Introduction to MATLAB®

for

Numerical Analysis and Mathematical Modeling

Selis Önel, PhD

Advantages over other programs

- Contains large number of functions that access numerical libraries (LINPACK, EISPACK)
  ex: solves simultaneous eqn. with a single function call

- Has strong graphics support
  ex: Plots results of computations with a few statements

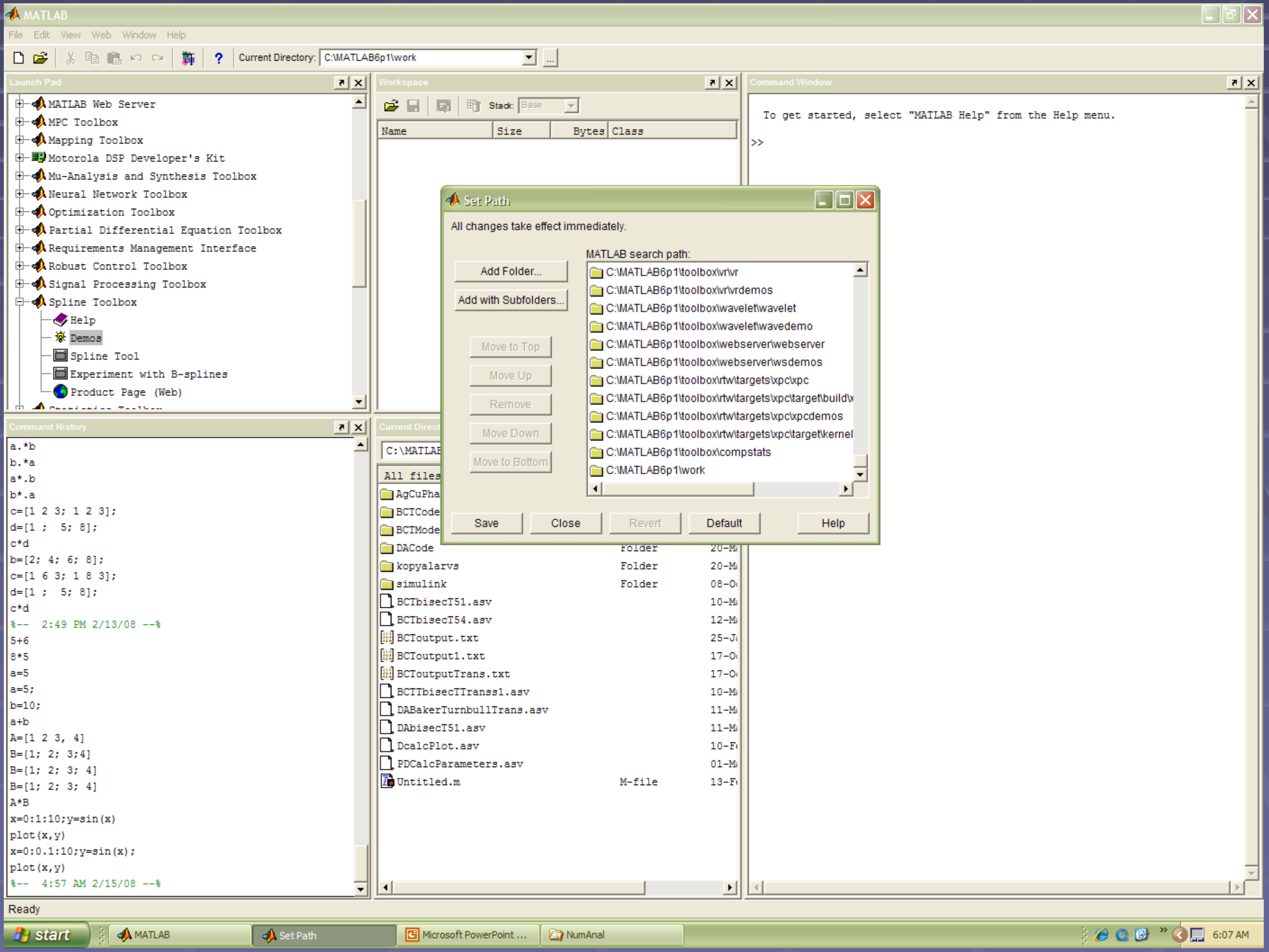
- Treats all numerical objects as double-precision arrays
  ex: Does not require declaration/conversion of data types

Reference: J. Kiusalaas, Numerical Methods in Engineering with MATLAB®, Cambridge University Press, New York, NY, 2005

Set the Path for the Folder First

Where will you keep your MATLAB® files?

