Postgraduate Electronics and Communications

كلية الهندسة بشبرا

Shoubra faculty of Engineering

Computer Aided Design (CAD)

Lecture 2

- Scalar variables, Complex Numbers.
- Vectors in Matlab

Dr. Sawsan Abdellatif

MATLAB®

The Language of Technical Computing

Reference:

Matlab by Example: Programming Basics, Munther Gdeisat

Chapter 2:
Scalars in Matlab

Scalars in Matlab

- In Matlab, every variable created should have a value.
- Variables are created either by Matlab or by the user.
- Variables created by Matlab are considered to be special variables, whose values are assigned by Matlab.

Matlab Special variables

```
>>pi
```

Then press Enter. Matlab responds with

```
ans =
      3.1416
```

This command generates another special variable "ans" and assigns the value 3.1416 to it.
The special variable "ans" saves the result of any Matlab operation if the value of the result is not specifically assigned to a variable.

Matlab Special variables

- Other examples of special variables are i and j. The value for both variables is defined as $\sqrt{-1}$.

```
>>i
```

Then press **Enter**. Matlab responds with

```
>>ans
        0 + 1.0000i
```

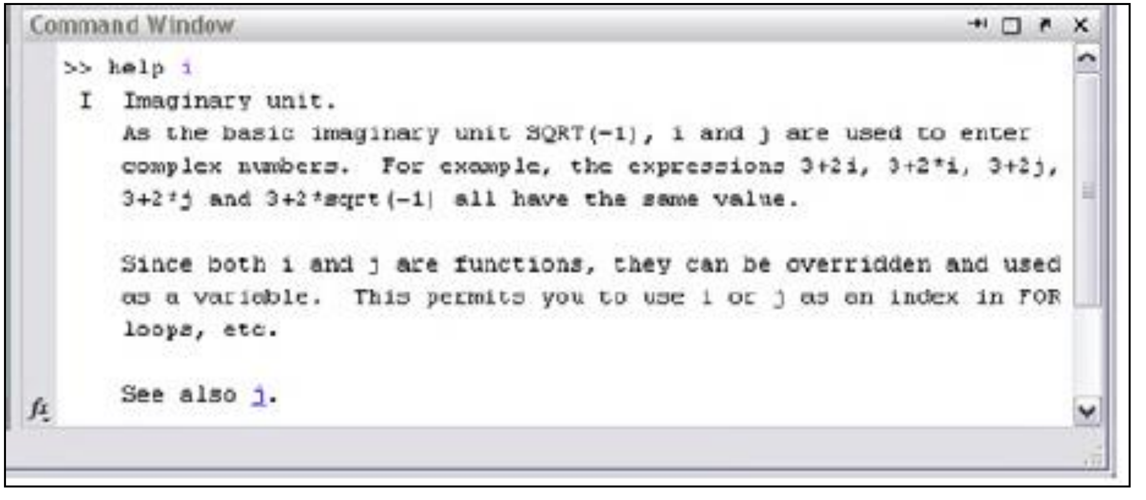
```
>>j
```

Then press **Enter**. Matlab responds with

```
>>ans
        0 + 1.0000i
```

- To get some help about the variable i

```
>>help i
```



Matlab Special variables (cont'd)

Changing the values of Matlab Special variables

```
>>pi=1
```

Then press **Enter**. Matlab responds with

```
pi =
        1
```

Now the value for the special variable `pi` has been changed to 1.

- To restore the value of the special variable pi,

```
>>clear pi
```

To display the value of the variable `pi`, type at the **Command Prompt**

```
>>pi
```

Then press **Enter**. Matlab responds with

```
ans =
        3.1416
```

User-Defined variables

Naming user defined-variables:

- A variable name must not contain spaces or hyphens (-).
- A variable name can contain up to 63 characters.
- A variable name must start with a letter (a–z or A–Z), followed by any number of letters, digits (0–9), or underscores (_).
- Punctuation characters such commas (,) or apostrophes (') are not allowed, because many of them have special meanings in Matlab.
- A variable name must not be a **Script M-file** name or an existing **Matlab function** name.
- The use of a **Matlab reserved word** as a variable name is not allowed.

➢ Matlab is **Case sensitive**

Clearing user defined-variables:

```
>>clear y
```

Approximating Numbers

- Matlab supports four functions to approximate real numbers:

  round, fix, ceil, floor

"round" Function

- This function rounds a real number upward, or downward, toward the nearest integer.

```
>> x = round(2.51)
x =
     3
```

```
>> y = round(2.49)
y =
     2
```

"fix" Function

- This function truncates (eliminates) the decimal part of a real number, leaving the integer part unchanged.

```
>> x = fix(2.51)
x =
     2
```

```
>> y = fix(-2.51)
y =
    -2
```

Approximating Numbers

"ceil" Function

- Rounds up a real number toward the nearest higher integer

```
>> x = ceil(2.51)

   x =
        3
```

```
>> y = ceil(2.49)

   y =
        3
```

"floor" Function

- Rounds up a real number toward the nearest lower integer

```
>> x = floor(2.51)

   x =
        2
```

```
>> y = floor(2.49)

   y =
        2
```

Difference between "fix" and "floor" Approximation Functions

a	fix(a)	floor(a)
– 2.5	–2	–3
–1.75	–1	–2
–1.25	–1	–2
–0.5	0	–1
0.5	0	0
1.25	1	1
1.75	1	1
2.5	2	2

- "fix" and "floor" functions give similar results for positive numbers.
- **But**, they give different results for negative numbers

Mathematical Expressions for Scalar Variables

Precedence of Mathematical Operations

- Matlab evaluates mathematical expressions from left to right.
- Mathematical expressions may contain addition, subtraction, multiplication, division, and exponential mathematical operations as well as parentheses.
- These mathematical operations are evaluated in the following order in Matlab:

I. Parentheses, by starting with the innermost set and proceeding outward
II. The exponentiation operation
III. Multiplication and division
IV. Addition and subtraction.

Mathematical Expressions for Scalar Variables

From Mathematical Expressions to Matlab Expressions

Example 1: $r = \frac{x+y}{z}$ → `>> r = (x + y)/z ;`

- The addition operation needs to be evaluated first followed by the division.
- Since the division operation has a higher priority in Matlab than the addition operation, parentheses are needed to alter this priority order to give the addition operation a higher priority than that of the division operation.

