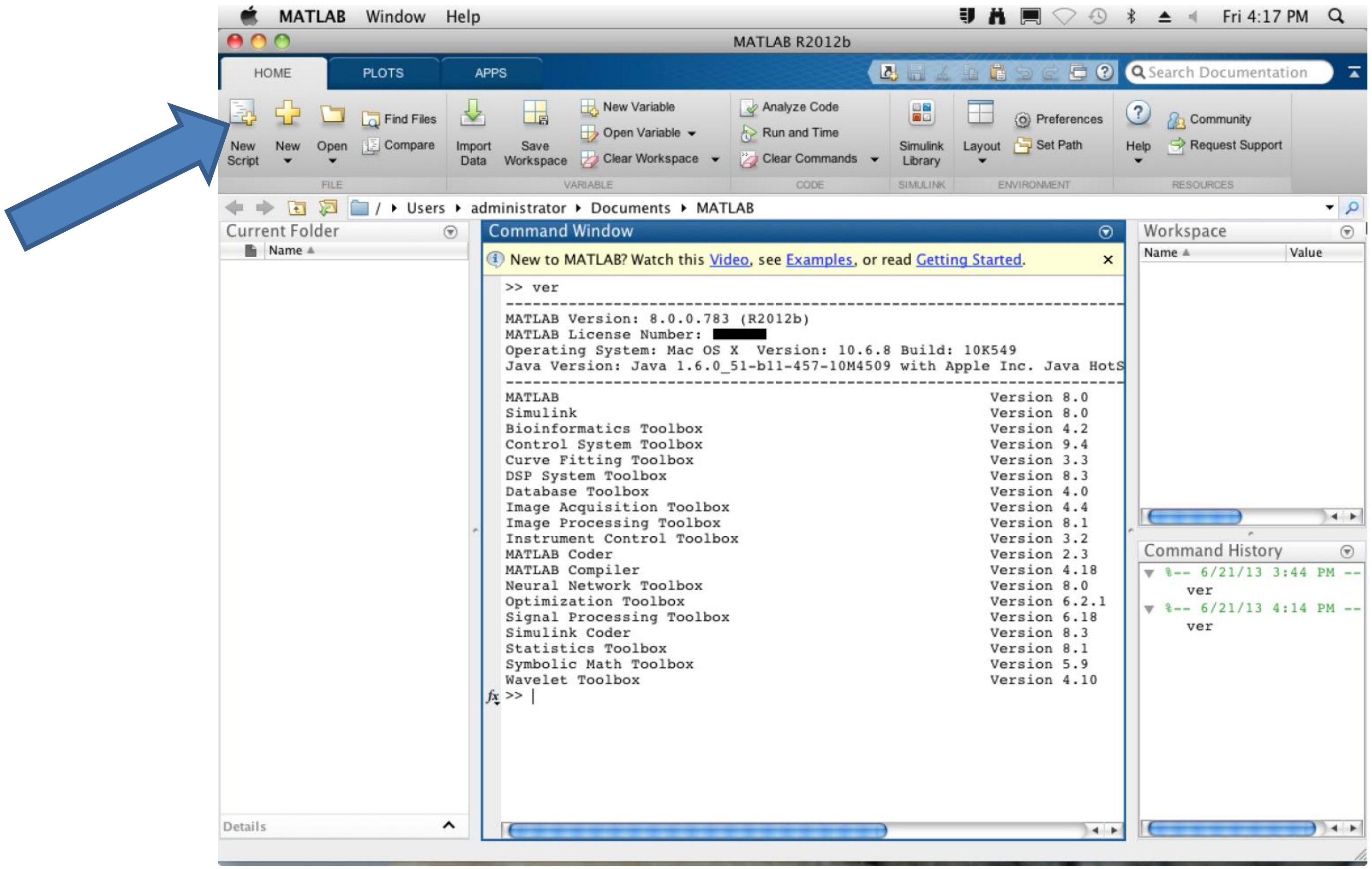
usual MATLAB commands

+coding structures (for, if, while, etc…)

