MATLAB Course

Adriana Hera

ahera @wpi.edu

Computing & Communications Center

- October 2010-

Part of the notes are from Matlab documentation

MATLAB Refresher Course

1.Variables, Operators
2.Matrices
3.Matlab Functions
4.Relational operators & Loops (Flow Control)
5.Scripts
6.User Defined Functions
7.Visualization
8. Curve fitting: Polynomial curve fitting
9. Interpolation
10.Publishing a script to HTML

What is Matlab ?

- **MATLAB® is a high-performance language for technical computing**.

- It integrates **computation**, **visualization**, and **programming** in an *easy-to-use environment* where problems and solutions are expressed in familiar mathematical notation.

- **MATLAB** stands for **matrix laboratory.**

- MATLAB is an interactive system whose **basic data element** is an **matrix** (**array)** that does not require dimensioning.

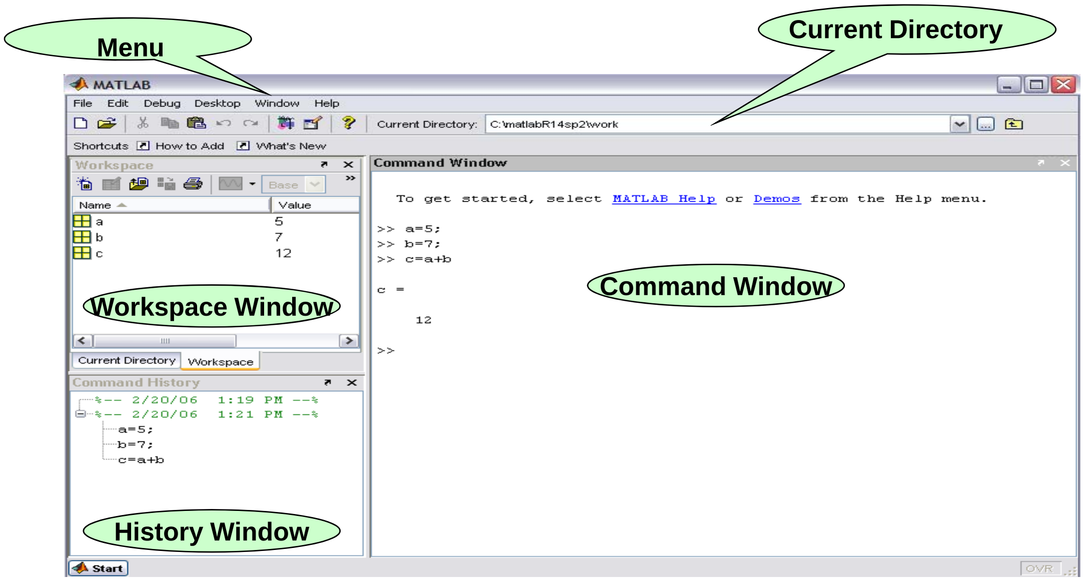
- This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

MATLAB -Typical Uses

- **Math and computation**
- **Algorithm development**
- **Data acquisition**
- **Modeling, simulation, and prototyping**
- **Data analysis, exploration, and visualization**
- **Scientific and engineering graphics**
- **Application development, including graphical user interface building**

Starting Matlab

- *Windows:* Start menu → Matlab → Matlab
- *Unix:* Terminal window→ type **`matlab`**


Menu
Current Directory
MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\matlabR14sp2\work
Shortcuts How to Add What's New
Workspace
Base
Command Window
Workspace Window
Command Window
Current Directory Workspace
Command History
History Window
Start
OVR

Name	Value
a	5
b	7
c	12

```
 To get started, select MATLAB Help or Demos from the Help menu.

>> a=5;
>> b=7;
>> c=a+b

c =

    12

>>
```

```
%-- 2/20/06  1:19 PM --%
%-- 2/20/06  1:21 PM --%
   a=5;
   b=7;
   c=a+b
```

Matlab Help

1. Using HELP menu → MATLAB Help

   HELP → Using Help Browser

2. >> helpdesk   Opens the Help browser.

3. `>> help commandname/toolboxname/functionname`

   `Ex: >> help sin`

4. `>> doc commandname/toolboxname/functionname`

   displays the detailed info in the Help browser.

   Ex: `>> doc sin`

Other commands:

`5. >> lookfor =  helpdesk -> search`

I. Matlab Programming

- ➢ Matlab Variables
- ➢ Numbers
- ➢ Operators
- ➢ Functions

…..

Matlab Variables

- A MATLAB variable is essentially a tag that you assign to a value in memory.
- MATLAB does not require any type declarations or dimension statements.
- When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage.
- If the variable already exists, MATLAB changes its contents.
- Variable names consist of a letter, followed by any number of letters, digits, or underscores.
- MATLAB uses only the first 64 characters of a variable name.
- *** MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters.*
- MATLAB stores variables in a part of memory called workspace.
- To view what is stored in a variable type its name.

Types of Variables: MATLAB provides three basic types of variables:

Local Variables
Global Variables
Persistent Variables

Matlab Variables

Rules for variable names:

- Make Sure Variable Names Are Valid
- Don't Use Function Names for Variables
- Check for Reserved Keywords
- Avoid Using i and j for Variables

Syntax:

```
varialeName=Value;
```

Example:

```
>> a=5;
>> b=7;
>> c=a+b
>> method=‘linear’
```

How to remove a variable from workspace:

```
>> clear variableName
>> clear
```

- removes all variables from the workspace (!!!!)