Example 2: $r = x + \frac{y}{z}$ → `>> r = x + y / z ;`

Example 3: Write a Matlab program to evaluate r using the minimum number of parentheses

$$r = \frac{\dfrac{x}{z^3 + y^4} + \dfrac{x^3 + y^3}{x^2}}{\dfrac{x^2 + 1}{y^3} - 3}$$

→ `>> r = (x/(z^3 + y^4) + (x^3 + y^3)/x^2)/((x^2 + 1)/y^3 − 3);`

Relational and Logical Operations for Scalar Variables

The logic Class

- Any variable with a logical class has a value of either true or false.
- Matlab represents true as 1, and false as 0.

```
>>r=true
```

Matlab responds with

```
r=
     1
```

To check the class of r, type at the **Command Prompt**

```
>>whos r
```

Matlab responds with

```
Name  Size  Bytes  Class    Attributes
r     1×1   1      logical
```

Relational and Logical Operations for Scalar Variables

The Relational operators

- Relational operators require two operands, and they compare two values.
- The relational operators produce variables with a logical class.

Matlab has six relational operators which are

1. Greater than ">"
2. Less than "<"
3. Greater than or equal ">="
4. Less than or equal "<="
5. Equal "=="
6. Not equal "~="

Example:

```
>>x = 1;
>>y = 2;
>>a = x > y
```

Matlab responds and displays the value of a as

```
a =
     0
```

Relational and Logical Operations for Scalar Variables

The Logical operators

- Matlab has three logical operators which are:

1. AND "&"
2. OR "|"
3. NOT "~"

- The logical operators produce variables with the logical class.

AND "&" Logical Operator

Operand 1	Operand 2	&
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

```
x = 1;
y = 2;
g = x&y    →    g =
                    1
```

OR "|" Logical Operator

x	y	x | y
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

```
x = 1;
y = 2;
n = x|y    →    n =
                    1
```

NOT "~" Logical Operator

```
x = 0;
z = ~x    → result →    z =
                            1
x = 1;
w = ~x    →    w =
                   0
x = -1;
y = ~x    →    y =
                   0
```

Relational and Logical Operations for Scalar Variables

Combining Logical and Relational operators

- Logical and rational operators can be combined. For example:

```
x = 1;
y = 2;
n = (x < 3)&(y < 0)
```

Matlab responds with

```
n =
    0
```

Complex Scalar Variables

Creating Complex Scalar Variables

```
>> z = 1 + 2i
```

Matlab responds as follows:

```
z =
   1.0000 + 2.0000i
```

You can use j instead of i to represent $\sqrt{-1}$. For example,

```
>> z = 1 + 2j;
```

A third method to create a complex number is

```
>> z = 1 + 2*i;
```

or

```
>> z = 1 + 2*j;
```

Note

```
>> i = 1;
>> z = 1 + 2*i
```

→

```
z =
   3
```

Be careful not to use i and j as variable names. This may cause unexpected errors in the use of complex numbers.

Complex Scalar Variables

Conjugate of a Complex Number

```
>> z = 1 + 2i;
>> z1 = conj(z)
```

→ Matlab responds as follows:

```
z1 =
     1.0000 - 2.0000i
```

Modulus and Angle of a Complex Number

```
>> z = 3 + 4i;
>> a = abs(z)
  a =
      5
```

```
>> b = angle(z)
  b =
      0.9273
```

- Note that the angle is given here in radians. To convert the angle from radians to degrees, multiply it by $180/\pi$.

```
>> angle_in_degrees = angle(z)*180/pi
angle_in_degrees =
  53.1301
```

Chapter 3:
Vectors in Matlab

Vectors in Matlab

Transpose operation

- Applying the transpose operation to vectors changes a row vector to a column vector and vice versa.

```
>> x = [2, 3, 5];
>> x = x';
```

→

```
x =
     2
     3
     5
```

Determining the Number of Elements in a Vector

```
>>x  =  [2,3,5];
>>n  =  length(x)
```

→

```
n =
   3
```

Converting a Vector to a Column Vector

```
>> y=[1,2,3,4,5];
>> y = y(:)
```

→

```
y =
     1
     2
     3
     4
     5
```

The Matlab colon operator, " : ", can be used to convert a vector to a column vector.

Creating Vectors Using Linear Method

- The linear method can be used to create a row vector that has linearly spaced elements, that is, the difference between two successive elements in the vector is constant.

```
>> x = 0:2:10          →    x =
                                 0   2   4   6   8   10

>> y = 10:-2:0         →    y =
                                 10   8   6   4   2   0
```

Creating Vectors Using the Linear Spacing Method

The Matlab function `linspace(x1,x2,N)` can be used to create a row vector.

- `x1` is the start value.
- `x2` is the final value.
- `N` is the number of elements in a vector.

```
>> x = linspace(0, 10, 6)    →    x =
                                       0   2   4   6   8   10
```

Vector Concatenation

- ➢ Two vectors can be concatenated and become a single vector.

```
>> x1 = [1,2,3];
>> x2 = [4,5,6];
>> x  = [x1,x2]
```

→

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

```
x1 = [1,2,3];
x2 = [4,5,6];
x  = [x1;x2]
```

→

```
x =

     1     2     3
     4     5     6
```

```
s1 = [1;2;3];
s2 = [4;5;6];
s3 = [s2,s1]
```

→

```
s3 =

     4     1
     5     2
     6     3
```

Transpose Operation for Complex Vectors

- Applying the transpose operation to a complex vector not only changes rows to columns and vice versa, but also conjugates the vector's elements

```
>> x = [2+i,3-2i,5+3i]
>> z = x';
```

→

```
z =
     2.0000 - 1.0000i
     3.0000 + 2.0000i
     5.0000 - 3.0000i
```

- To change rows to columns and columns to rows only without conjugating the vector y elements, the command y.' can be applied

```
>> y=[4-3i;9+4i;7-5i;12+11i]
```

→

```
y =

   4.0000 - 3.0000i
   9.0000 + 4.0000i
   7.0000 - 5.0000i
  12.0000 +11.0000i
```

```
>> z=y.'

z =

   4.0000 - 3.0000i   9.0000 + 4.0000i   7.0000 - 5.0000i  12.0000 +11.0000i
```

The Relational Operations on Vectors

```
>>x=[2,4,7,9,-1,2];
>>y=[-1,4,8,1,-4,6];
>>z=x>y
z=
     1   0   0   1   1   0
```

(x>y) command determines whether the value of each element in the vector x is greater than the corresponding element in the vector y. The result is saved in z vector.

