Matlab Basics

S. Awad, Ph.D.
E.C.E. Department
University of Michigan

Invoking Matlab

- The program will give the following prompt:
  >>
- Now we can enter commands to be executed
- **>> a=1,b=2, c=a+b**

Matlab Punctuation

- **,** (comma) Separates commands on a line.
  Also used to separate elements in a matrix.
- **;** Separates commands on a line and inhibits the display of intermediate results.
  Also used to terminate a row in a matrix.
- **%** Denotes comment after this sign.
- **…** Denotes a continuation character
- **!** A system command follows this character
- **Note: Matlab is case sensitive**

Entering Data

```
>> a = 2;        % A scalar, 1 by  1 matrix
>> b = [ 1 2 3]; % A row vector
>> c = [5 6 7]'; % A column vector
```

Entering Matrices

2 ways to enter Matrices:

- Use ; to separate rows

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

- Type *Enter* to go to following line

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


Creating Complex Data

```
>> x = 2 + j*2; % A complex number
>> x = 2 + 2j;  % Another way
```

- **j** = **i** = $\sqrt{-1}$ are Matlab Constants

Entering a Complex Matrix

$$C = \begin{bmatrix} 2-j2 & 5 & -2 \\ 0 & 5-j7 & 10-j12 \\ 11 & 2+j5 & 1+j \end{bmatrix}$$

- Enter the real part of 3 x 3 Matrix
  - **`cr=[ 2, 5, -2;0, 5, 10;11, 2, 1];`**
- Enter Complex part
  - **`ci=[-2, 0,  0;0,-7,-12; 0, 5, 1];`**
- Add both components of the Matrix c
  - **`c=cr + j*ci;`**

Getting Help

- There are three ways of getting help in Matlab
  - **`help`** command
  - **`lookfor`** command
  - **`helpdesk`**

Getting Help: Help

- The **`help`** command displays information about the command:

```
>> help sum

 SUM Sum of elements.
    For vectors, SUM(X) is the sum of the elements of X. For
    matrices, SUM(X) is a row vector with the sum over each
    column. For N-D arrays, SUM(X) operates along the first
    non-singleton dimension.
   .
   .
   .
```


Getting Help: Lookfor

- The **`lookfor`** command searches all of the functions based on a keyword and lists all related commands.

```
>>lookfor spectrum
FFTSHIFT Shift DC component to center of spectrum.
PMEM   Power Spectrum estimate via MEM (Maximum PMTM
PMUSIC  Power Spectrum estimate via MUSIC eigenvector
SPECPLOT Plot the output of the SPECTRUM function.
SPECTRUM Power spectrum estimate of one or two
   .
   .
   .
```

Examining the Matlab Workspace

- The data and variables the user creates and/or imports in the command window is found in the so called Matlab workspace.
- To see what variable names the user has entered, use the **who** command:

```
» who

Your variables are:

B          a          b          c
```


Examining the Matlab Workspace (cont.)

- For more detailed information, use **whos** command

```
» whos
  Name        Size              Bytes  Class

  B           3x3                  72  double array
  a           1x1                   8  double array
  b           1x3                  24  double array
  c           3x1                  24  double array

Grand total is 16 elements using 128 byte
```

- Note: 8 bytes per element is used in a double

Removing Variables from the Workspace

- Use the **clear** command to remove variables

```
clear a b;          % Removes a & b
clear;              % Removes all variables
clear functions;    % Removes all compiled
                    % functions
```


Matlab Operations for Scalars

- Addition a+b
- Subtraction a-b
- Multiplication a*b
- Division $\frac{a}{b}$ = a/b

  $\frac{b}{a}$ = a\b
- Exponential a^b = a^b

Special Variables and Constants

```
ans;  % Default variable name used for results
pi;   % Pi ≡ π = 3.141592…
eps;  % Smallest computer number = 2.206x10^-16 = 2^-52
inf;  % Stands for ∞ (infinity)
i,j;  % √-1
nargin;  % Number of function input arguments
nargout; % Number of function output arguments
nan;     % 0/0 or ∞/∞
```


Saving Data

- Use the **save** command to store all variables in Matlab binary format in the file matlab.mat by typing:

```
>> save
```

- To save the variables in another file data.mat type:

```
>> save data
```

- To save only some variables type:

```
>> save a b c;          % Save in matlab.mat
>> save data a b c;     % Saves in data.mat
```

  - Note a, b, and c are user defined variables

- To append data to a file use:

```
>> save data d -append % Appends data.mat
```

Retrieving Saved Data

- Use the command **`load`** with the same syntax as the **`save`** command.