ans = default variable, when the result is not assign to a variable

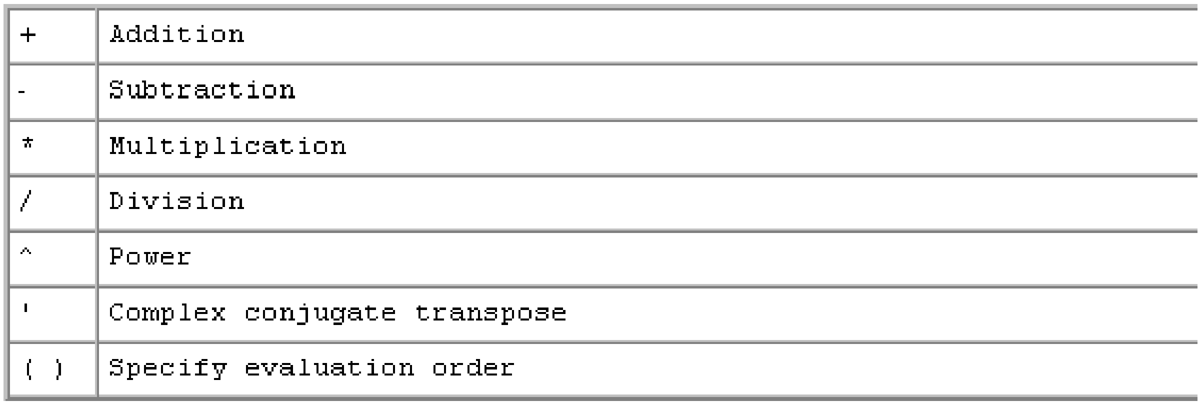
Exercise: 1. Define **a1=8 and b2=8, c1=a1+b2**

2. Other commands: **variable= input(‘prompt’)** **(>>help input)**

```
>> a3=input(‘Please enter the value of a3:’)
```

Operators

Expressions use familiar **arithmetic operators** and precedence rules.



+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Functions

1. Standard elementary mathematical functions

```
>> help elfun
```

Trigonometric (`sin, cos`)

Exponential (`exp, log`)

Complex (`abs, angle`)

Rounding and remainder (`round`)

2. Elementary matrices and matrix manipulation.

```
>> help elmat
```

3. Specialized math functions.

```
>>  help specfun
```

Functions

1. Built-in functions (Ex. sqrt, sin)

Some of the functions, like `sqrt` and `sin`, are built in.

Built-in functions are part of the MATLAB core

They are very efficient

The computational details are not readily accessible.

(you cannot see the code)

2. Function implemented in M-files (ex. `factorial`, `mean`, `det`)

You can see the code and even modify it, if you want.

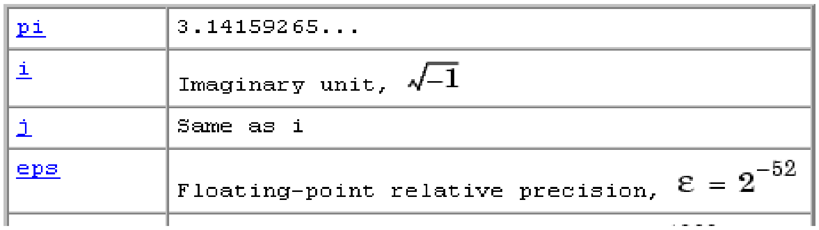
Syntax:

```
>> outputArgs = functionName(inputArgs)
```

Related commands: `edit`

Constants, inf, NaN

Constants



pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, $\varepsilon = 2^{-52}$

inf Infinity : division by zero and overflow, which lead to results too large to represent as conventional floating-point values.

```
ex: 1/0, 1.e1000
```

NaN Not-a-Number: a result of mathematically undefined operations like 0.0/0.0 and inf-inf.

Related commands: `edit`

✓ I. Matlab Programming

We talked about:

- ✓ Matlab Variables
- ✓ Numbers
- ✓ Operators
- ✓ Functions

→ **II. Matlab Programming**

I. Matlab Programming

- Matrices
- Operators
- Functions

…..

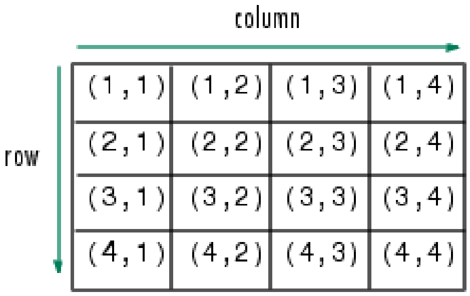
Matrix & basic matrix functions

Define a matrix:

1. Type the matrix
2. Use Specialized Matrix Functions

Matrix Manipulation

Matrix Functions



(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

Matrix: Define a matrix

1. Type the matrix

- Separate the elements of a row with **blanks** or **commas**.
- Use a **semicolon**, **;** , to indicate the end of each row.
- Surround the entire list of elements with **square brackets**, [ ].

$$S = \begin{bmatrix} 3 & -10 & 0 \\ -10 & 0 & 30 \\ 0 & 30 & -27 \end{bmatrix}$$

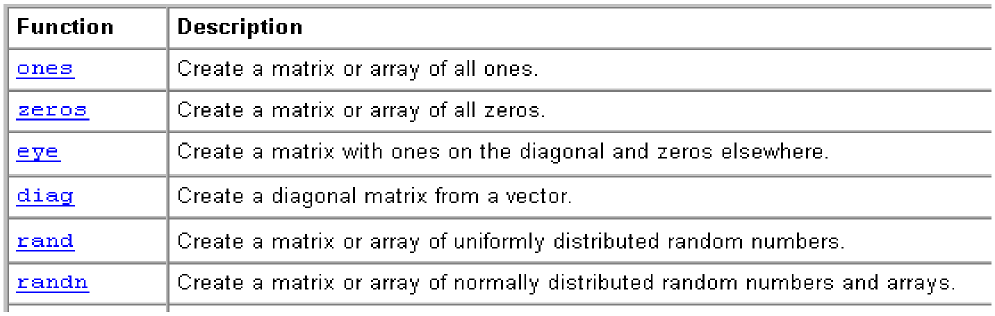
```
1. >>  S=[3 -10 0; -10 0 30; 0 30 -27]
```

Basic matrix information: **size** (size of a matrix)

```
>> [m,n] = size(X)
```

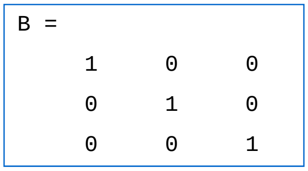
Matrix: Define a matrix

2. Use Specialized Matrix Functions



Function	Description
ones	Create a matrix or array of all ones.
zeros	Create a matrix or array of all zeros.
eye	Create a matrix with ones on the diagonal and zeros elsewhere.
diag	Create a diagonal matrix from a vector.
rand	Create a matrix or array of uniformly distributed random numbers.
randn	Create a matrix or array of normally distributed random numbers and arrays.

```
>> B=eye(3)
```



```
B =

     1     0     0
     0     1     0
     0     0     1
```

```
>> D=diag([23,0,47])
```