The Logical Operations on Vectors

```
x=[0,4,7,0,-1,2];
y=[1,2,8,0,-4,6];
z=x&y

z=
    0   1   1   0   1   1
```

- Remember: An input to relational and logical operators is considered to be true if it has a nonzero value.

Accessing Elements in Vectors

Accessing an **Individual** Element in a Vector Using its Index

create the vector `x = [3, 6, 9, 12, 15, 18]`

Index	1	2	3	4	5	6
Value	3	6	9	12	15	18

- To access the third element in the vector,

```
>> r = x(3)      →      r =
                              9
```

- To access the last element in the vector,

```
>> s = x(end);   →      s =
                              18
```

- Try to access the 7th element in vector x:

```
>> x(7)
```

Matlab responds with the error message

```
??? Index exceeds matrix dimensions.
```

Accessing Elements in Vectors (cont'd)

Accessing a **Group** of Elements in a Vector Using Their Indices

```
y =
      2    5    8    11    14    17
```

- To access the first three elements of the vector y

```
>> a = y(1:3);
```

- To access the last three element of the vector y

```
>> b = y(end - 2:end);
```

- To access the 2nd , 3rd , and the 4th elements of the vector y

```
>> c = y(2:4);
```

or

```
>> c = y([2,3,4]);
```

- To access the 2nd , 4th , and the 6th elements of the vector y

```
>> d = y([2,4,6]);
```

Accessing Elements in Vectors (cont'd)

Accessing Elements in a Vector Using Their Values

```
>>y=[2,3,5,5,7,10,12];
```

- To find the indices of the elements whose values are equal to 5,

```
>>a=find(y==5)    →    a =
                          3   4
```

"==" sign checks whether the values of the elements in the vector y are equal to 5.

- To find the indices of the elements in the vector y whose values are greater than 7,

```
>>b=find(y>7)    →    b =
                          6   7
```

- To find the indices of the elements in the vector y whose values are less than or equal to 9,

```
>>c=find(y<=9)    →    c =
                          1   2   3   4   5
```

- To find the values of the elements in y whose values are less than or equal to 9,

```
>>d=y(c)    →    d =
                     2   3   5   5   7
```

Accessing Elements in Vectors (cont'd)

Accessing Elements in a Vector Using the Relational and Logical Operators

- Matlab has an interesting way of using the relational and logical operations to access elements in vectors.

```
x = [0,4,7,0,-1,2];
y = [1,3,8,0,-4,6];
x > 3                    →    ans =
                                    0   1   1   0   0   0

>> r = y(x > 3)          →    r =
                                 3   8
```

- The command y(x>3) here outputs elements of y that correspond to the same positions where x is greater than 3.

Arithmetic Operations on Vectors

Vectors addition and subtraction:

- The addition and subtraction of two vectors is performed on an element-by-element basis.
- The vectors must be of equal dimensions.

```
x = [1,2,3];
y = [4,5,6];
```

```
>> z = x + y

z =
     5   7   9
```

```
>> s = x - y

s =
    -3   -3   -3
```

Adding a Number to a Vector

```
x = [1,2,3];
s = x + 10

s =
    11   12   13
```

add value to all elements in the vector x

Subtracting a Number from a Vector

```
x = [1,2,3];
t = x - 2

t =
    -1   0   1
```

Sub value from all elements in the vector x

Arithmetic Operations on Vectors (cont'd)

Vector Multiplication

Element-by-element multiplication

```
x = [1,2,3];
y = [4,5,6];

>> z = x.*y

z =
     4    10    18
```

Matrix-based multiplication

```
x = [1,2,3];
y = [4;5;6];

>> z = x*y

z =
    32
```

- Note matrix multiplication condition: Number of columns of the 1st vector (x) must equal number of rows of the 2nd vector (y)

Matrix Multiplication for Vectors

Case1: The 1st vector is **Row** vector and the 2nd vector is **Column** vector

```
>> z = x*y
```

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & \dots & x_n \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \end{bmatrix} = z = x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4 + \dots + x_ny_n$$

- The multiplication process here produces a single value scalar number.

Case2: The 1st vector is **Column** vector and the 2nd vector is **Row** vector

```
>> y*x
```

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \end{bmatrix} \times \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & \dots & x_n \end{bmatrix} =$$

y_1x_1	y_1x_2	y_1x_3	y_1x_4	.....	y_1x_n
y_2x_1	y_2x_2	y_2x_3	y_2x_4	.....	y_2x_n
y_3x_1	y_3x_2	y_3x_3	y_3x_4	.....	y_3x_n
y_4x_1	y_4x_2	y_4x_3	y_4x_4	.....	y_4x_n
.....	.....	.....	.....	.....	.....
y_nx_1	y_nx_2	y_nx_3	y_nx_4	.....	y_nx_n

- The multiplication process here produces a square matrix.

Remember: It is important to realize that in matrix multiplication $\mathbf{xy} \neq \mathbf{yx}$.

Arithmetic Operations on Vectors (cont'd)

Vector Division

Element-by-element Division

```
x = [1,2,3];
y = [4,5,6];

>> z = x./y

z =
    0.2500    0.4000    0.5000
```

Matrix-based Division

- Matrix division of vectors does not have any mathematical meaning.