- Click File and click Set Path
- Add the folder you want to use to the top of the list
- Save and Close the window
- Set the Current Directory to the same folder



Using the Command Window

The command window is *interactive*
Each command is executed upon its entry
i.e. just like an electronic calculator

>> MATLAB®'s prompt for input
% (percent) Marks the beginning of a comment
; (semicolon)

- Suppresses printout of intermediate input and results
- Separates the rows of a matrix

, (comma) Separates variables

Creating an Array

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

>> %  or create it using semicolons to make new rows
>> M = [1 2 3; 4 5 6; 7 8 9]
M =
     1     2     3
     4     5     6
     7     8     9
```

Matrix and Vector Operations

```
>> %Create a 3x3 matrix
>>A = [1 5 3; -2 4 -3; 6 7 8];    % Input 3x3 matrix
>> B = [4; 9; 0];    % Input column vector
>> A
A =
     1     5     3
    -2     4    -3
     6     7     8
>> B
B =
     4
     9
     0
>> C = A\B    % Solve A*C=B by left division
C =
   -0.5775
    1.3803
   -0.7746
```

Elements of an Array

```
>> A11=1; A12=12; A21=21; A22=22;
>> A=[A11 A12; A21 A22];
>> A
A =
     1    12
    21    22
>> % Section of this array can be extracted by use of colon notation
>> A(1,2) % Element in row 1, column 2
ans =
    12
>> A(:,2) % Elements in the second column
ans =
    12
    22
>> A(1,:) % Elements in the first row
ans =
     1    12
```

Arithmetics

```
>> A=[ 1 0 3; 2 4 6]; B=[20 30 40; -1 3 5];
>> C=A.*B                    % Element-wise multiplication
C =
   20     0   120
   -2    12    30

>> C=A*B                     % Matrix multiplication fails
??? Error using ==> *   % due to incompatible dimensions
Inner matrix dimensions must agree.