Matrix: Accessing Matrix Elements

$$S = \begin{bmatrix} 3 & -10 & 0 \\ -10 & 0 & 30 \\ 0 & 30 & -27 \end{bmatrix}$$

- **individual element**

```
>> S(2,2)   →   ans =
                      0
```

- **column**

```
>>  S(:,2)   →   ans =
                 -10
                  0
                 30
```

: (colon) → all elements

- **row**

```
>> S(2,:)   →   ans =
                     -10     0     30
```

- **group of elements**

```
>> S(3,1:2)   →   ans =
                        0     30
```

first element : step: last element

Define a vector

first element : step: last element

```
v=[1:0.5:20];
      v=[1, 1.5, 2, 2.5, …  19, 19.5, 20]
```

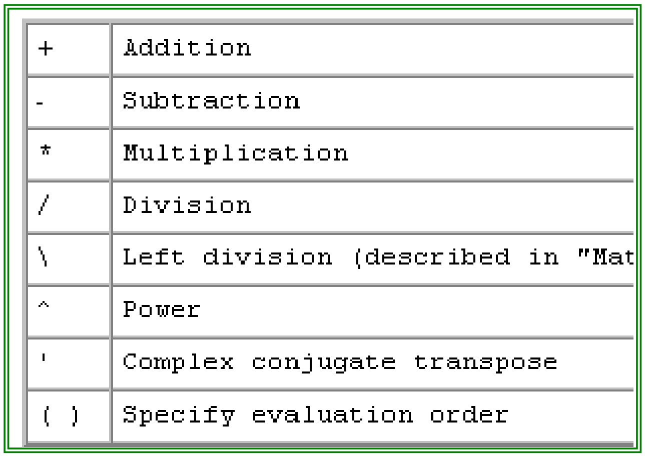
linspace

```
y = linspace(a,b,n)
```

generates a row vector y of n points linearly spaced between and including a and b.

Example: `>> y=linspace(2.1, 10, 9)`

Matrix: Operations



+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in "Mat
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

A+B

A-B

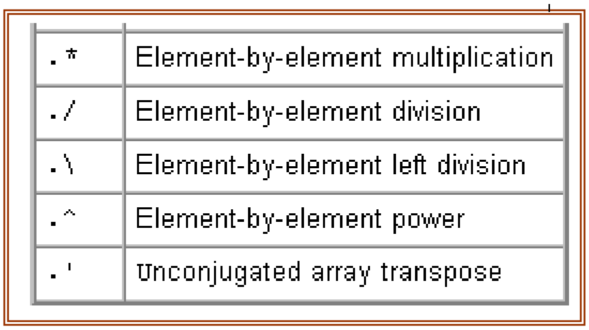
A*B

A/B

A\B

A^B

A'



.*	Element-by-element multiplication
./	Element-by-element division
.\	Element-by-element left division
.^	Element-by-element power
.'	Unconjugated array transpose

A.*B

A./B

A.\B

A.^B

A.'

Matrix: Operations

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 2 & 3 & 1\\ 2 & 2 & 2\end{bmatrix},\qquad \mathbf{B}=\begin{bmatrix}10 & 20 & 30\\ 11 & 21 & 31\\ 1 & 2 & 3\end{bmatrix},$$

```
>> A=[1 2 3; 2 3 1; 2 2 2];
>> B= [10 20 30; 11 21 31; 1 2 3];
```

```
>> A*B
```

```
ans =
       35     68    101
       54    105    156
       44     86    128
```

$$\mathbf{A}=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix},\qquad \mathbf{B}=\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\\ b_{31} & b_{32} & b_{33}\end{bmatrix},$$

$$\mathbf{A}\cdot{*}\mathbf{B}=\begin{bmatrix}a_{11}\cdot b_{11} & a_{12}\cdot b_{12} & a_{13}\cdot b_{13}\\ a_{21}\cdot b_{21} & a_{22}\cdot b_{22} & a_{23}\cdot b_{23}\\ a_{31}\cdot b_{31} & a_{32}\cdot b_{32} & a_{33}\cdot b_{33}\end{bmatrix}$$

```
>> A.*B
```

```
ans =
    10    40    90
    22    63    31
     2     4     6
```

Matrix: Functions

Few matrix functions :

```
det     -Determinant
linsolve-Solve linear systems of equations (using LU factorization)

\       - Linear equation solution
        (X = A\B is the solution to the equation AX = B
        computed by Gaussian elimination)
```

Matrix: Solution of a linear system

x = A\b is the solution to the equation Ax = b

computed by Gaussian elimination

A=square matrix

$$\begin{aligned} x_1 + \quad x_2 + \quad x_3 &= 2 \\ x_1 + \quad 2x_2 + \quad 3x_3 &= 5 \\ x_1 + \quad 3x_2 + \quad 6x_3 &= 7 \end{aligned} \qquad \begin{aligned} x_1 &= ? \\ x_2 &= ? \\ x_3 &= ? \end{aligned}$$

\ = ldivide

```
>>A=[1,1,1;1,2,3;1,3,6];
>>b=[2;5;7];
1. >>x=A\b
2. >> x=linsolve(A,b)
3. >> x =inv(A)*b
```

Not recommended!!!

```
x =
   -2
    5
   -1
```

(x = A\b is the solution to the equation Ax = b computed by Gaussian elimination)
linsolve-Solve linear systems of equations (using LU factorization)

Matlab Programming

We talked about:

….

✓ Matrices

✓ Operators

✓ Functions

…..

IV. Matlab Programming

- How to write a program (M-files)
  - Script
  - Function
- How to plot data

M-files

- **Files that contain code in the MATLAB** language are called *M-files*.
- You create M-files using a text editor.
- Use a M-file as any other MATLAB function or command.
- A M-file is a plain text file.

Two kinds of **M-files**:

Scripts

do not accept input arguments or *return output arguments*

operate on data in the workspace.