- To retrieve all data stored in matlab.mat
- `>> load;`
- To retrieve all data stored in data.mat
- `>> load data;`

Matrix Operations

- Transpose
- Addition
- Subtraction
- Multiplication
- Matrix Powers

Transpose of a Matrix

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

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad b = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Addition of Matrices

- For addition c(i,j) = a(i,j) + b(i,j)
- Matrices a, b, & c will have same dimensions

```
>> c = a + b
c =

     2     6    10
     6    10    14
    10    14    18
```

- Note: c = a + 1 means c(i,j) = a(i,j) + 1

Subtraction of Matrices

- For subtraction d(i,j) = a(i,j) - b(i,j)
- Matrices a, b, & d will have same dimensions

```
>> d = a - b
d =

     0    -2    -4
     2     0    -2
     4     2     0
```

- Note: d = a - 1 means d(i,j) = a(i,j) - 1

Multiplication of Matrices

- Suppose: **a** is n x m (n by m) & **b** is m x ℓ
- for **c = a * b**, c must be n x ℓ

$$c(i, j) = \sum_{k=1}^{m} a(i,k)b(k, j)$$

```
>> a = [ 1 1 1];
>> c = a * a';        % c = 1 + 1 + 1
c =
     3
```

Matrix Powers

- Assume that a is a square matrix, then
  - b = a^2 means b = a* a
  - c = a^5 = a*a*a*a*a
- In general
  - d = a^p = a*a*a*...*a (for p times)
- In case p is not an integer (i.e. p = 1/2)
  - e = a^0.5 means a = e*e
- Note a^-1 ≡ inv(a)

Element by Element Operations (Array Operations)

- Multiplication
- Division
- Power
- Functions

Element by Element Multiplication

- Assume a and b are matrices of the same dimensions
- **c = a.*b** will give the matrix c with the same dimensions as a & b with
- c(i,j) = a(i,j)*b(i,j)

```
>> a=[1 2 3];b=[4 5 6];
>> c=a.*b   % [ 1x4  2x5 3x6 ]

c =

     4    10    18
```


Element by Element Division

- Assume a and b are matrices of the same dimensions
- **d = a./b** will give the matrix d with the same dimensions as a & b with
- d(i,j) = a(i,j)/b(i,j)

```
>> a=[1 2 3];b=[4 5 6];
>> d=a./b   % [ 1/4  2/5 3/6 ]

d =

    0.2500    0.4000    0.5000
```

Element by Element Power

- Assume a and b are matrices of the same dimensions
- **e = a.^b** will give the matrix d with the same dimensions as a & b with
- e(i,j) = a(i,j)^b(i,j)

```
>> a=[1 2 3];b=[4 5 6];
>> e=a.^b   % [ 1^4  2^5 3^6 ]

e =

     1    32   729
```


Element by Element Power (cont)

- The exponent can be a scalar

```
f = a.^ 2   % [ 1^2  2^2  3^2 ]
f =
     1     4     9
```

- The base can be a scalar

```
m = 2.^ a   % [ 2^1 2^2 2^3 ]
m =
     2     4     8
```

Element by Element Functions

- In Matlab, **all functions** are element by element functions.
  - b = sin(a) means b(i,j)=sin(a(i,j))for all i & j
- There are a large set of functions such as
  - cos, tan, acos, exp, ...