Plotting Vectors

If we need to plot the function y = x^2, where x is in the range [- 3, 3].

```
>> x = -3:1:3
```

→

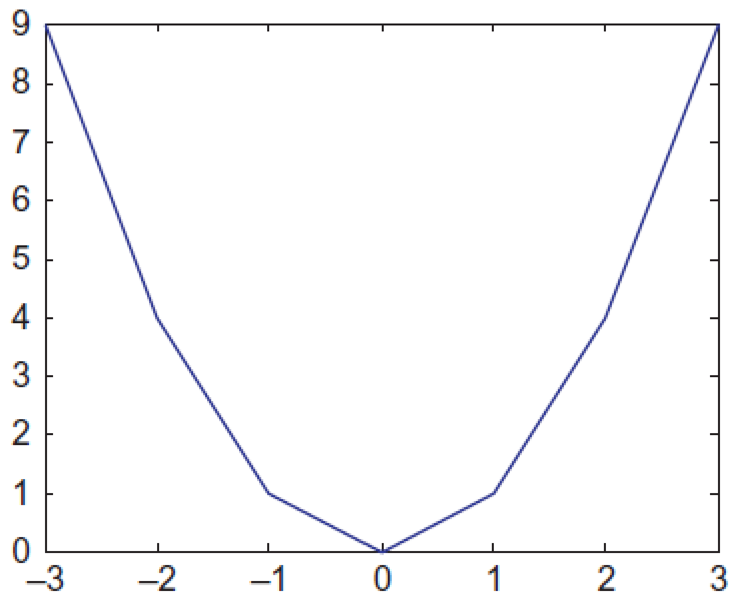
```
x =
    -3   -2   -1    0    1    2    3
```

```
>> y = x.^2
```

→

```
y =
     9    4    1    0    1    4    9
```

```
>> plot(x,y)
```



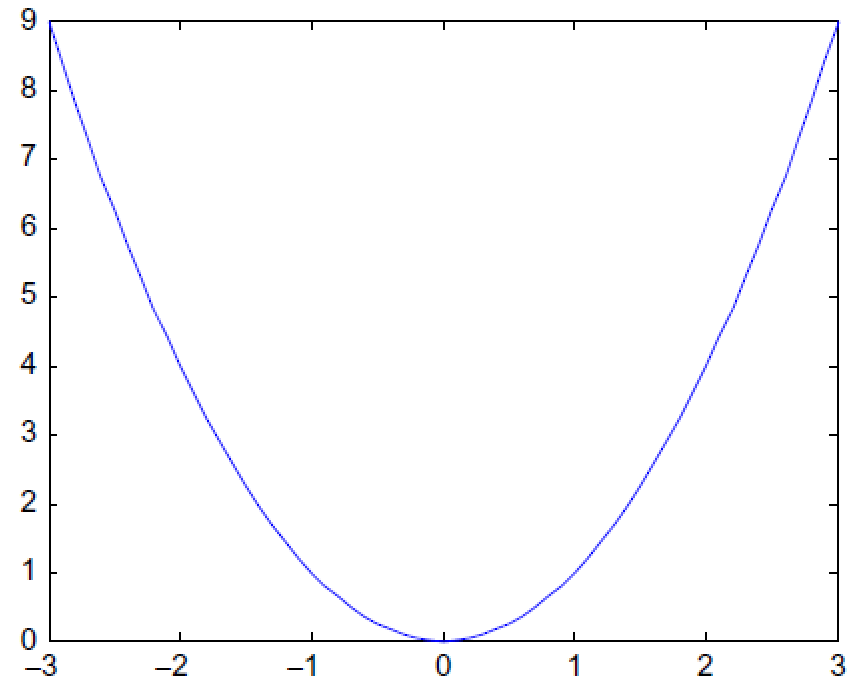
- Plot (x,y) draws points of y w.r.t points of x vector and connects the points together using straight lines.
- Note: the 1st argument is the horizontal axis and the 2nd argument is the vertical axis

Plotting Vectors (cont'd)

Increasing the resolution of a Plot

- To improve the resolution of the plot, you need to increase the number of points for the x vector

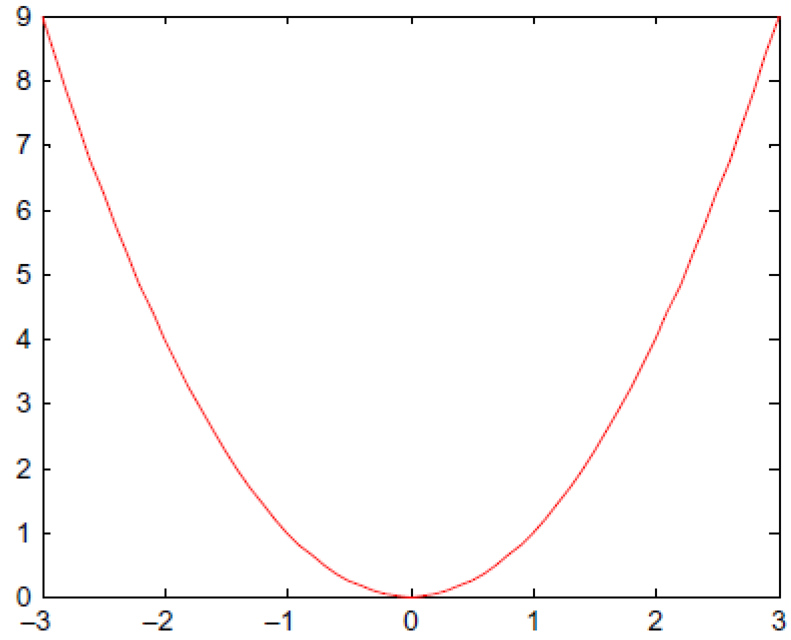
```
x = -3:0.1:3;
y = x.^2;
plot(x,y)
```



Plotting Vectors (cont'd)

Changing The Color of a Plot

```
>>plot(x,y,'r')
```

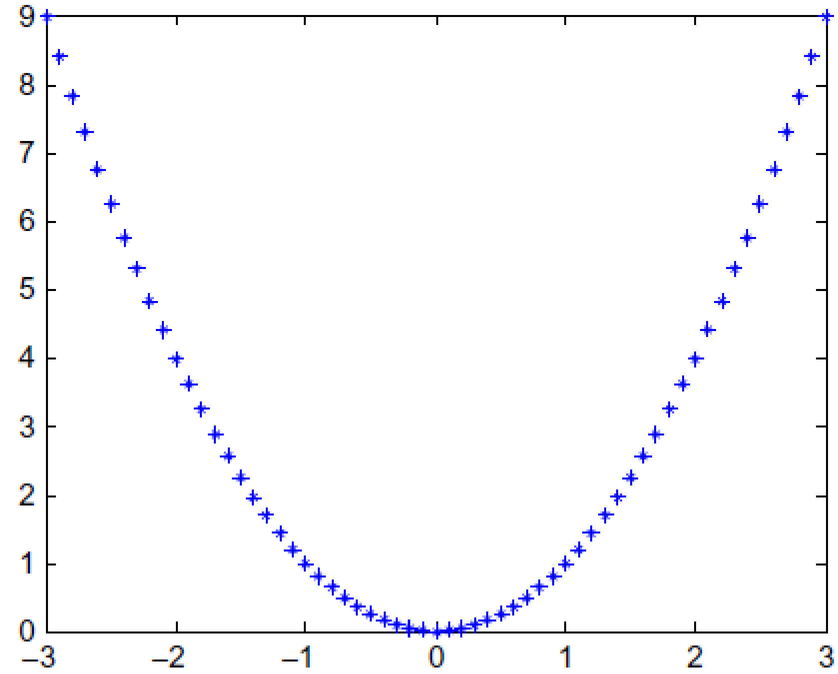


- Matlab support the colors:
  - red "r",
  - green "g",
  - blue "b",
  - cyan "c",
  - magenta "m",
  - yellow "y",
  - white "w"
  - black "k".