>> C=A*B`                     % Matrix multiplication
C =
  140    14
  400    40
```

Arithmetic operators

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
^	**Power**
'	**Transpose**
****	**Left division**
/	**Right division**

In MATLAB®,

- if matrix sizes are incompatible for the matrix operation, an error message will result, except for scalar-matrix operations (for addition, subtraction, and **division** as well as for multiplication)
- In scalar-matrix operations each entry of the matrix is operated on by the scalar. The "matrix **division**" operations have special cases.

Array operations:

- Addition and subtraction operate **entry-wise** but others do not
- *, ^, \, and /, can operate entry-wise if preceded a period

ex: Both [1,2,3,4]**.***[1,2,3,4] or [1,2,3,4]**.^**2 yield [1,4,9,16]

Matrix Division

If **A** is an invertible square matrix and

- **If b** is a compatible **column vector**,
  then **x = A \ b** is the solution of **A * x = b** and
- **If B** is a compatible **row vector**,
  then **x = B / A** is the solution of **x * A = B**.

In **left division**,

- if **A** is square, then it is factored using Gaussian elimination. The factors are used to solve **A * x = b**
- If **A** is not square, it is factored using Householder orthogonalization with column pivoting and the factors are used to solve the under- or over-determined system in the least squares sense

Right division is defined in terms of **left division** by

b / A = (A' \ b')'

Matrix Division

Division is not a standard matrix operation !

To remember the notation for the operators: **Backslash ** and **Slash /**

**Backslash ** solves systems of linear equations of the form Ax=b

Slash / solves systems of linear equations of the form xA=b

If **A** is an invertible square matrix and if b is a compatible vector:

Left multiplication by A^{-1} gives $A^{-1}Ax = A^{-1}b \rightarrow x = A^{-1}b \rightarrow x=A\backslash b$

Right multiplication by A^{-1} gives $xA\,A^{-1}= bA^{-1} \rightarrow x = bA^{-1} \rightarrow x=b/A$

Note: \ and / apply to nonsquare and singular systems where the inverse of the coefficient matrix does not exist.

Matrix Building Functions

A is a square matrix

n is an integer

eye(n)	Identity matrix
zeros(n)	Matrix of zeros
ones(n)	Matrix of ones
diag(A)	Diagonal matrix
triu(A)	Upper triangular part of a matrix
tril(A)	Lower triangular part of a matrix
rand(n)	Randomly generated matrix
hilb(n)	Hilbert matrix
Magic(n)	Magic square

Built-in Constants and Special Variables

%The smallest difference between two numbers

>> eps = 2.2204e-016

>> pi = 3.1416

%Limits of floating numbers shown as $a*10^b$ where $0 \le a < 10$ and $-308 \le b \le 308$

>> realmin = 2.2251e-308

>> realmax= 1.7977e+308

>> i = 0 + 1.0000i

>> j = 0 + 1.0000i

%Undefined numbers (Not a number like 0/0)

>> NaN = NaN

>> inf = Inf

%Overflow when limit is exceeded:

>>(2.5e100)^2 = 6.2500e+200

>> (2.5e200)^2 = Inf

Attention to some calculations

%1-0.4-0.2-0.4 should be equal to 0, BUT in MATLAB®:

>> 1-0.4-0.2-0.4 = -5.5511e-017

Reason:

In binary computer representation 0.2 has continuous digits after the decimal point

$(0.2)_{10}=(0.0011001100110011\ldots)_2$

So the result will never be equal to 0

Format function

Affects only how numbers are displayed, not how MATLAB computes or saves them

Type	Result	Example
short	Scaled fixed point format, with 5 digits	3.1416
long	Scaled fixed point format, with 15 digits for double; 7 digits for single	3.14159265358979
short e	Floating point format, with 5 digits	3.1416e+000
long e	Floating point format, with 15 digits for double; 7 digits for single	3.141592653589793e+000
short g	Best of fixed or floating point, with 5 digits	3.1416
long g	Best of fixed or floating point, with 15 digits for double; 7 digits for single	3.14159265358979
short eng	Engineering format that has exactly 6 significant digits and a power that is a multiple of three	3.1416e+000
long eng	Engineering format that has exactly 16 significant digits and a power that is a multiple of three	3.14159265358979e+000

Special Commands

- Clear removes all variables from the workspace
- Clc clears the command window and homes the cursor
- Clf deletes all children of the current figure with visible handles
- More Control paged output in command window:
  - More on / More off enables/disables paging of the output in the MATLAB command window
  - More(n) specifies the size of the page to be n lines
- Who lists the variables in the current workspace
- Whos lists more information about the variables and the function to which each variable belongs in the current workspace
- Who -file filename lists the variables in the specified .mat file
- Date returns current date as date string

  S = Date returns a string containing the date in dd-mmm-yyyy format
- Clock = [year month day hour minute seconds]

Simple Mathematical Functions

- abs(x) : Absolute value of x

  Ex: abs(-20.0560) = 20.0560
- sign(x) : Signum function

  For each element of x, it returns 1 if the element is greater than zero, 0 if it equals zero and -1 if it is less than zero. For the nonzero elements of complex x, sign(x) = x./abs(x)

  Ex: sign(-20.0560) = -1
- fix(x) : rounds the elements of x to the nearest integers towards zero

  Ex: fix(20.0560) = 20
- round(x) : rounds the elements of x to the nearest integers

  Ex: round(20.0560) = 20
- rem(x,y) : remainder after division

  rem(x,y) is x-n.*y where n=fix(x./y) if y ~= 0. If y is not an integer and the quotient x./y is within roundoff error of an integer, then n is that integer

  The inputs x and y must be real arrays of the same size, or real scalars

  Ex: rem(20.056,5) = 0.0560

Simple Mathematical Functions

- exp(x) : Exponential of the elements of x, e to the x
  For complex z=x+i*y, exp(z) = exp(x)*(cos(y)+i*sin(y))
  Ex: exp(100) = 2.6881e+043, exp(-100) = 3.7201e-044
  exp(100+i*100) = 2.3180e+043 -1.3612e+043i

- log(x) : natural logarithm of the elements of x.
  Ex: log(100) = 4.6052,   log(-100) = 4.6052 + 3.1416i

- log10(x) : Common base 10 logarithm of the elements of x
  Ex: log10(100) = 2,   log10(-100) = 2.0000 + 1.3644i

- sqrt(x) : square root of the elements of x
  Ex: sqrt(100) = 10,   sqrt(-100) = 0 +10.0000i

Complex results are produced if x is not positive in log(x), log10(x), sqrt(x)

Comparison Operators and Logic Operators

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

== Equal to

~= Not equal to

& AND

| OR

~ NOT

Comparison Operators and Logic Operators