Functions

can accept input arguments and *return output arguments*

internal variables are local to the function.

```
>> edit   fileName
>> edit  exSwitch
```

M-files: Scripts

- do not accept input arguments or *return output arguments*
- operate on data in the workspace.

```
>> edit  myScript
```

type the code

File/Save

```
>> myScript
```

Example →

(to run the script type its name)

```
% comments
clear; close; clc
tol=1e-2; x=1;
k=0; f=0; fk=1;
while fk>tol
    fk=x/(factorial(k));
    f=f+fk;
    k=k+1;
end
disp(['f=', num2str(f), '    k=',...
num2str(k)])
```

To practice:

*** Use command window**

•To present hw write scripts or functions   (.m files)

% - indicates a comment

Matlab - Plotting

`plot`

Sintax:

```
plot(y);        plot(x,y);     plot(x,y,s)
```

The plot function has different forms, depending on the input arguments.

If y is a vector, `plot(y)` produces a piecewise linear graph of *the elements of y* versus the *index of the elements of y*.

If you specify two vectors as arguments, `plot(x,y)` produces a *graph of y versus x*.

Matlab - Plotting

```
plot(x,y, s);
```

s allows to plot : colors, symbols, different lines

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		….			

`plot (x,y,'c+:')` plots a cyan dotted line with a plus at each data point;

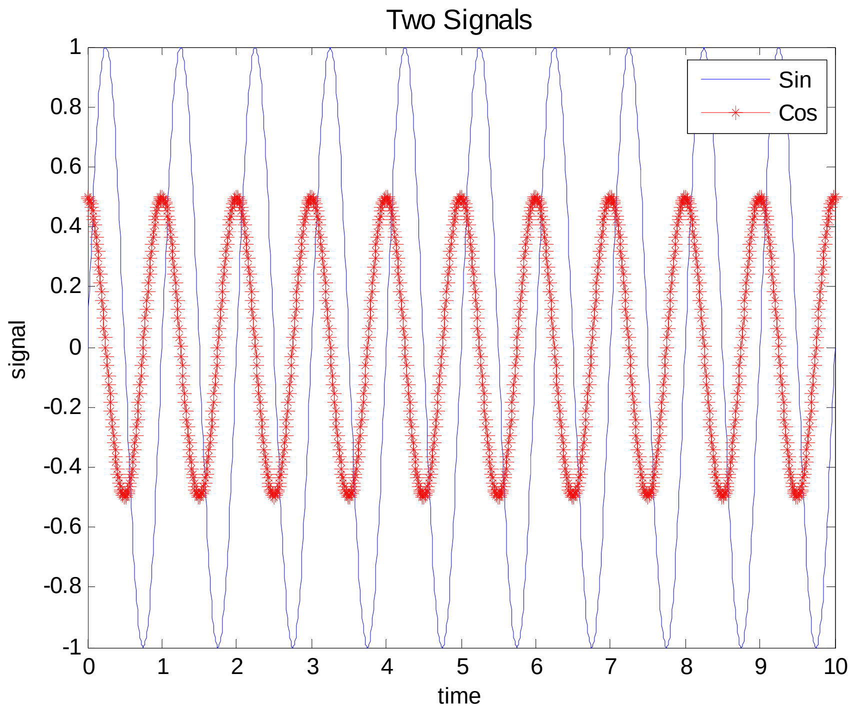
Matlab - Plotting

```
clear
t=0:0.01:10; % time seconds
signalSin=sin(2*pi*t); % signal1 - frequency =1 Hz
signalCos=0.5*cos(2*pi*t); % signal2 - frequency =1 Hz

figure
plot(t,signalSin);

hold on
plot(t,signalCos, '-*r');

xlabel('time');  ylabel('signal');
legend('Sin', 'Cos');
title('Two Signals','FontSize',12)
```

plot2signals.m

Other commands: figure xlabel ylabel legend, title

Matlab - Plotting

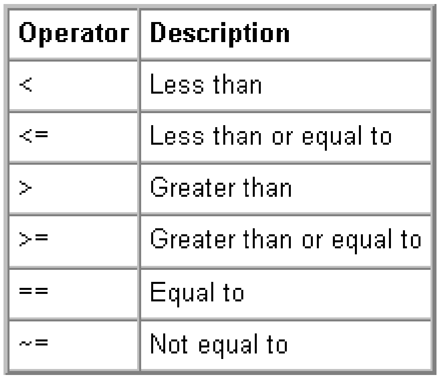


Relational Operators

The relational operators are <, >, <=, >=, ==, and ~=.

Relational operators perform element-by-element comparisons between two arrays.

→ Logical array with elements set to logical 1 (true) or to logical 0 (false)



Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

```
syntax
A < B
A > B
A <= B
A >= B
A == B
A ~= B
```

Ex. $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

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

>> A>2
ans =
    0    0    1
    1    1    1
```

```
Ex:
>>  v=rand (1, 10000);
>>  v1=find (v>0.5);
>>  v2=find (v<0.5);
```

Loops (Flow Control)

MATLAB has several flow control commands:

- **if**, **else**, and **elseif**
- **switch** and **case**
- **for**
- **while**

- **continue**
- **break**
- **return**

Flow Control: if … else

if : conditionally executes statements

```
if relation
        statements 1
else
        statements 2
end
```

Example:

```
a=5; b=7;
if a>b
   disp('a greater than b');
else
   disp('b greater than a');
end
```

```
if expression1
              statements1

elseif  expression2
              statements2
else
              statements3
end
```

Flow Control: for

The for loop executes a group of statements a number of times.

```
for  variable = expression

         statements
end
```

```
x=2; % exp(2)
for k = 1:5
   r(k) = x^(k-1)/(factorial(k-1));
end
f=sum(r);

disp(['f=', num2str(f)]);
disp(['r=',num2str(r)]);
```

expression:

first value: last value

first value: step: last value

file: exFor.m

$$e^x=\sum_{k=0}^{\infty}\frac{x^k}{k!}=\underset{\substack{\downarrow\\ \mathbf{r(1)}}}{\boxed{\frac{x^0}{0!}}}+\frac{x^1}{1!}+\frac{x^2}{2!}+\cdots+\underset{\substack{\downarrow\\ \mathbf{r(k+1)}}}{\boxed{\frac{x^k}{k!}}}+\cdots$$

Change of increment:

for k = 1:2:10; statement; end;

for k=10:-1:1; statement; end;

Flow Control: while

The while loop executes a group of statements until a logical test is false.