Plotting Vectors (cont'd)

Draw a Function as Points

```
>>plot(x,y,'*')
```



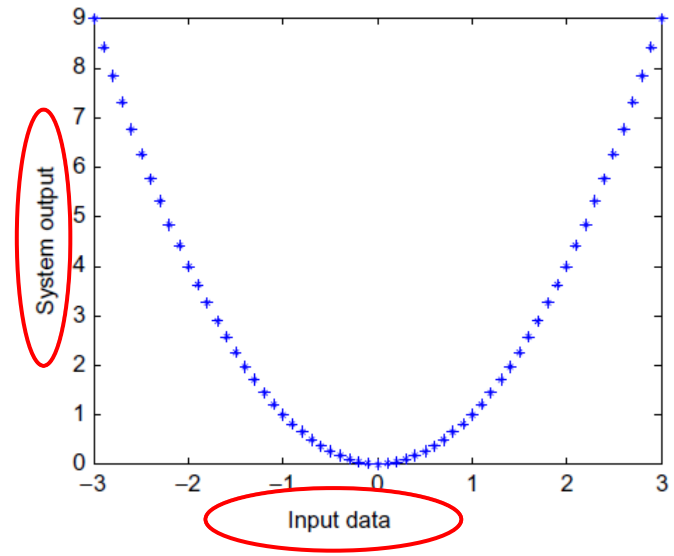
- Different symbols can be used to represent points in a curve. e.g., "+", "o" or "x".
- For more information: use help

```
>>help plot
```

Plotting Vectors (cont'd)

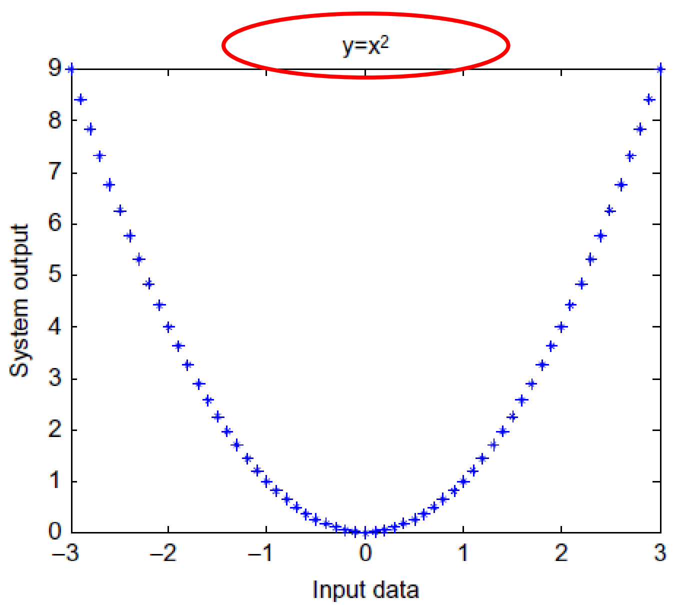
Labeling the x and y Axes

```
xlabel('Input data')
ylabel('System output')
```



Adding a Title to a Figure

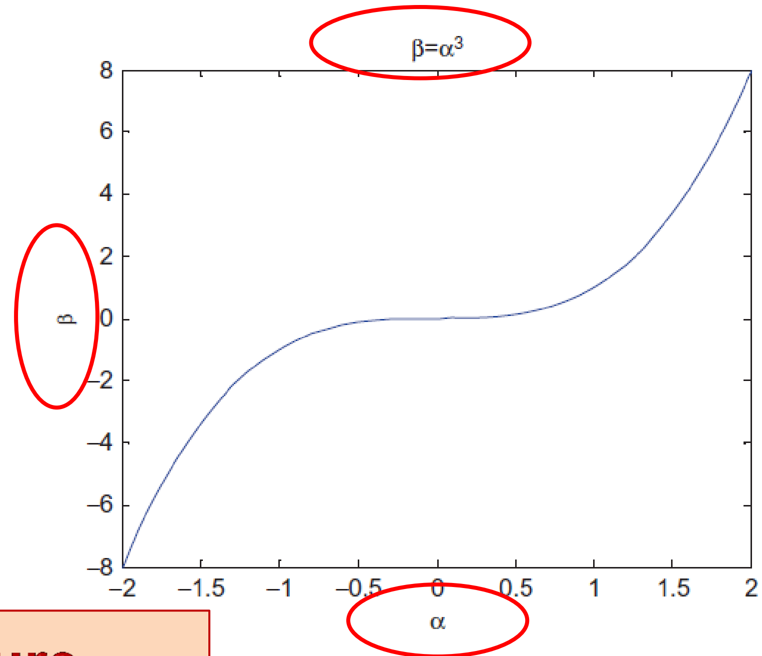
```
>>title('y=x^2')
```



Plotting Vectors (cont'd)

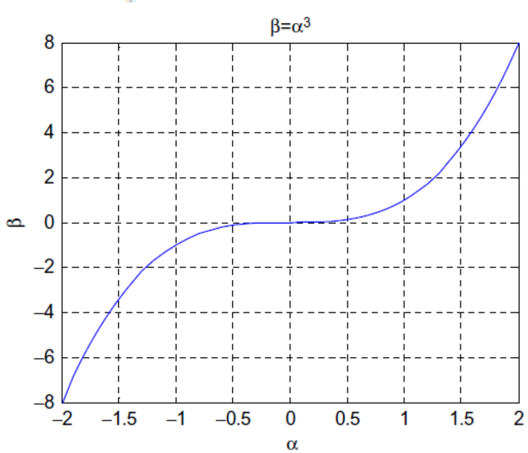
Using Greek Letters

```
alpha = -2:0.1:2;
beta = alpha.^3;
plot(alpha,beta)
xlabel('\alpha')
ylabel('\beta')
title('\beta=\alpha^3')
```