```
>> A=[ 1 0 3; 2 4 6]; B=[20 30 40; -1 3 5];

>> (A>B)|(B>=5)
ans =
    1    1    1
    1    1    1

>> (A<=5)&(B<=5)
ans =
    0    0    0
    1    1    0
```

Flow Control

Conditionals: If, else, elseif

Switch: case

Loops: while, for,

break, continue,

return, error

Flow Control

```
%This exercise uses if, else, elseif conditionals
a=5; b=50; c=5*10^4;
d=a*b;
if d<c
    d=d;
elseif d==c
    d=c;
else
    d=0;
end
d
```

Flow Control
Compare the results for the following

Try both cases

```
% This exercise uses
% the while loop
agemax=100; age=0;
while age<agemax
age=age+1
end
age
```

```
% This exercise uses
% the while loop
agemax=100; age=0;
while age<agemax
age=age+1;
end
age
```

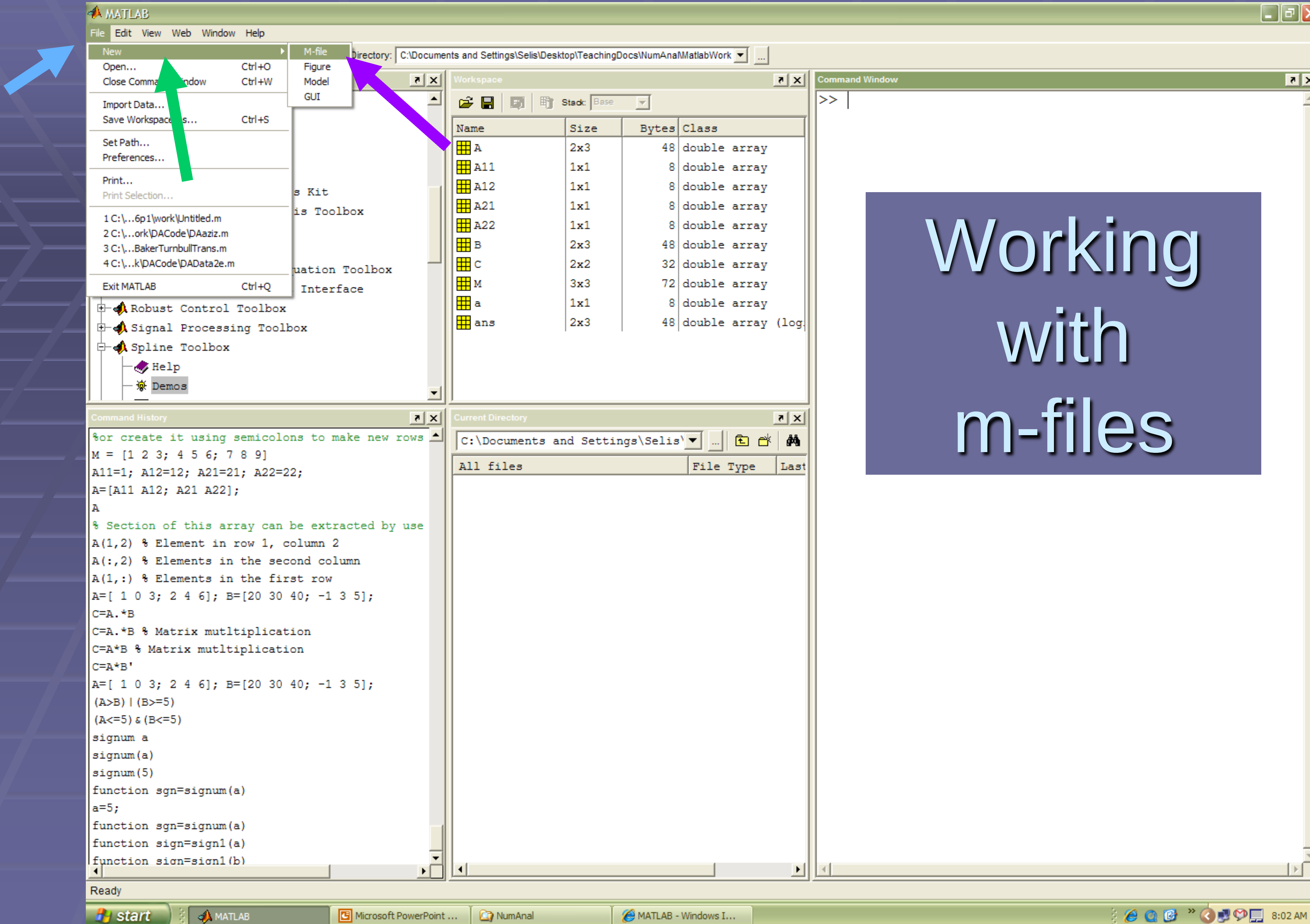
Flow Control
Compare the results for the following

Try both cases