Relational Operations

- These operations compare two matrices of equal dimensions. The following operations are used:
  - < Less than
  - <= Less than or Equal to
  - > Greater than
  - >= Greater than or Equal to
  - == Equal to
  - ~= Not Equal to

Relational Operations (cont)

- Comparison is done between pairs of corresponding elements. Results are placed in a matrix of ones & zeros.
  - If the comparison is TRUE put 1
  - If the comparison id FALSE put 0

```
» a=[1 5; -2 6]; b=[2 3;1 10];
» c=(a>b)

c =

     0     1
     0     0
```

$$a = \begin{bmatrix} 1 & 5 \\ -2 & 6 \end{bmatrix}$$

$$b = \begin{bmatrix} 2 & 3 \\ 1 & 10 \end{bmatrix}$$

$$c = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Array Comparisons

- The following functions do array comparisons

`isequal(b,a)`	True if both are identical
`intersect(b,a)`	Values in the intersection of a & b
`setdiff(a,b)`	Values from a that are not in b
`setxor(a,b)`	Values in the exclusive or of a & b
`union(a,b)`	Values in the union of a & b

Array Manipulation

- We need to insert, extract, replace, etc elements of a matrix. For example:

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

- To replace a(3,3) by a zero,

```
>> a(3,3) =0
a =
     1     2     3
     4     5     6
     7     8     0
```


Array Manipulation (cont.)

- To replace the entire first row by [10 10 10]

```
>> a(1,:) = [10 10 10]
```

All Columns

First Row

```
a =
    10    10    10
     4     5     6
     7     8     0
```

Array Manipulation (cont.)

- To replace the second first row by [20 20 20]'

```
>> a(:,2) = [20 20 20]'
```

Second Column

All Rows

```
a =
    10    20    10
     4    20     6
     7    20     0
```


Array Subscripting

- To subscript the array shown below

```
a =
    10    20    10
     4    20     6
     7    20     0
```

Array to Subscript

```
>> as1= a(2:3,2:3)
as1 =
    20     6
    20     0
```

Second to Third Column

Second to Third Row

- Note: The increment here is one

Constructing Arrays

- To construct a row array x=[0, 0.1, 0.2, … , 10]
- Instead of entering all of the elements use

```
x=[0:0.1:10]
```

Last Value
Increment Amount
First Value

- For an increment of one it is simply

```
y=[0:20] = [ 0, 1, 2, … , 20]
```

- Only the first, last, and increment are specified
- Note that the increment between data is constant here and we cannot control the number of points.

Constructing Arrays (cont.)

- Specify a **linearly** spaced array by **number of points**
- Increment will be constant but will be calculated instead of specified

```
x=linspace(-pi,pi,4)
```

Number of Values
Last Value
First Value

```
x =

   -3.1416   -1.0472    1.0472    3.1416
```

[- π - π/3 π/3 π]

- Increment = 2π /(4-1) = 2π /3

Constructing Arrays (cont.)

- To specify a **logarithmically** spaced array by **number of points**

```
x=logspace(0, 2, 11)
```

Number of Values

Last Exponent

First Exponent

```
x =
```

```
   1.0000   1.5849   2.5119  ...  100.0000
```

10^0 $10^{0.2}$ $10^{0.4}$ ••• 10^2

This is equivalent to

Number of Points -1

```
x=10.^[0:2/(11-1):2]
```


Addressing Row and Column Arrays

- Assume the Row vector x

```
» x=[ 3 5 -1 20];
```

- The first element in the array x(1) = 3
- To step through the array

```
» x(1:2:end)
```

Up to end of array

Step Size (Increment)

```
ans =
     3    -1
```

[x(1) x(3)]

Addressing Row and Column Arrays (cont.)

```
» a=[-5 1 20 30 -15 4]'
» a(2:end)
```

```
ans =
     1
    20
    30
   -15
     4
```

a(2)
a(3)
a(4)
a(5)
a(6)

```
» a([3 5 1])
```

```
ans =
    20
   -15
    -5
```

a(3)
a(5)
a(1)

Addressing Row and Column Arrays (cont.)