```
while expression
      statements
end
```

Example:

```
tol=1e-2; x=2;
k=1; f=1; fk=1;
while fk>tol
      fk=x^k/(factorial(k))
      f=f+fk;
      k=k+1;
end
disp(['f=', num2str(f), '   k=', num2str(k)])
```

File: exWhile.m

$$e^x = \sum_{k=0}^{\infty}\frac{x^k}{k!} = \frac{x^0}{0!}+\frac{x^1}{1!}+\frac{x^2}{2!}+\cdots+\boxed{\frac{x^k}{k!}}+\cdots$$

fk

Flow Control: switch, case

Ex: Find the structure of the command.

```
>> help switch
>> doc switch
```

My Example:

```
clear
a=6; b=2;
method=input(' method=');
switch method
    case 1
        c=a+b;
    case 2
        c=a*b;
    case 3
        c=a/b;
    otherwise
        disp('no valid method')
end
```

files: exSwitch.m

exSwitch2.m

Other commands: **input**, **disp**

Loops: Exit commands

`break`

Lets you exit early from a for loop or while loop.

In nested loops, break exits from the innermost loop only.

`return`

Terminates the current sequence of commands.

Returns control to the invoking function or to the keyboard.

`Ctrl + C`

Emergency exit

✓ III. Matlab - Programming

We talked about:

- ✓ Relational operators (>, <. <=, >=…)
- ✓ Loops (Flow Control)
  - ✓`if`, `else`, and `elseif`
  - ✓`switch` and `case`
  - ✓`for`
  - ✓`while`
  - ✓`continue`
  - ✓`break`
  - ✓`return`

Matlab Programming

- ✓ Scripts
- ✓ Visualization
- ➢ Functions

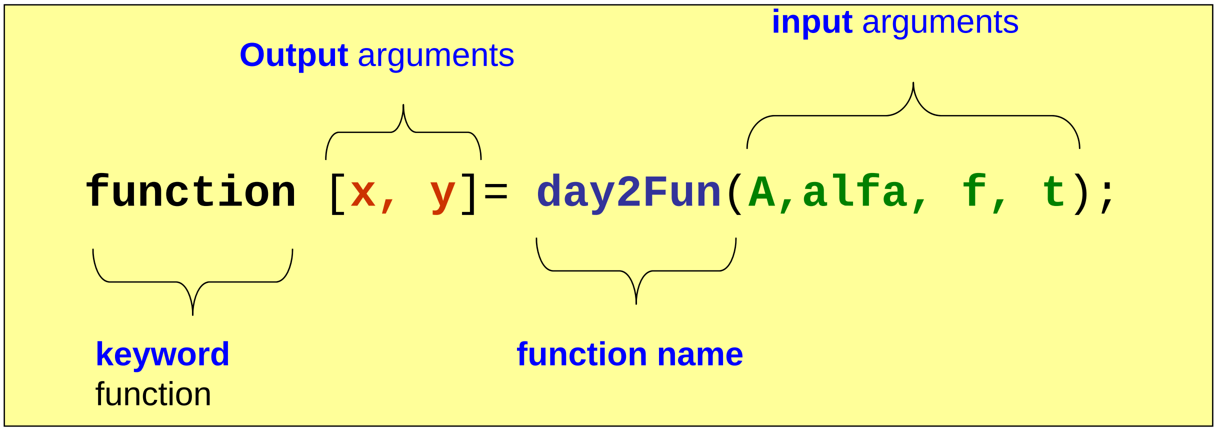
FUNCTIONS

- Functions are M-files that can accept input arguments and return output arguments.
- The M-file and functions should have the same name.
- Each M-file function has an area of memory, called the function workspace.
- Separate from the MATLAB base workspace, in which it operates.
- Matlab functions can be found in:

```
C:\matlabR14sp2\toolbox\matlab\
```

FUNCTIONS

Function definition:



input arguments

Output arguments

```
function [x, y]= day2Fun(A,alfa, f, t);
```

keyword
function

function name

Calling the function:

```
>> [x1, y1]= day2Fun (A1,alfa1, f1, t1);
```

FUNCTIONS

Function definition:

Open ***exWhile.m***

```
function [f, k]=myExp(x, tol)
% function description

k=1; f=1; fk=1;
while fk>tol
    fk=x/(factorial(k));
    f=f+fk;
    k=k+1;
end
disp(['f=', num2str(f), '   k=', num2str(k)])
```

myExp1.m

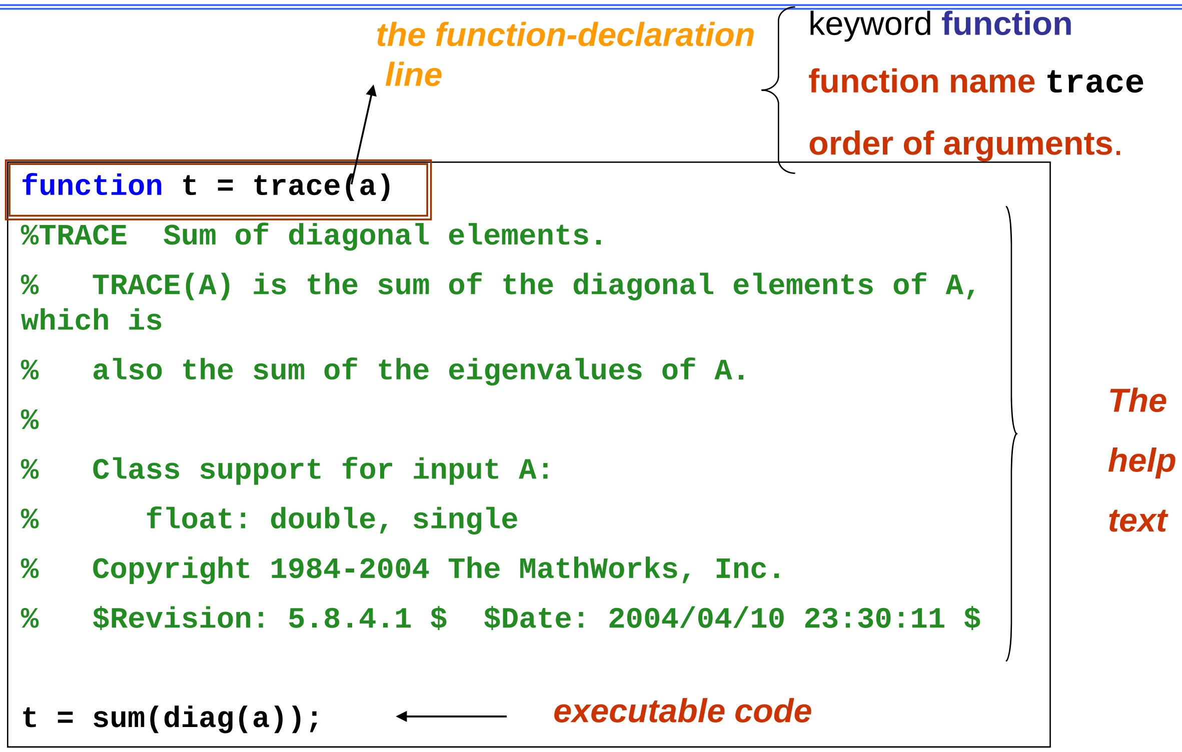
Main code