```
% This exercise uses
% the for loop
for n=0:1:10;
   y(n+1)=2^(n);
end
y
```

```
% This exercise uses
% the for loop
for n=0:1:10;
   y(n+1)=2^(n)
end
y
```

Working with m-files



Working with m-Files: Functions

Write this in a new m-file and <u>save</u> it as ExConditionalsFunc, i.e., the exact name of the function, and <u>run</u> it

```
%This exercise uses
%if, else, elseif conditionals
% in a function you create
function d=ExConditionalsFunc(a,b,c)
d=a*b;
if d<c
   d=d;
elseif d==c
   d=c;
else
   d=0;
end
d
```

Use the Command Window (or a new m-file) to assign values to *a*, *b*, *c* and call the function to calculate *d*

```
>> a=1; b=3; c=8;
>> ExConditionalsFunc(a,b,c)
d =
    3
```

Finding Roots Using Built-in Functions ‘roots’ and ‘fzero’

```
% x = fzero(f,x0)
% tries to find a zero of f near x0
% Write an anonymous function f:
f = @(x)x.^4-3*x-4;
% Then find the zero near x0=-2:
x0=-2;
z = fzero(f,x0)
% To find all the roots of a polynomial f
% use roots([c1 c2 c3 ...])
f_root=roots([1 0 0 -3 -4])
```

When you run this script in an m-file, here is what you will see in the Command Window

```
z =
    -1

f_root =
   1.7430
  -0.3715 + 1.4687i
  -0.3715 - 1.4687i
  -1.0000
```

MATLAB variables are …

Case sensitive

- **MyNumber** and **mynumber** represent different variables
- Length of the name is unlimited, but the first *N* characters are significant
- To find *N* for the MATLAB installed on your computer type: namelengthmax
- Applies to: Variable names, Function and subfunction names, Structure fieldnames, Object names, M-file names, MEX-file names, MDL-file names

Displaying numbers and values on the command window

- Omit the **;** at the end of the line

```
>> cost=500;
>> cost=500
cost =   500
```

- Use the **disp** command

```
>> disp(cost), disp('dollars')
500
dollars
>> disp([num2str(cost), ' dollars'])
1000 dollars
```

- Use the **fprintf** command