[+coding methods (object oriented programming) if needed]

Programming in MATLAB -2

- We will write our program in another window



```
>> ver
-------------------------------------------------------------------------------------
MATLAB Version: 8.0.0.783 (R2012b)
MATLAB License Number: [redacted]
Operating System: Mac OS X  Version: 10.6.8 Build: 10K549
Java Version: Java 1.6.0_51-b11-457-10M4509 with Apple Inc. Java HotS
-------------------------------------------------------------------------------------
MATLAB                                                Version 8.0
Simulink                                              Version 8.0
Bioinformatics Toolbox                                Version 4.2
Control System Toolbox                                Version 9.4
Curve Fitting Toolbox                                 Version 3.3
DSP System Toolbox                                    Version 8.3
Database Toolbox                                      Version 4.0
Image Acquisition Toolbox                             Version 4.4
Image Processing Toolbox                              Version 8.1
Instrument Control Toolbox                            Version 3.2
MATLAB Coder                                          Version 2.3
MATLAB Compiler                                       Version 4.18
Neural Network Toolbox                                Version 8.0
Optimization Toolbox                                  Version 6.2.1
Signal Processing Toolbox                             Version 6.18
Simulink Coder                                        Version 8.3
Statistics Toolbox                                    Version 8.1
Symbolic Math Toolbox                                 Version 5.9
Wavelet Toolbox                                       Version 4.10
fx >> |
```

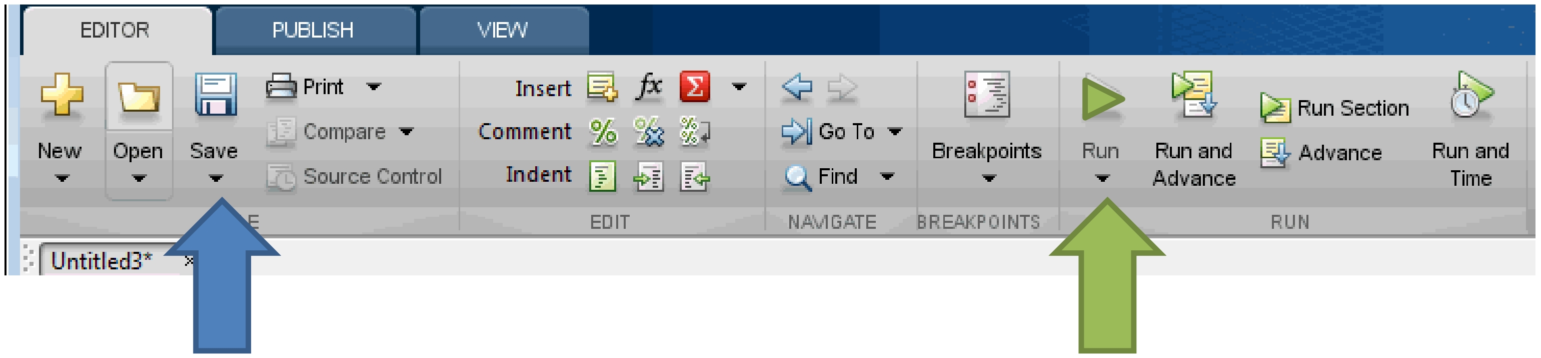
```
%-- 6/21/13 3:44 PM --
   ver
%-- 6/21/13 4:14 PM --
   ver
```