```
>> [f1, k1]=myExp(x1, tol1)
```

fk

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \cdots + \boxed{\frac{x^k}{k!}} + \cdots$$

FUNCTIONS



the function-declaration line

keyword function

function name trace

order of arguments.

```
function t = trace(a)
%TRACE  Sum of diagonal elements.
%   TRACE(A) is the sum of the diagonal elements of A,
which is
%   also the sum of the eigenvalues of A.
%
%   Class support for input A:
%      float: double, single
%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 5.8.4.1 $  $Date: 2004/04/10 23:30:11 $

t = sum(diag(a));
```

The help text

executable code

```
>> trace(A)
>> results= trace(A);
```

Function: Inline functions

There are essentially two ways to create a new function in MATLAB:

1. in a command entered at run-time (inline)

2. or in a file saved to permanent storage.

inline **function, feval**

```
>> f=inline('x^2+y^2‘, ‘x’, ‘y’)
f =
     Inline function:
     f(x,y) = x^2+y^2
>> m=f(1,2)
m =5
```

Vector form

```
>> f=inline('x.^2+y.^2‘, ‘x’, ‘y’)
```

file **f.m**

```
function result=f(x,y)
result=x^2+y^2;
```

Curve fitting: Polynomial curve fitting

polyfit

finds the coefficients of a polynomial **p**(x) of degree n that fits the data to

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

Syntax:
```
p = polyfit(x,y,n)
```

myFit.m

```
To see how good the fit is, evaluate the polynomial
at the data points with
f = polyval(p,x);
```

Figure/Tools/Basic Fitting

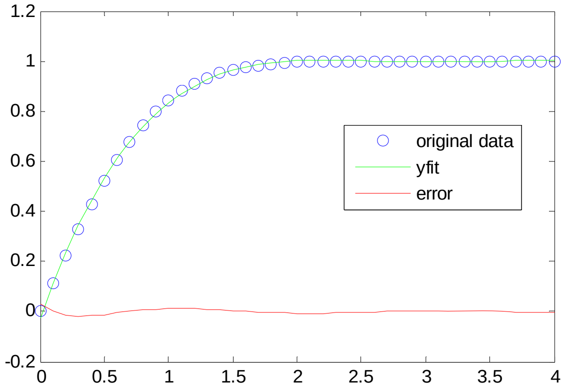
other forms:
```
[p,S] = polyfit(x,y,n)
[p,S,mu] = polyfit(x,y,n)
```
What does mu mean? see help polyfit / doc polyfit

```
Advanced commands:
To fit an arbitrary function
find the parameters by
minimizing (fminsearch) the sum
of squares of errors between the
data and the given function.
```

Curve fitting: Polynomial curve fitting

Example:

```
>> %%% generate data %%%%%%
>> x = (0: 0.1: 4)';
>> y = erf(x);
>> figure
>> plot(x,y, 'bo');
>> %% polynomial fitting %%%%
>> p = polyfit(x,y,4)
p =
   -0.0135    0.1662   -0.7480    1.4538   -0.0271
>> f = polyval(p,x);
>> plot(x,y,'bo',x,f,'g')
>> hold on
>> legend('original data', 'yfit', 'error')
```



myFit.m

Interpolation

interp1 One-dimensional interpolation

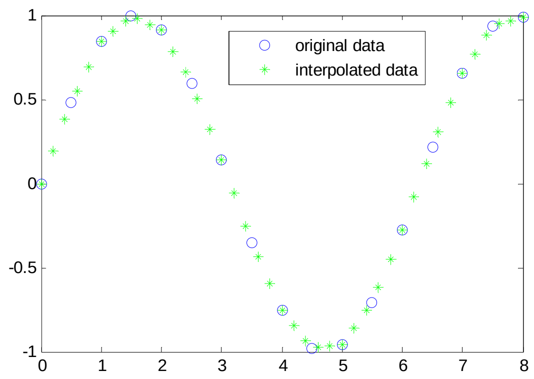
```
yi=interp1(x, y, xi, ‘method’)

Given (x,y) and interpolates to find yi corresponding to xi
      method:
'nearest'  - nearest neighbor interpolation
'linear'   - linear interpolation
‘spline'   - piecewise cubic spline interpolation (SPLINE)
'cubic'    - same as 'pchip'
```

myInterp.m

Advanced commands:
`interp2, interp3, spline, griddata`, etc

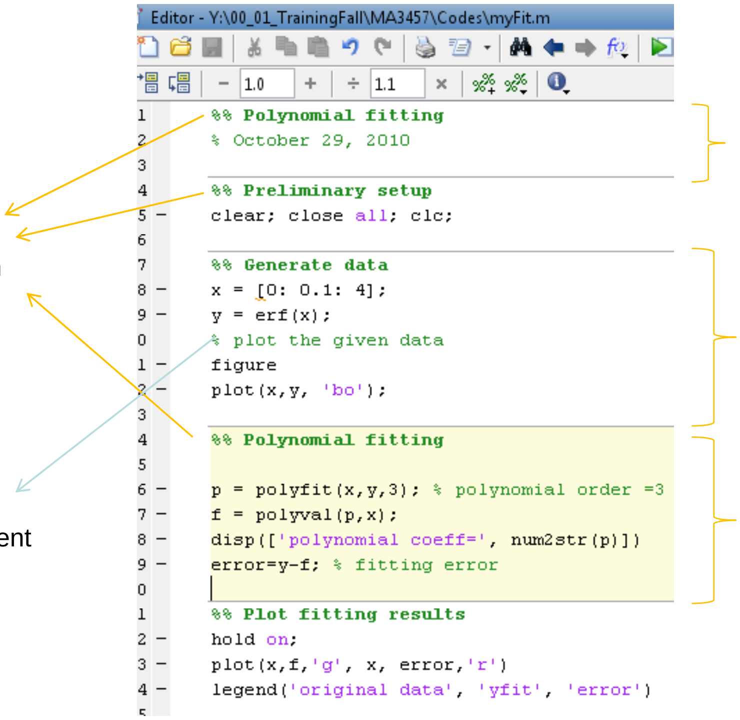
Interpolation: Example

