MATLAB - MATrix LABoratory

- Powerful tool in engineering problem solving, data analysis, modeling and visualisation.
- The “interactive environment” is user friendly.
- MATLAB provides its own high-level language in which users can extend the capabilities of MATLAB.
  - C-like programming language
  - Many user defined functions

MATLAB - MATrix LABoratory

- The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.
- Useful throughout your degree & used in other courses.

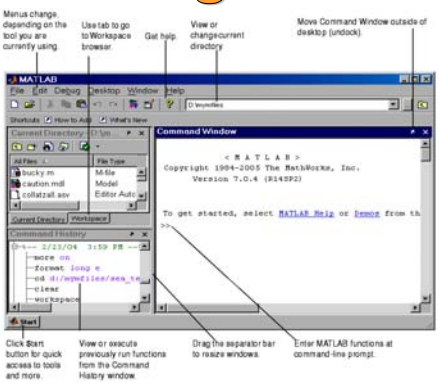
MATLAB - MATrix LABoratory

- Specific applications are collected in packages referred to as toolbox.
  - Signal processing,
  - Image processing,
  - Communication
  - Symbolic computation,
  - Control theory,
  - Simulation,
  - Optimization,
  - Several other fields of applied science and engineering.

Starting MATLAB

- When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:
  - The Command Window
  - The Command History
  - The Workspace
  - The Current Directory
  - The HelpBrowser
  - The Start button

Starting MATLAB



Using MAtlab as a calculator

Matlab capable of simple mathematical operations analogous to a calculator:

```
>> 9.3 + 5.6
ans =
  14.9000
>> 13.1 - 4.113
ans =
   8.9870
>> 10.1 * 890.99
ans =
  8.9990e+03
```


Using MAtlab as a calculator

```
>> 9.6 / 3.2
ans =
   3.0000
>> 9.9 ^ 3.1
ans =
  1.2203e+03
```


Using MAtlab as a calculator

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable ans, short for answer, to store the results of the current calculation.

Notethat the variable ans is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example,

```
>> x = 1+2*3
x =
       7
```

Using MAtlab as a calculator

The following basic mathematical operators are supported by Matlab:

Symbol	Operation	Example
+	Addition	2+3
-	Subtraction	2-3
*	Multiplication	2*3
/ or \	Division	2/3
^	Exponentiation	2^3

Precedence of Arithmetic Operations

Precedence	Operation
1	Parentheses, innermost first
2	Exponentiation, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

For example,

$$\frac{1}{2+3^2}+\frac{4}{5}\times\frac{6}{7}$$

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7
ans =
0.7766
```


Output Suppression

To suppress the screen output use a semicolon:

```
>> 5 ^ 2;      (" ^ " is the power operator, 5 raised to
                the second power in this example)
>>             ( no response from matlab)
```

To examine the current value of ans type

```
>> ans
ans =
      25
```


Creating MATLAB variables

MATLAB variables are created with an assignment statement. The syntax of variable assignment is:

variable name = a value (or an expression)

For example,

```
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

- manual entry
- built-in functions
- user-defined functions

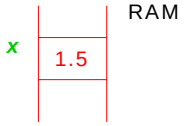
Creating MATLAB variables

All variables are created as array variables. For example,

>> *x = 1.5;* **(this is called an "assignment statement")**

creates a (1 x 1) array of type real and assigns the value 1.5 to x.

Variables only change values when you assign values to them using an assignment statement.



To see the value of x type,

>> *x*

x =

1.5000

Variable names must begin with a character and are case sensitive.

Creating MATLAB variables

The variable to be "assigned" a value must be on the left hand side of the assignment statement...

>> *1.5 = x;* **(illegal "assignment statement")**

The above is illegal because "1.5" does not specify a legal address in memory.

An assignment statement is not a "math" equation. For example, the following sequence is legal in Matlab:

>> *x = 2.0;*

>> *x = x + 1.0*

x =

3.0000

Whereas the math equation x = x + 1.0 has no solution!

Workspace

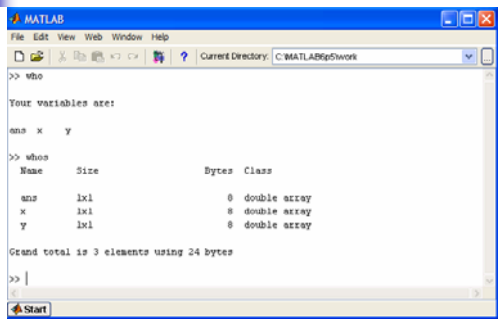
The current environment for our computing session is called the workspace.

We can have many variables and functions in our workspace.

For example:

```
>> x=1+2*3;
>> y=5^2;
>> 7*8+5/2;
```


Workspace



System Commands

>> save la3 (saves all the workspace variables to the file la3.mat)

>> load la3 (loads the la3.mat file from the current directory)

>> diary on (saves all text, displayed in the command window, in a file named "diary")

>> diary off (turns off the diary command)

>> diary (toggles the diary command on/off)

System Commands

>> clc (clears the command window)

>> clf (clears the figure window)

>> clear var1 var2 ... (clears only the variables var1 var2 ...)