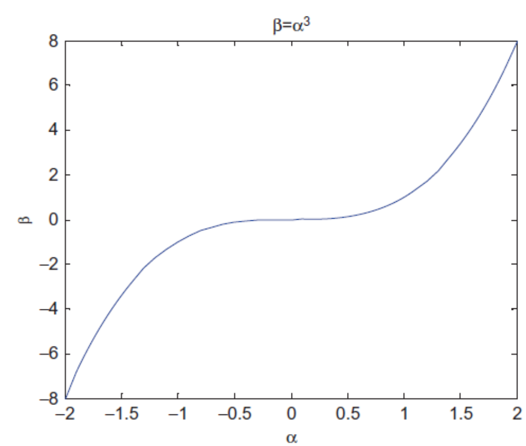


Adding/Removing a Grid to/from a Figure

```
>> grid on
```



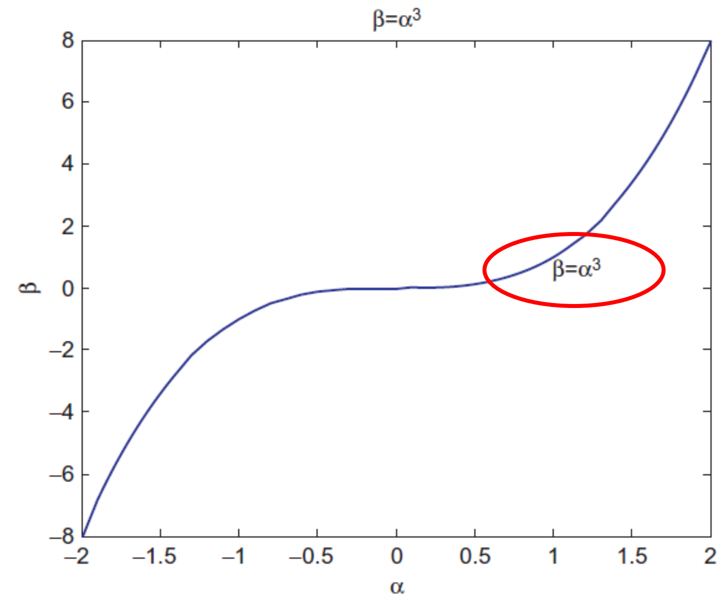

```
>> grid off
```



Plotting Vectors (cont'd)

Adding a Text to a Figure

```
>> text(1,0.75,'\beta = \alpha^3')
```



Here, the added text starts at the plot coordinate location (1, 0.75) on the figure.

Changing the Font Size

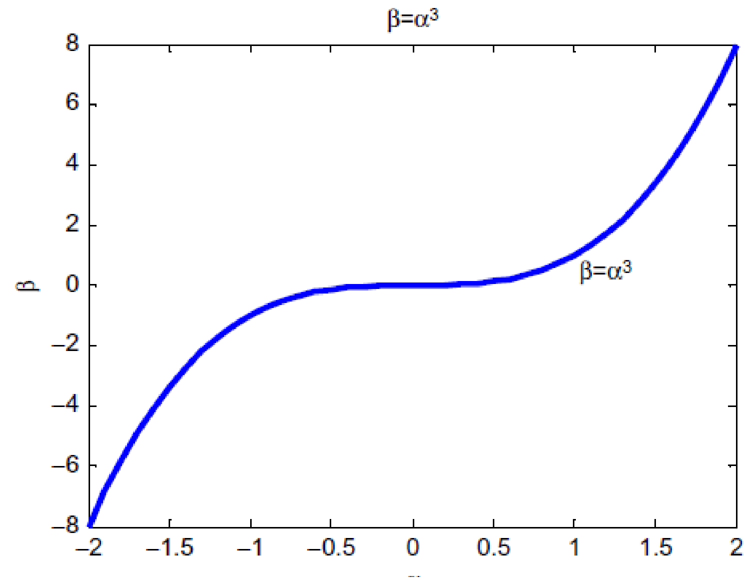
We can set the font size for the axes labels, the figure title, and any text added to the figure.

```
alpha = -2:0.1:2;
beta = alpha.^3;
plot(alpha, beta)
xlabel('\alpha','FontSize',24)
ylabel('\beta','FontSize',24)
title('\beta = \alpha^3','FontSize',17)
text(1,0.75, '\beta = \alpha^3','FontSize',18)
```

Plotting Vectors (cont'd)

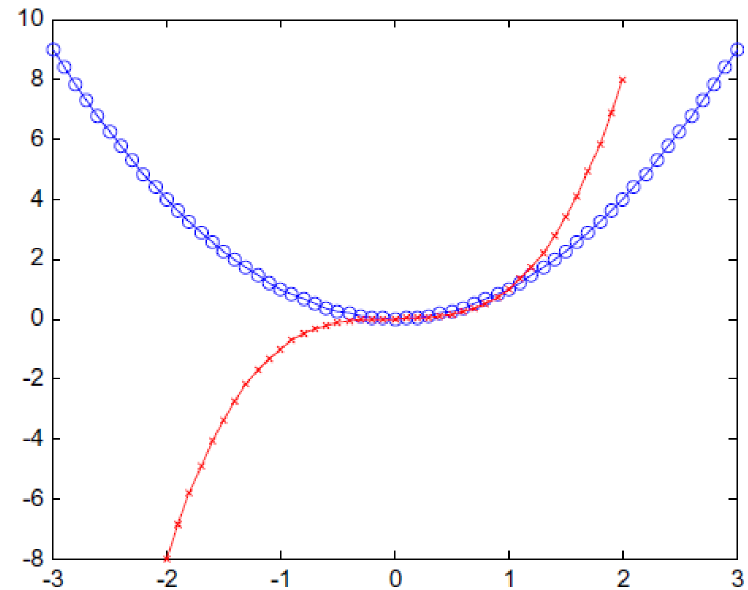
Changing the Line Width

```
alpha =-2:0.1:2;
beta = alpha.^3;
plot(alpha, beta ,'LineWidth',3)
```