```
>> fprintf('1. cost = %3.2f  \n2. cost = %3.2e \n3. cost = %3.2g \n ', cost,
   cost, cost)
1. cost = 500.00
2. cost = 5.00e+002
3. cost = 5e+002
```

Command fprintf

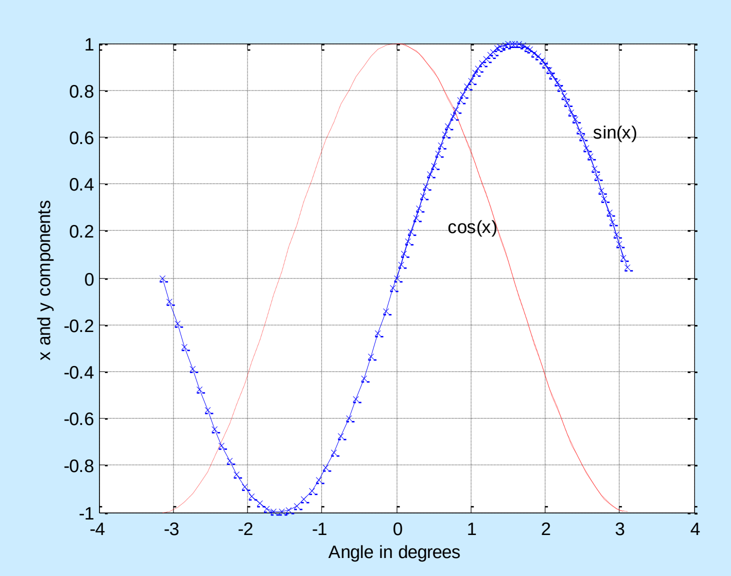
```
x = 0:.5:5; y = [x; exp(x)];
fid = fopen('fprintfex.txt','wt');
fprintf(fid,'%6.2f  %12.8f\n',y);
fclose(fid);