Programming in MATLAB -3

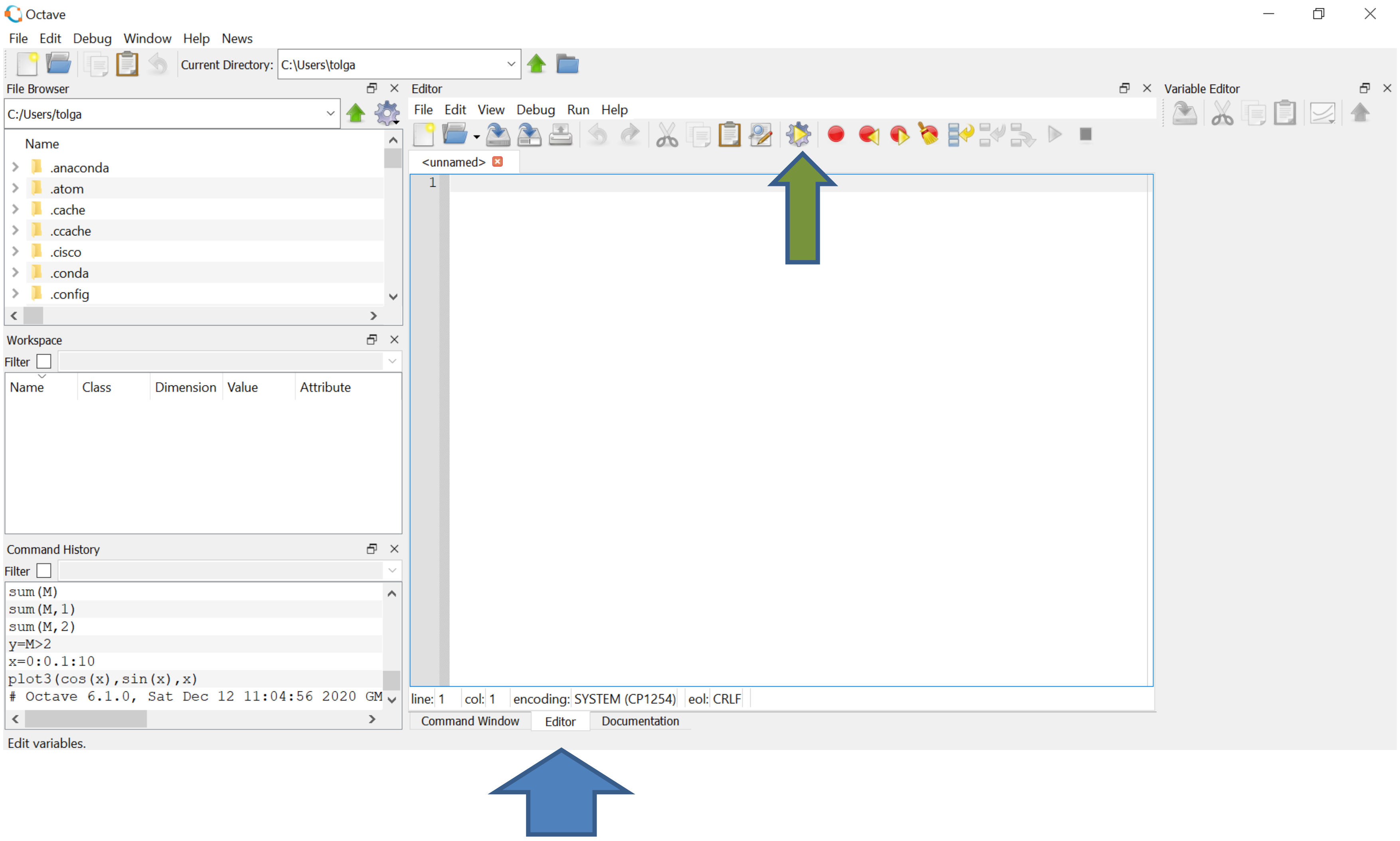
- Write your first program:

```
clear all
clc

t=0:0.001:3*pi;
plot(t,sin(t),t,cos(t))
```