- To find the number of elements in a vector, use the **`length`** command

```
» l1=length(a)
l1 =
     6

» l2=length(a([3 5 1]))
l2 =
     3
```

Find Command

- **Problem**: How many elements in a are ≥ to 1?
- First find the indices of all numbers ≥ 1

```
» ind = find(a>=1)    % Will return column vector
                      % ind = [2 3 4 6]'
```

- To verify the results, find these numbers

```
» b = a(ind)          % b = [1 20 30 4]'
```

- The number of elements in a ≥ 1 is the length of vector **ind**.

```
» num = length(ind)
num =
     4
```

- Thus we have four elements which satisfy the required condition

Special Matrices

- Matrices with all elements = 0

```
» a = zeros(2,1)      % Creates a 2x1 matrix
a =
     0
     0
» b = zeros(2)        % Creates a 2x2 matrix
b =
     0     0
     0     0
```

- Matrices with all elements = 1

```
» c = ones(1,5)       % Creates a 1x5 matrix
c =
     1     1     1     1     1
```

Special Matrices (cont.)

```
» d = eye(3)
d =
     1     0     0
     0     1     0
     0     0     1
```

3x3 Identity Matrix

```
» e = diag([3 -1 10])
e =
     3     0     0
     0    -1     0
     0     0    10
```

3x3 matrix with Diagonals specified and zeros elsewhere

Random Number matrices can also be created. (To be shown later)

Two-Dimensional Plots

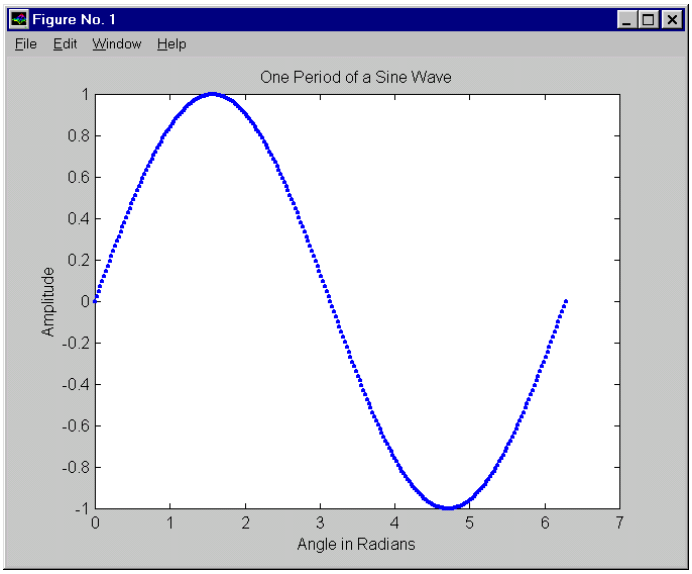
```
» theta=[0:pi/128:2*pi]';
» y=sin(theta);
» plot(theta,y,'b.');
» xlabel('Angle in Radians');
» ylabel('Amplitude');
» title('One Period of a Sine Wave');
```

Marker: Point

Color: Blue

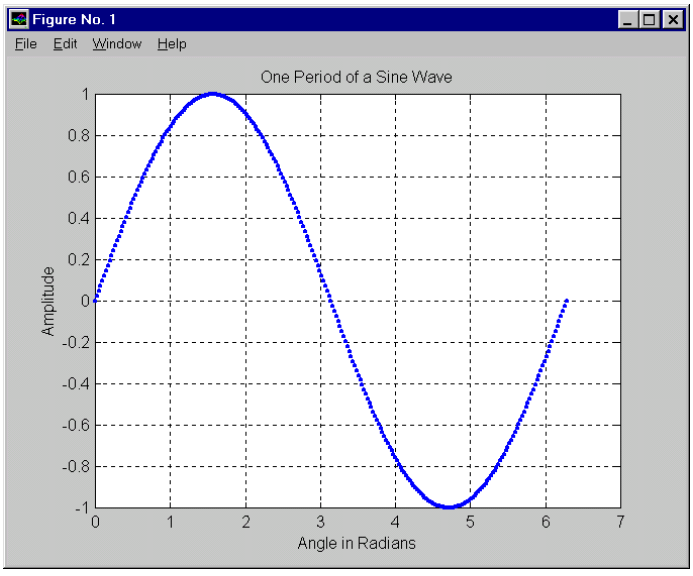
Writes Title of Graph

2-D Plots (cont.)