Multiple Plots

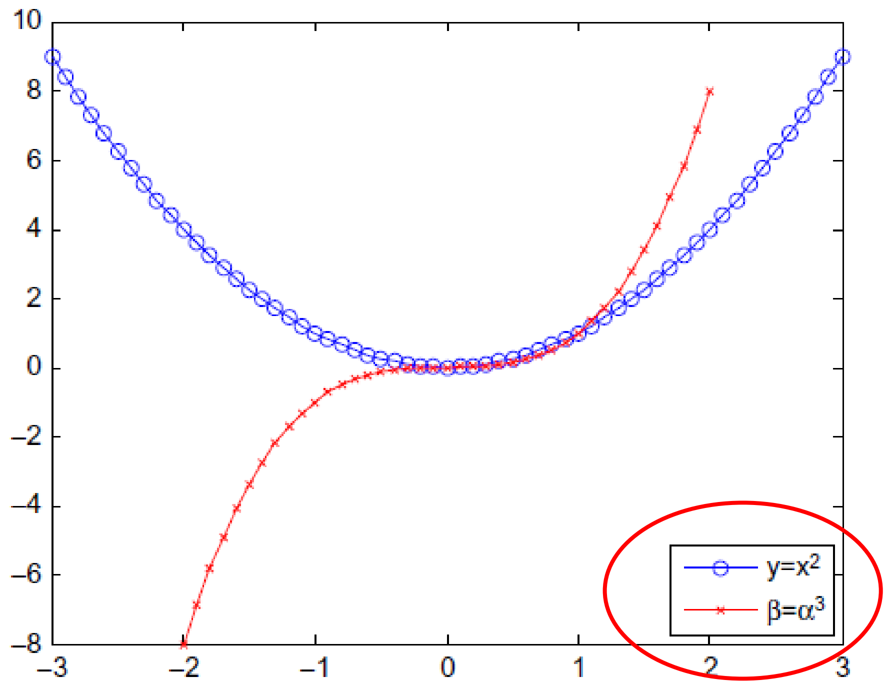
```
x =-3:0.1:3;
y = x.^2;
plot(x,y, 'bo-')
hold on
alpha =-2:0.1:2;
beta = alpha.^3;
plot(alpha, beta,'rx-')
hold off
```



Plotting Vectors (cont'd)

Adding a Legend to a Plot

```
>> legend('y=x^2','\beta=\alpha^3','Location','SouthEast')
```



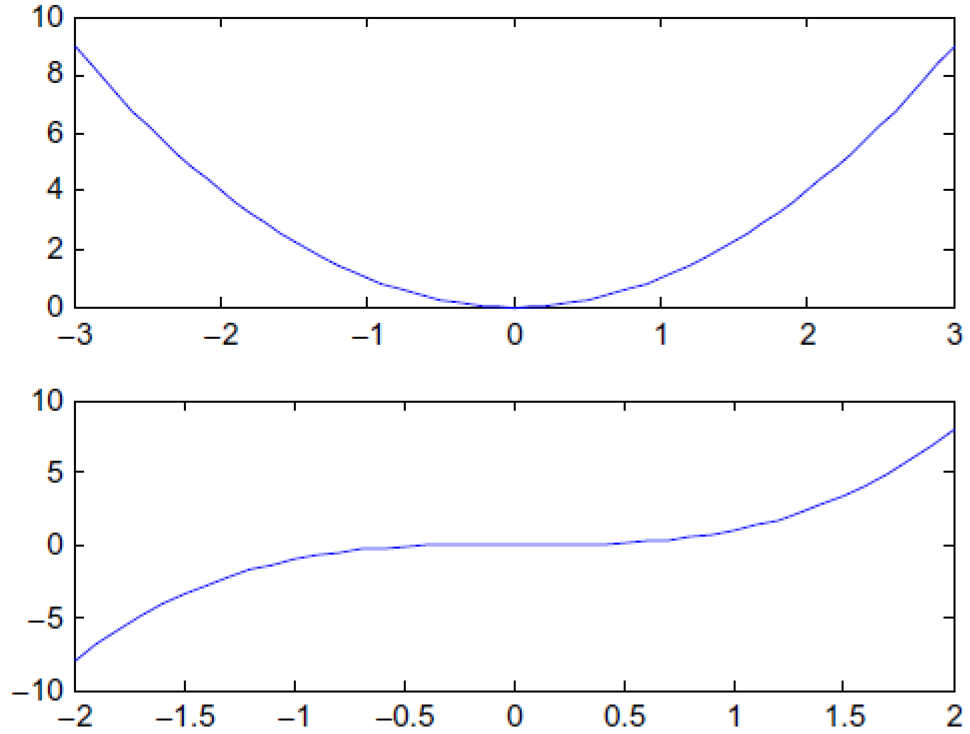
Note: 'Location' parameter sets the location of the legend on the figure: e.g.,

'SouthEast', 'SouthWest', 'NorthEast', 'NorthWest'

Plotting Vectors (cont'd)