- Save it and run



In GNU Octave


Octave
File Edit Debug Window Help News
Current Directory: C:\Users\tolga
File Browser
C:/Users/tolga
Name
.anaconda
.atom
.cache
.ccache
.cisco
.conda
.config
Workspace
Filter
Name Class Dimension Value Attribute
Command History
Filter
Editor
File Edit View Debug Run Help
<unnamed>
1
line: 1 col: 1 encoding: SYSTEM (CP1254) eol: CRLF
Command Window Editor Documentation
Variable Editor
Edit variables.

```
sum(M)
sum(M,1)
sum(M,2)
y=M>2
x=0:0.1:10
plot3(cos(x),sin(x),x)
# Octave 6.1.0, Sat Dec 12 11:04:56 2020 GM
```

if -1

```
clear all
clc

x=input('Enter a number... ');

if(x>0)
    disp('The number is positive')
end
```

if -2

```
clear all
clc

x=input('Enter a number... ');

if(x>0)
    disp('The number is positive')
 elseif(x<0)
    disp('The number is negative')
 else
    disp('The number is zero')
end
```

if -3

```
clear all
clc

% Division by 6
x=input('Enter a number... ');

if(mod(x,2)==0) && (mod(x,3)==0)
    disp('can be divided by 6 without a remainder')
else
    disp('canNOT be divided by 6 without a remainder')
end
```

if -4

`==`	`equal`
`~=`	`not equal`
`<`	`less than`
`<=`	`less than or equal`
`>`	`greater than`
`>=`	`greater than or equal`
`&`	`and`
`|`	`or`
`~`	`not`

if -5

```
clear all
clc

A=[1,2;3,4];
B=[5 6;7 8];

rowA=size(A,1);
colA=size(A,2);

rowB=size(B,1);
colB=size(B,2);
```

```
if (rowA==rowB) && (colA==colB)
    disp('These matrices can be added.')
    disp('The addition yields')
    C1=A+B;
    disp(C1)
else
   disp('These matrices canNOT be added.')
end

if (colA==rowB)
    disp('These matrices can be multiplied.')
    disp('The multiplication yields')
    C2=A*B;
    disp(C2)
else
   disp('These matrices canNOT be multiplied.')
end
```

for

```
clear all
clc
% Factorial of x

x=input('Enter a number... ');

factorial=1;

for i=1:x
    factorial=i*factorial;
end

disp('The factorial is')
disp(factorial)
```

while -1

```
clear all
clc
% Factorial of x

x=input('Enter a number... ');

factorial=1;
i=1;

while i<=x
      factorial=i*factorial;
      i=i+1;
end

disp('The factorial is')
disp(factorial)
```

while -2

```
clear all
clc
% Factorial of x

x=input('Enter a number... ');

factorial=1;
i=1;

while 1
      factorial=i*factorial;
      i=i+1;
      if i>x
        break
      end
end

disp('The factorial is')
disp(factorial)
```

Functions -1

```
function [out1, out2, ...] = myfun(in1, in2, ...)
.
.
.
end
```

```
function [addresult,multresult]=addmultiply(a,b)
 addresult=a+b;
 multresult=a*b;
end
```

Main program and functions are all **separate** files on the disk.

Do not forget to **save** your function before calling it.

Functions -2

```
function [myaverage,mymax,mymin]=gradeanalysis(mygrades)

myaverage=sum(mygrades)/length(mygrades);
mymax=max(mygrades);
mymin=min(mygrades);

end
```

```
clear all
clc

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

[average,mymax,mymin]=gradeanalysis(grades);

disp('The average is')
disp(average)
disp('The minimum is')
disp(mymin)
disp('The maximum is')
disp(mymax)
```

File Input/Output -1

```
>> v=[1,2,3,4,5]

v =

     1     2     3     4     5

>> dlmwrite('myvector.txt',v)
```

```
>> v2=dlmread('myvector.txt')

v2 =

     1     2     3     4     5
```

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

m =

     1     2     3
     4     5     6

>> dlmwrite('mymatrix.txt',m)
```

```
>> m2=dlmread('mymatrix.txt')

m2 =

     1     2     3
     4     5     6
```

File I/O -2

What if data are delimited by a semicolon?