2-D Grids

- To Enable Grid
  » grid on
- To Disable Grid
  » grid off



Stem Command

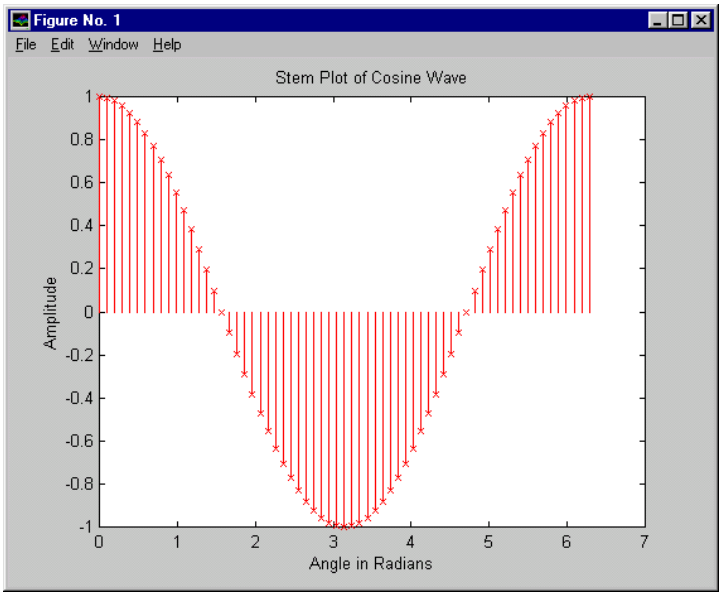
```
» theta=[0:pi/32:2*pi]';
» y=cos(theta);
» stem(theta,y,'rx');
» xlabel('Angle in Radians');
» ylabel('Amplitude');
» title('Stem Plot of Cosine Wave');
```

Marker: x

Color: Red

Writes Title of Graph

Stem Command (cont.)



Clearing a Figure

- In case a figure already exists, the plot command will clear the current figure window and draws a new plot.

- To clear a figure use the command

```
» clf
```


Multiple Plots

- Consider the following vector pairs:

```
» x1=[1 2 3 4]';      y1=[4 4.1 4.2 4.3]';
```

(x1 and x2 have same length)

```
» x2=[1 2 3 4 5]';    y2=[5 5.1 5.2 5.3 5.4]';
```

(x2 & y2 have same length but may be different from x1 & y1)

```
» x3=[1 2 3 4 5 6]'; y3=[6 6.1 6.2 6.3 6.4 6.5]';
```

(x3 & y3 have same length but may be different from x1 & y1...)

- To plot the 3 plots on the same graph

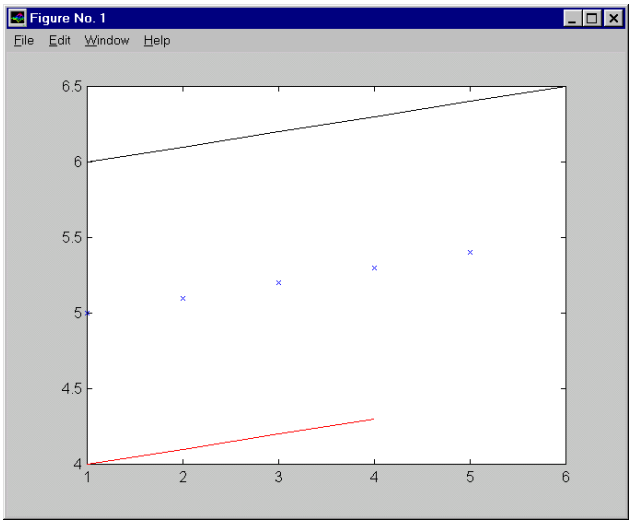
```
plot(x1,y1,'r',x2,y2,'xb',x3,y3,'k');
```

- xlabel, ylabel, & title can also be put on the graph

Multiple Plots (cont.)

```
plot(x1,y1,'r',x2,y2,'xb',x3,y3,'k');
```

r = *color:* red
xb = *marker:* x
color: blue
k = *color:* black



Multiple Plots (cont.)