```
%%% example for interpolation %
x=0:0.5:8;
y=sin(x);

xi=0:0.2:8;

yi=interp1(x,y,xi, 'linear');

plot(x, y, 'ob');
hold on;
plot(xi, yi, '*g');
legend('original data', 'data by interp');
```



myInterp.m

How to document a Matlab script



```
%% Polynomial fitting
% October 29, 2010

%% Preliminary setup
clear; close all; clc;

%% Generate data
x = [0: 0.1: 4];
y = erf(x);
% plot the given data
figure
plot(x,y, 'bo');

%% Polynomial fitting

p = polyfit(x,y,3); % polynomial order =3
f = polyval(p,x);
disp(['polynomial coeff=', num2str(p)])
error=y-f; % fitting error

%% Plot fitting results
hold on;
plot(x,f,'g', x, error,'r')
legend('original data', 'yfit', 'error')
```

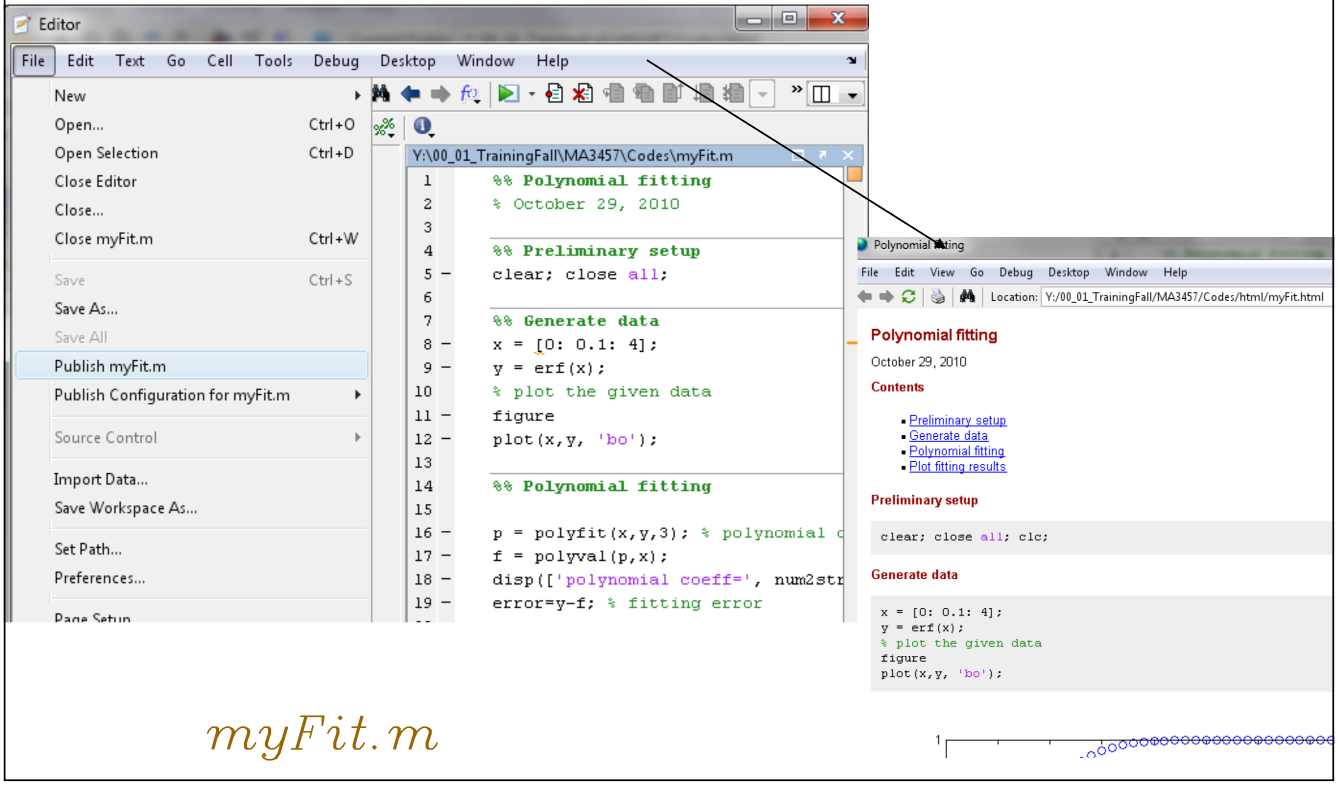
cell

Define the boundaries of the cells in a MATLAB script using %%

% → comment

Publishing a script to HTML

Editor window → File Menu→ Publish fileName



```
%% Polynomial fitting
% October 29, 2010

%% Preliminary setup
clear; close all;

%% Generate data
x = [0: 0.1: 4];
y = erf(x);
% plot the given data
figure
plot(x,y, 'bo');

%% Polynomial fitting

p = polyfit(x,y,3); % polynomial o
f = polyval(p,x);
disp(['polynomial coeff=', num2str
error=y-f; % fitting error
```

Polynomial fitting

October 29, 2010

Contents

- Preliminary setup
- Generate data
- Polynomial fitting
- Plot fitting results

Preliminary setup