%   Now examine the contents of
    fprintfex.txt:
```

```
>> type fprintfex.txt
  0.00    1.00000000
  0.50    1.64872127
  1.00    2.71828183
  1.50    4.48168907
  2.00    7.38905610
  2.50   12.18249396
  3.00   20.08553692
  3.50   33.11545196
  4.00   54.59815003
  4.50   90.01713130
  5.00  148.41315910
```

Plotting in MATLAB®

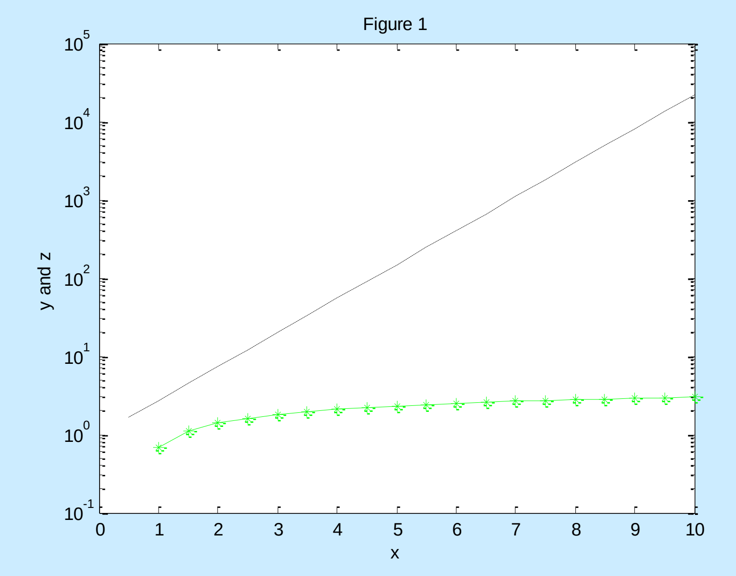
Write and run the following script in an m-file or in the command window

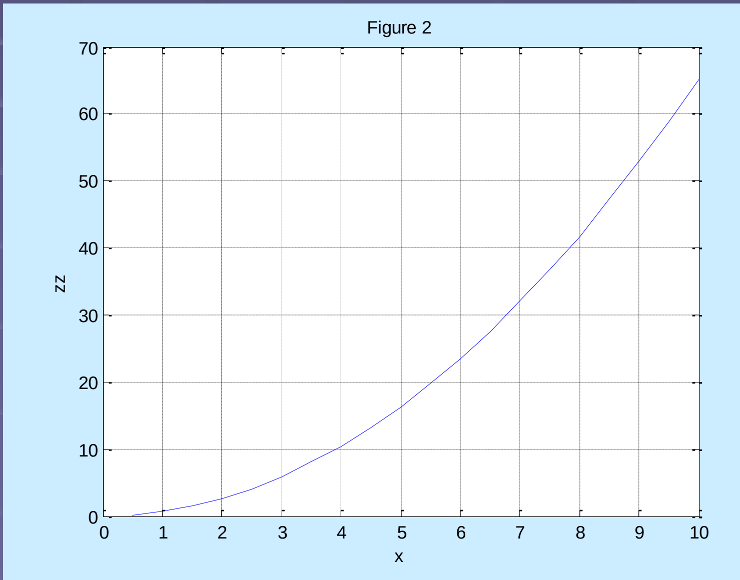
```
% This program draws a graph of sin(x) and cos(x)
% where 0 <= x <= 3.14
ang=-pi:0.1:pi;            % Create array
xcomp=cos(ang);            % Create array
plot(ang,xcomp,'r:');      % Plot using dots(:) with red (r)
hold on                    % Add another plot
                           % on the same graph
ycomp=sin(ang);            % Create array
plot(ang,ycomp,'b-x');% Plot using lines(-)
                           % and the symbol x at each
                           % data point with blue (b)
grid on                    % Display coordinate grids
xlabel('Angle in degrees');      %Display label for x-axis
ylabel('x and y components'); %Display label for y-axis
gtext('cos(ang)');         % Display mouse-movable text
gtext('sin(ang)');         % Display mouse-movable text
```



Plotting

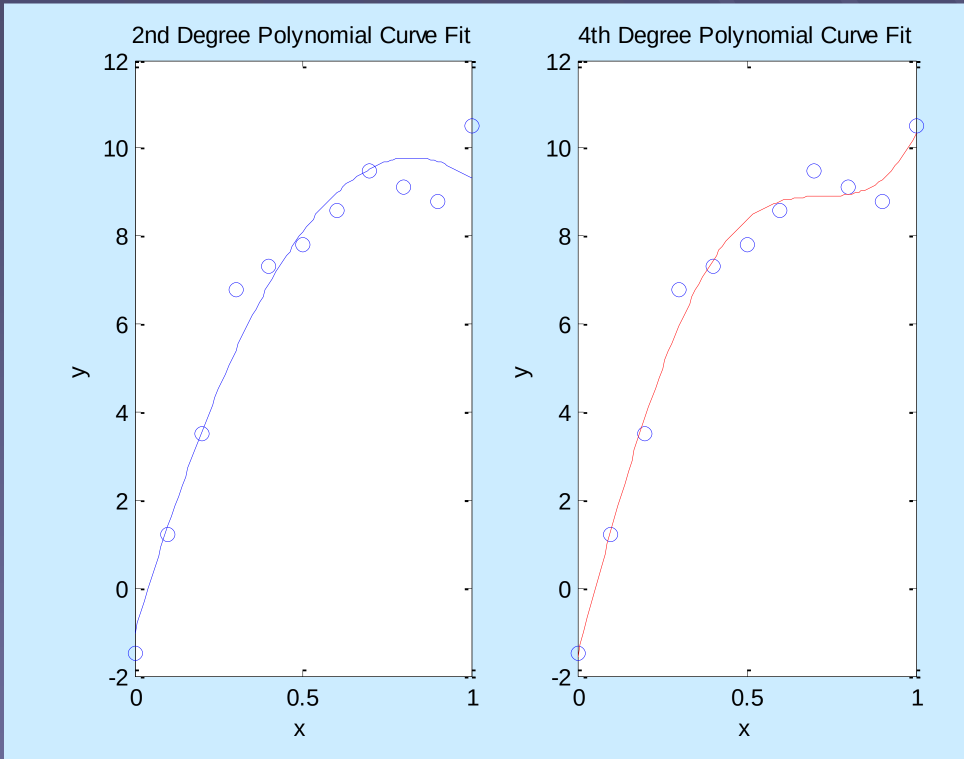
```
% This program draws multiple graphs
a=0.652;          % Assign constant parameter a
x=10:-0.5:0.5;  % Create x-array
y=exp(x);          % Create y-array
z=log(2*x);       % Create z-array
zz=a*x.^2;        % Create zz-array
figure(1)           % Create a figure
semilogy(x,y,'k-.',x,z,'g-*');
                         % Use logarithmic plot on y axis
xlabel('x');         % Display label for x-axis
ylabel('y and z');% Display label for y-axis
title('Figure 1'); % Insert title for figure
figure(2)             % Create new figure
plot(x,zz,'-');       % Plot using lines(-)
xlabel('x');           % Display label for x-axis
ylabel('zz');          % Display label for y-axis
title('Figure 2');  % Insert title for figure
grid on                 % Display coordinate grids
```





Plotting: Subplots