- If x1, x2, & x3 are the same vector x:

```
» x =[1   2   3   4 ];
```

- And there are y1, y2, & y3 of same size:

```
» y1=[1   2   3   4 ];
» y2=[1.2 2.2 3.2 4.2];
» y3=[1.4 2.4 3.4 4.4];
```

- Then to plot the 3 plots at the same time

```
» y = [ y1; y2; y3] % 3 columns
» plot( x,y );
```

Legend Box

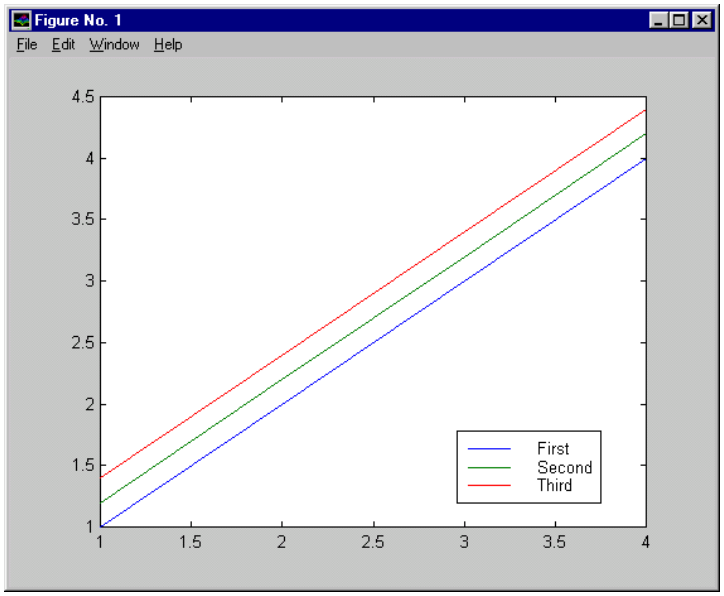
- To create a legend box in the upper right corner of the plot for the previous example:

```
» legend('First', 'Second', 'Third' );
```

- To **move** the legend, **click and hold** the left mouse button near the edge of the legend and **drag** it to the required place.
- To delete the legend, use

```
» legend off
```


Legend Box (cont.)

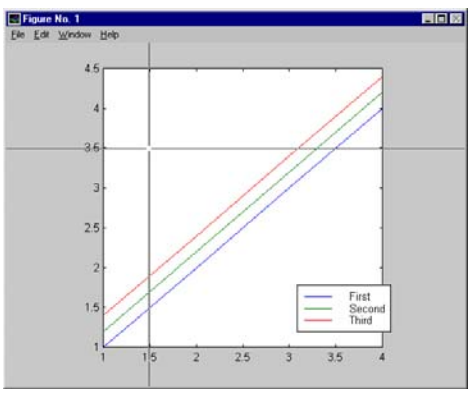


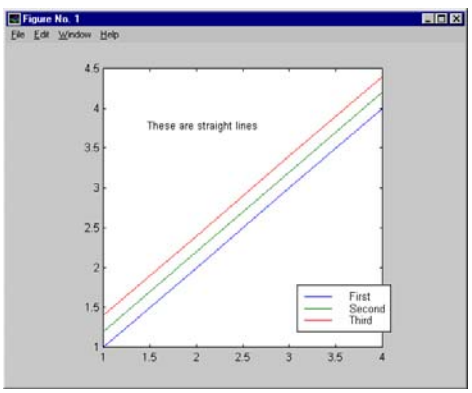
Plot Axis Customization

- To set each axis minimums and maximums
  » axis([xmin xmax ymin ymax]);
- To return to default settings
  » axis('auto') ***or*** **» axis auto**
- Set scaling factors for both axes to be equal
  » axis equal
- To keep MATLAB from altering the proportions of the axes if the view is changed
  » axis vis3d