```
clear; close all; clc;
```

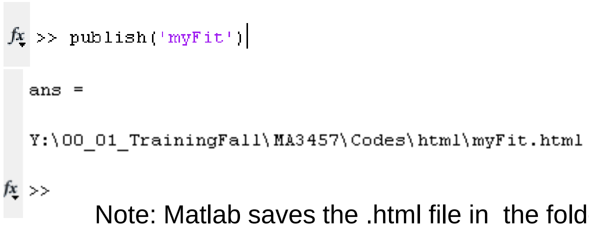
Generate data

```
x = [0: 0.1: 4];
y = erf(x);
% plot the given data
figure
plot(x,y, 'bo');
```

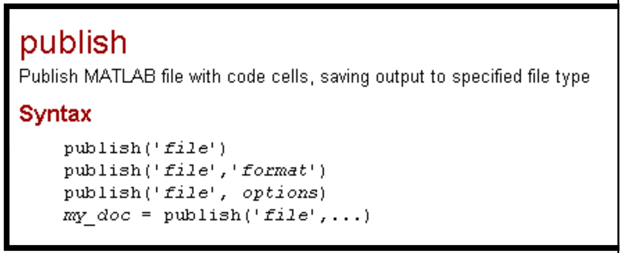
myFit.m

Publishing a script (in various formats)

1. Publish the script in **html format**



```
>> publish('myFit')

ans =

Y:\00_01_TrainingFall\MA3457\Codes\html\myFit.html

>>
```



publish

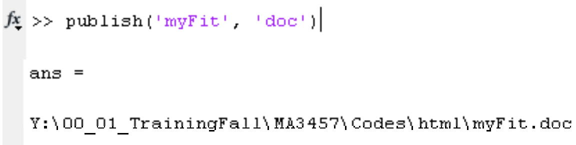
Publish MATLAB file with code cells, saving output to specified file type

Syntax

```
publish('file')
publish('file','format')
publish('file', options)
my_doc = publish('file',...)
```

Note: Matlab saves the .html file in the folder html

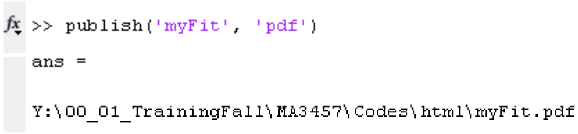
2. Publish the script in **doc format**



```
>> publish('myFit', 'doc')

ans =

Y:\00_01_TrainingFall\MA3457\Codes\html\myFit.doc
```

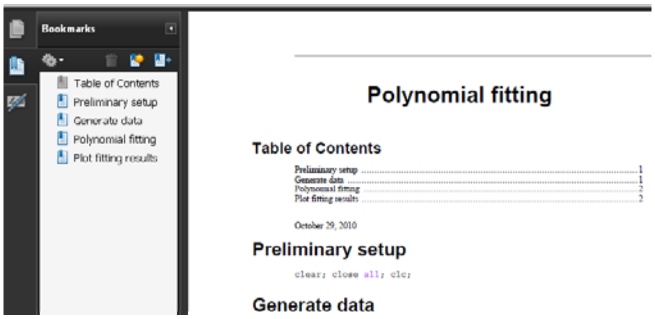
Note: Matlab saves the .doc file in the folder html

3. Publish the script in **pdf format**



```
>> publish('myFit', 'pdf')
ans =

Y:\00_01_TrainingFall\MA3457\Codes\html\myFit.pdf
```

myFit.pdf



Polynomial fitting

Table of Contents

Preliminary setup 1
Generate data 1
Polynomial fitting 2
Plot fitting results 2

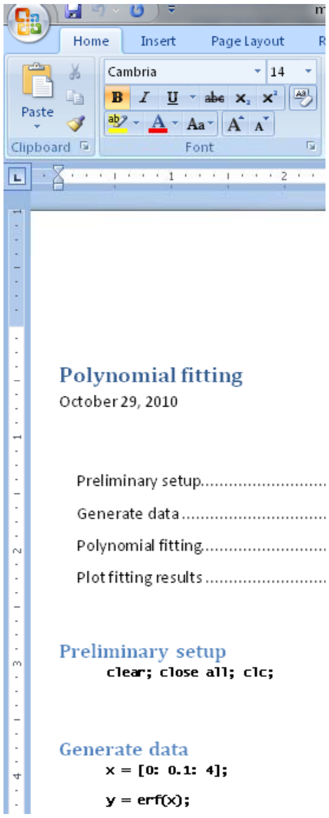
October 29, 2010

Preliminary setup

```
clear; close all; clc;
```

Generate data

myFit.doc



Polynomial fitting

October 29, 2010

Preliminary setup..........................
Generate data..............................
Polynomial fitting.........................
Plot fitting results.........................

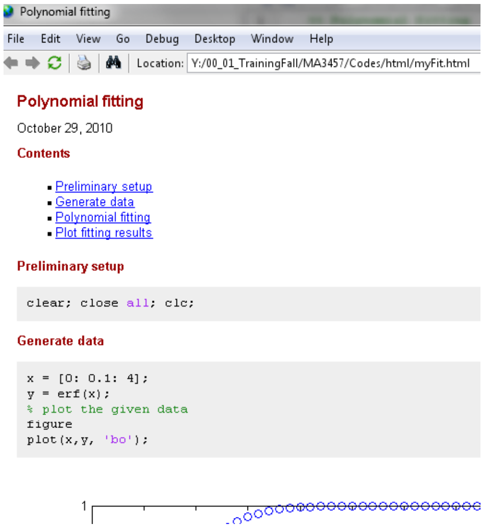
Preliminary setup

```
clear; close all; clc;
```

Generate data

```
x = [0: 0.1: 4];
y = erf(x);
```

myFit.html



Polynomial fitting
File
Edit
View
Go
Debug
Desktop
Window
Help
Location: Y:/00_01_TrainingFall/MA3457/Codes/html/myFit.html
1

Polynomial fitting

October 29, 2010

Contents

- Preliminary setup
- Generate data
- Polynomial fitting
- Plot fitting results

Preliminary setup

```
clear; close all; clc;
```

Generate data

```
x = [0: 0.1: 4];
y = erf(x);
% plot the given data
figure
plot(x,y, 'bo');
```

MATLAB Refresher Course

1.Variables, Operators
2.Matrices
3.Matlab Functions
4.Relational operators & Loops (Flow Control)
5.Scripts
6.User Defined Functions
7.Visualization
8.Publishing a script to HTML