```
12.2;  123213.4;     45.7;
45.78; 1.314567;     234.7;
```

```
>> data = dlmread('data.txt', ';')
data =
   1.0e+005 *
      0.0001      1.2321      0.0005
      0.0005      0.0000      0.0023
```

?

Learn how to use,
xlsread, **xlswrite**
to use Excel files.

Random Numbers-1

REAL NUMBERS:

Generating a real random number between [0,1]: rand()

Generating an mxn matrix with real random numbers between [0,1]: rand(m,n)

Generating an mxm matrix with real random numbers between [0,1]: rand(m,m) or rand(m)

Generating a real random number between [a,b]: (b-a)*rand()+a

Generating an mxn matrix with real random numbers between [a,b]: (b-a)*rand(m,n)+a

Random Numbers-2

INTEGER NUMBERS:

Generating an integer random number between [1,k]:   randi(k)

Generating an mxn matrix with integer random numbers between [1,k]:   randi(k,m,n)

Generating an mxm matrix with integer random numbers between [1,k]:   randi(k,m,m) or randi(k,m)

Generating an integer random number between [h,k]: randi([h,k])

Generating an mxn matrix with integer random numbers between [a,b]: randi([h,k],m,n)

Exercise 1

- Define two vectors with 100 elements:

  - The first vector contains the numbers from 1 to 100

  - The second vector contains random numbers between 0 and 1. (You can create a random number between 0 and 1 using the command **rand()** in MATLAB.)
- Create a matrix with two rows and 100 columns using these vectors.
- Write this matrix in the file "randomexercise.txt". (Use dlmwrite)

Exercise 2

Main program:

Read the matrix from the file "randomexercise.txt"

Create two vectors from the matrix. You need to have the same vectors you have created in the previous exercise. (The first vector with numbers from 1 to 100 and the second one with random numbers.)

Plot the two vectors. x-axis is the elements of the first vector and the y-axis is the elements of the second vector.

Send the second vector (the one with random numbers) to a function "findeverything".

Print the output of the function.

Function "findeverything":

Find the maximum element.

Find the minimum element.

Find the average of the elements.

Find the standard deviation using

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \quad \text{where} \quad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i .$$

Here, *N* is the number of elements and x_i is the ith element.

Find the element that has the closest value to the average.

Performance-1

Vectorization:

1.
Create a vector with $N=10^6$ random values:
v=rand(1,N); creates a vector v with N random entries.

Find the number of elements whose value is greater than 0.5:

a) Using a loop (for or while)

b) Using vectorized commands/operators:
sum(v>0.5); command counts the total number of elements satisfying this condition in vector v.

2.
Taking $N=10^n$ where n=1,2,3,4,5,6,7,8, measure the evaluation time needed for the calculations described in the previous question. For example,

```
tic;
sum(v>0.5);
time1=toc
```

measures the evaluation time for the command sum(v>0.5); and assings the time value to the "time1" variable.

Plot the ("time values" vs. "n values") in two graphs, one for the case with loop and the other for the vectorization case.

Answer the question: Which method takes more time to evaluate? Explain using your results.

Performance-2

Preallocating arrays:

3.

Define a vector v with $N=10^6$ elements such that the ith element is the value i*i. Do this using a loop and measure the time needed for this procedure:

```
tic;
for i=1:N
  v(i)=i*i;
end
toc
```

Now, do the same thing but before using the loop, allocate memory for the vector. You can do this by creating a zero vector of the same size:

```
tic;
v=zeros(1,N);
for i=1:N
  v(i)=i*i;
end
toc
```

Answer the question: Does allocation of arrays before using them is a good thing for saving evaluation time? Explain by giving some proper time tests (take 10^n and n=1,2,3,4,5,6,7,8).