Adding Text to a Graph

- The puts cross-hair that follows the mouse and waits for a click at the desired location
  » gtext('These are straight lines');

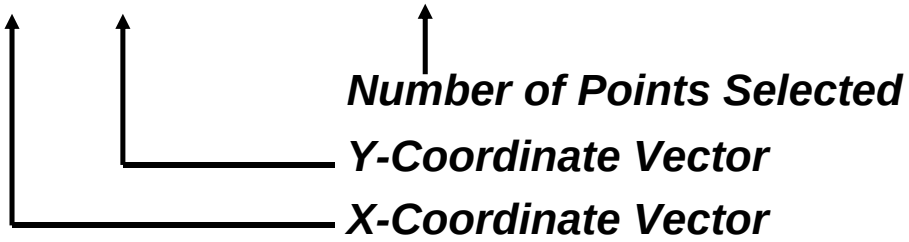




Getting Data from a Graph

- Suppose there is a current plot on the screen and we want to select some points from the graph and return the x & y coordinates.
- To do this use the ginput command.

```
» [xp, yp] = ginput(3);
```



Polynomials

- In Matlab, a polynomial is represented by a **row** vector of it's coefficients in **descending** order.
- For example, the polynomial

$$p = x^4 - 12x^3 + 25x + 116$$

- Is entered as:

```
» p=[1 -12 0 25 116]
```

- Zero term coefficients MUST be included

Roots

- To find the roots of a polynomial use **`roots`**
- Roots are always **column vectors**

```
» p=[1 -12 0 25 116];
» r=roots(p)
r =
   11.7473
    2.7028
   -1.2251 + 1.4672i
   -1.2251 - 1.4672i
```


Multiplication

- Use the conv (convolution) command to evaluate the product of two polynomials:

$$a = \left(x^3 + 2x^2 + 3x + 4\right)$$
$$b = \left(x^3 + 4x^2 + 9x + 16\right)$$

```
» a=[1 2 3 4];b=[1 4 9 16];
» c=conv(a,b)
c =
     1     6    20    50    75    84    64
```

$$\left(x^6 + 6x^5 + 20x^4 + 50x^3 + 75x^2 + 84x + 64\right)$$

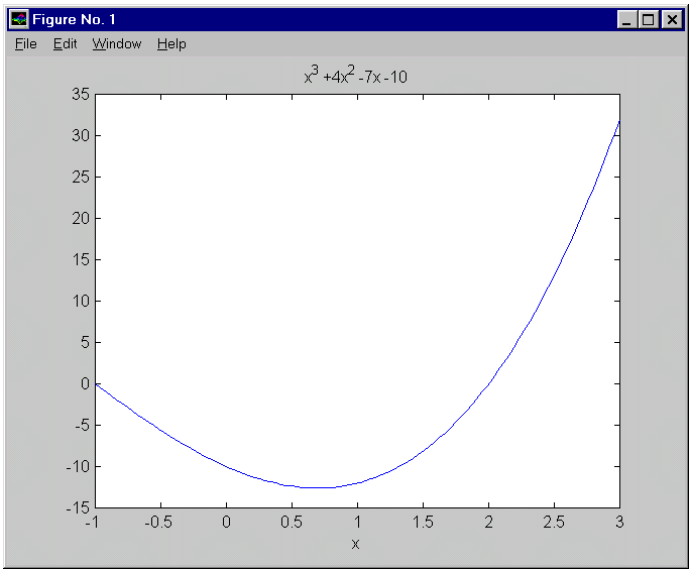
Evaluation

- Evaluate the polynomial $p(x)$ between -1 and 3

$$p(x) = x^3 + 4x^2 - 7x - 10$$

```
» x=linspace(-1,3);  % 100 data points
» p=[1 4 -7 -10];    % Coefficients
» v=polyval(p,x);    % Evaluate
» plot(x,v);
» title('x^3 +4x^2 -7x -10');
» xlabel('x');
```


Evaluation (cont.)