```
% Using polyfit to fit an n order curve to data y and
% drawing multiple figures in one page
x=0:0.1:1;
y=[-1.5 1.2 3.5 6.8 7.3 7.8 8.6 9.5 9.1 8.8 10.5]; % Data points
% Coefficients for the 2nd degree polynomial curve fit:
coef2 = polyfit(x,y,2);
% Coefficients for the 4th degree polynomial curve fit:
coef4 = polyfit(x,y,4);
% Generate 101 points between 0 and 1:
xi = linspace(0,1,101);
% Get the values of the polynomials at xi:
yi2 = polyval(coef2,xi);
yi4 = polyval(coef4,xi);
figure(1)          % Create a figure page
subplot(1,2,1)  % First plot on the first row
plot(x,y,'o',xi,yi2,'b-');
xlabel('x'); ylabel('y');
title('2nd Degree Polynomial Curve Fit');
subplot(1,2,2)   % Second plot
plot(x,y,'o',xi,yi4,'r-');
xlabel('x'); ylabel('y');
title('4th Degree Polynomial Curve Fit');
```

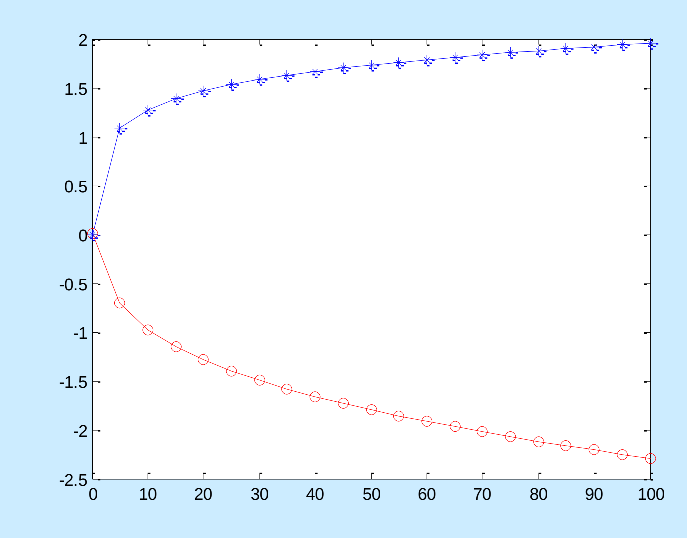


Using 'fsolve' in Solving Parameterized Functions

```
x0=[-1 1];              % Assign initial values
k=0;  C=-5;
while C<100,
C=C+5;  k=k+1;
% Call optimizer:
[x,fval] = fsolve(@(x) flinsys(x,C),x0);
num1=x(1);              % Assign x(1) to a scalar number
num2=x(2);              % Assign x(2) to a scalar number
x1(k)=num1;             % Create a new array for x(1)
x2(k)=num2;             % Create a new array for x(2)
c(k)=C;                 % Create a new array for C
x0=[x1(k) x2(k)];       % Assign x1 and x2 as initial
    values
% Save workspace variables to the binary "MAT-file":
save('ExPlot3.mat','c','x1','x2');
end
% Load workspace variables from disk:
load('ExPlot3.mat');
plot(c,x1,'r-o',c,x2,'b-*');
```

```
% This is a function for a nonlinear
%  system of algebraic equations
function F = flinsys(x,C)
F = [5*x(1)+3*(x(2))^2;
    8*(x(1))^3-2*x(2)+C];
end
```

flinsys.m



MATLAB® Study Sources

- Go to Mathworks web site OR
- Just type the following in Google Search
  - MATLAB introduction
  - MATLAB tutorial

to find various very useful sources in personal and university web sites