Multiple Subplots

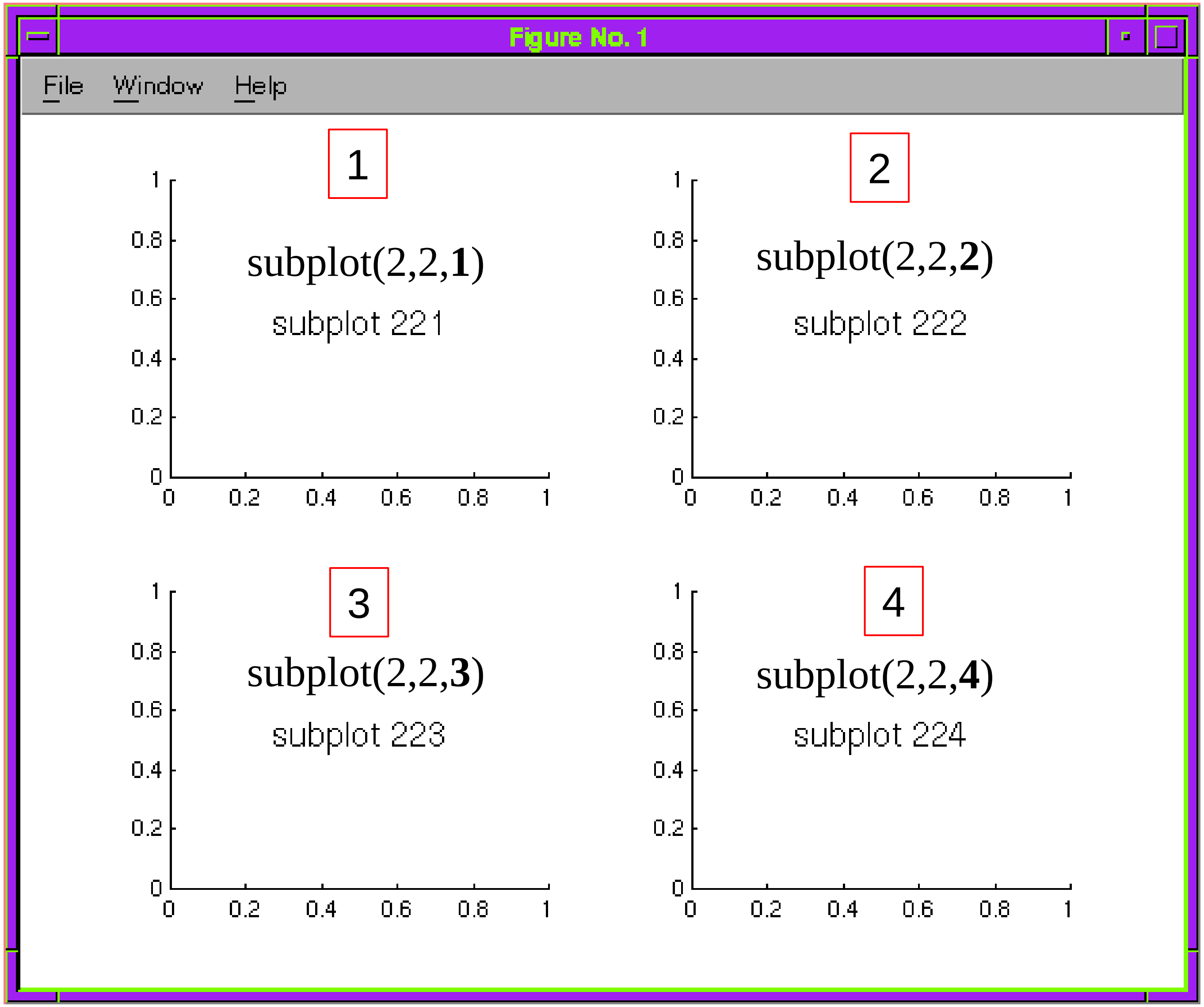
```
x1 = -3:0.1:3;
y1 = x1.^2;
subplot(2,1,1)
plot(x1,y1)
x2 = -2:0.1:2;
y2 = x2.^3;
subplot(2,1,2)
plot(x2,y2)
```



The `subplot(m,n,p)` command breaks the figure window down into an `m×n` matrix of smaller axes and selects the `pth` axis to display the current plot. For example,

Plotting Vectors (cont'd)

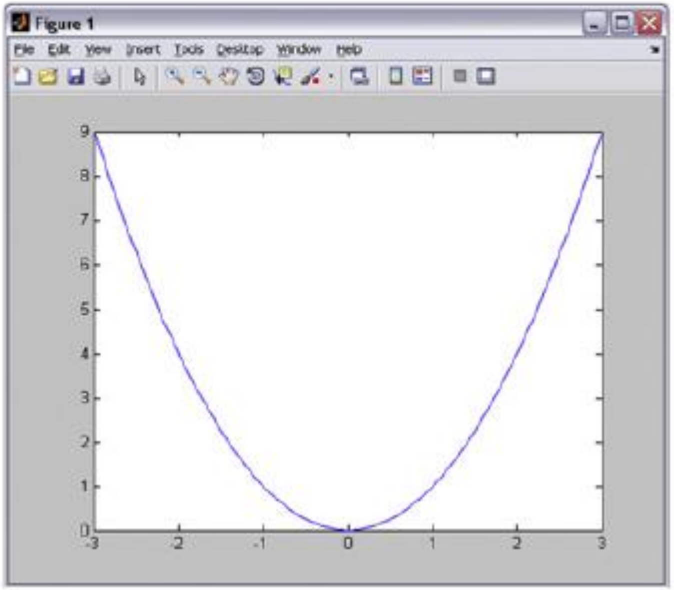
Multiple Subplots (cont'd)

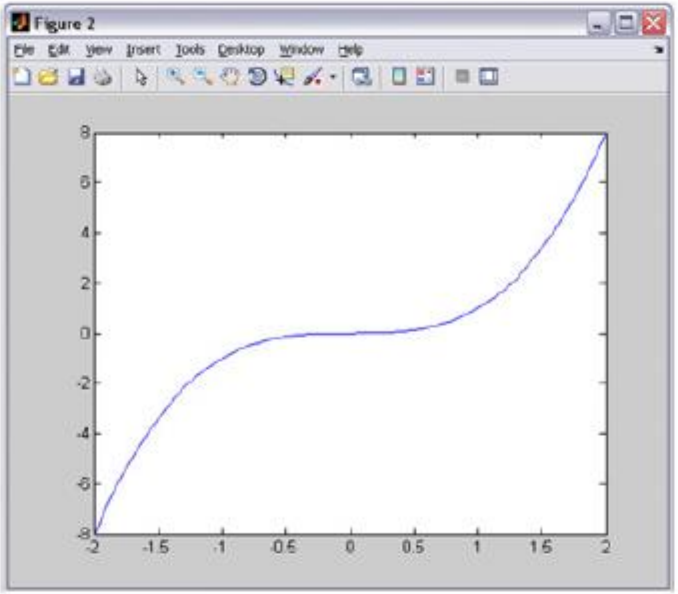


Plotting Vectors (cont'd)

Multiple Figures

```
x1 = -3:0.1:3;
y1 = x1.^2;
figure(1)
plot(x1,y1)
x2 = -2:0.1:2;
y2 = x2.^3;
figure(2)
plot(x2,y2)
```





Chapter 4:
Arrays in Matlab

Next Lecture

Thanks for attention