Control Flow Commands

- For Loop
- While Loop
- If-Elseif-Else-End
- Switch
- Other Useful Functions

For Loop

- Format:

```
for v = expression (or array)
   -    commands
end
```

- Example to Create the Vector x= [1^2 2^2 3^2 4^2]'

```
» x = zeros(4,1);
» for i=1:4
     x(i) = i*i;
  end;
```

Nested For Loops

- Create an m x n Hilbert Matrix

$$a = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & \cdots & j & \cdots & n \end{array} \\ \begin{bmatrix} 1 & 0.5 & & & & \\ 0.5 & 0.\overline{33} & & & & \\ & & & & & \\ & & & \frac{1}{i+j-1} & & \\ & & & & & \\ & & & & & \frac{1}{m+n-1} \end{bmatrix} \end{array}$$

$$a(i,j) = \frac{1}{i+j-1}$$

Nested For Loops (cont.)

```
» m = 4;n = 5;
» a = zeros(m,n);
» for i=1:m,
      for j=1:n
         a(i,j)=1/(i+j-1);
      end
   end
» a
  a =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
```

For Loop Array Example

- Matlab Code

```
» data = [3 9  4  5;
          7 16 -1 5];
» for n = data
      x = n(1) - n(2)
  end
```

- Results

```
x =
   -4
x =
   -7
x =
    5
x =
    0
```

⇒ ***First Column in data***

⇒ ***Second Column in data***

⇒ ***Third Column in data***

⇒ ***Fourth Column in data***

While Loops

`while` ***expresion***

-
- ***commands***
-

`end`

While Loop Example 1

- The MacLaurin series expansion for log(1+x) (natural log) where |x| < 1, is given by:

$$\log(1+x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k}$$

- Estimate **log(1+x)** for **x=0.5** by summing the series until the term to be added next is less than **eps** in absolute value

While Loop Ex. 1 (cont.)

```
» v=0;x=0.5;k=1;
» while abs((x^k)/k) >= eps
  v = v+(-1)^(k+1)*((x^k)/k);
  k = k+1;
  end
» v,k
v =
    0.4055
k =
    47
```

log(1.5)

Number of Iterations

While Loop Example 2

- What is the first integer n for which n! (n factorial) is a 100 digit number?

```
» n = 1;
» while prod(1:n) < 1.e100
      n = n + 1;
  end
» n
n =
    70
```


While Loop Ex. 2 (cont.)

- Verify the results

```
» prod(1:n)      % n! = 70!
ans =
  1.1979e+100
» prod(1:n-1)    %(n-1)! = 69!
ans =
  1.7112e+098
```

- Therefore n=70 is the first integer where n! is a 100 digit number

If - End Constructions

- For one alternative expressions:

if *expression*
- ◆
- ◆ Commands
- ◆

end

- Example:

```
» if a > 1
    b=0;
    c=5;
  end
```


If-Else-End Constructions

- For two alternative expressions:

if *expression*
- ◆ Command Set 1
- ◆

else *expression*
- ◆ Command Set 2
- ◆

end

- Example:

```
» if a > 0
    b=1;
  else
    b=-1;
  end
```

If-Elseif-Else-End Constructions

- For several alternative expressions:

if *expression*

- Command Set 1

elseif *expression*

- Command Set 2

elseif *expression*

- Command Set 3

else *expression*

- Command Set 4

end

Breaking Out of a Loop

- It is possible to break out of for loops & while loops using the **break** command.
- When the **break** statement is executed, MATLAB jumps from the current loop in which it appears.
- In the case of nested for or while loops, MATLAB jumps out of only the current loop.

Break Example

- Find the value of eps

```
» EPS=1;
» for i=1:1000
        EPS=EPS/2;
        if(1+EPS)<=1
            EPS=EPS*2;
            break;
        end
  end
» EPS
EPS =
  2.2204e-016
```

NOTE: i=53