>> clear (clears(deletes) all variables in the workspace)

>> ctrl-c (aborts a MATLAB computation)

```
>> help cos
 COS    Cosine.
   COS(X) is the cosine of the elements of X.
```

>> lookfor Function Name

Output Format

***format** options* **(controls how numeric output is displayed)**

Examples-- two of the ***options*** are shown below:

```
>> x = 12.34567890123456789
x =
        12.34567890123457

>> format short
>> x
x =
        12.3457
>> format long e
>> x
x =
        1.234567890123457e+01
```


Mathematical Functions

Matlab contains literally hundreds of built-in functions:

abs(x)	Absolute value
acos(x)	Inverse cosine
acosh(x)	Inverse hyperbolic cosine
angle(x)	Phrase angle
asin(x)	Inverse sine
asinh(x)	Inverse hyperbolic sine
atan(x)	Inverse tangent
atanh(x)	Inverse Hyperbolic tangent
ceil(x)	Round towards +infinity

Mathematical Functions

conj(x)	Complex conjugate
cos(x)	Cosine
cosh(x)	Hyperbolic cosine
exp(x)	Exponentiation e^x
fix(x)	Round towards zero
floor(x)	Round towards –infinity
gcd(x,y)	Greatest common divisor
imag(x)	Complex imaginary part
lcm(x,y)	Least common multiple
log(x)	Natural logarithm
log10(x)	Common (base 10) logarithm

Mathematical Functions

real(x)	Complex real part
rem(x,y)	Remainder after division
round(x)	Round towards nearest integer
sign(x)	Sign
min(x)	Minimum value
max(x)	Maximum value
sin(x)	Sine
sinh(x)	Hyperbolic sine
sqrt(x)	Square root
tan(x)	Tangent
tanh(x)	Hyperbolic tangent

Mathematical Functions

```
>> abs(-5.2)
ans =
    5.2000
>> ceil(4.3)
ans =
     5
>> ceil(4.7)
ans =
     5
```

```
>> fix(6.3)
ans =
     6
>> fix(-6.7)
ans =
    -6
>> floor(7.5)
ans =
     7
```

```
>> floor(7.3)
ans =
     7
>> floor(-7.9)
ans =
    -8
>> round(5.1)
ans =
     5
```

```
>> round(5.5)
ans =
     6
>> ceil(-4.3)
ans =
    -4
>> rem(9,2)
ans =
     1
```


Mathematical Functions

As a first example, the value of the expression

$$y = e^{-a}\sin(x) + 10\sqrt{y},$$

for a=5, x=2, y=8

```
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y =
28.2904
```

Mathematical Functions

Second example is

>> log(142) → Natural Logarithm
ans =
4.9558
>> log10(142) → Decimal Logarithm
ans =
2.1523

Arrays - Vectors and Matrices

In Matlab (for CS101) all data will be stored in arrays.
An array can represent a:
vector - size 1 x n (row vector with 1 row and n columns) or
n x 1 (column vector with n rows and 1 column)
matrix - any size m x n (a table of m rows and n columns).

Conventions
Two arrays are defined to be equal if they are of the same size and at each position, they have the same value.

Row and Column Vectors

We can represent a vector either as a row vector or as a column vector, depending on the application. E.g.,

$$\mathbf{V}=\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \text{ or } \mathbf{V}=\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Creating a Row-Vector

To create vector arrays in MATLAB type the following:

```
>> F1 = [1  2.5  3.5];
>> F2 = [-1.75,  2.75,  1.0e-1];
>> F3 = [1  2 ];
>> whos F1 F2 F3
```

(The use of the " , " operator is optional)

Name	Size	Bytes	Class
F1	1x3	24	double array
F2	1x3	24	double array
F3	1x2	16	double array

F1 and F2 are 1 row x 3 column arrays and F3 is a 1 row x 2 column array.
To display their values type the variable name:

```
>> F1
F1 =
    1.0000    2.5000    3.5000
```

Creating a Column-Vector

To create column-vector arrays in MATLAB either use the semi-colon operator ; or use the transpose operator '

```
>> F1 = [1 ; 2.5;  3.5];
```
(; means ... append to next row)

```
>> F2 = [-1.75,  2.75,  1.0e-1] ' ;
```
(transpose changes row vector to column vector and vice-versa)

```
>> F3 = [ ];
```
(empty vector)

```
>> whos F1 F2 F3
```

Name	Size	Bytes	Class
F1	3x1	24	double array
F2	3x1	24	double array
F3	0x0	0	double array

```
>> F1
F1 =
    1.0000
    2.5000
    3.5000
```


Creating a Vector: Colon Operator

The " : " (colon) operator is important in construction of arrays

```
>> x = 1:100;
```
(creates a 1x100 row vector with values 1,2,3,4, … 100)

```
>> y = (-1 :  0.1 : 1)';
```
(creates a column vector with starting value -1 and increments by 0.1 until 1 is reached but not exceeded)

```
>> z = 0 : 0.3 : 1
z =
    0    0.3000    0.6000    0.9000
```


Creating a Vector: Subscripting

Use (parenthesis) to select a subset of the values of an array.

```
>> x = [10 9 8 7];
>> y = [3; 4; 5];
>> z = x(2)
```
(note the use of parenthesis and not the brackets)

```
z =
     9
```
(subscripting x does not change x)

```
>>  z = x(1:3)
```
(note the use the colon operator)

```
z =
    10     9     8
>> x = x(2:end)
```
(now x has changed)

```
x =
     9     8     7
>> z = y(1:2)
z =
     3
     4
```


Creating a Vector: Function linspace

linspace(a,b,N) creates a row vector of N equally spaced values starting at a and ending at b. This is equivalent to a : (b-a)/(N-1) : b

Example:
Create a row vector vec1 of 4 points starting at -1 and ending at +1.

```
>> vec1 = linspace(-1,1,4)
vec1 =
   -1.0000   -0.3333    0.3333    1.0000
```

Arrays - Arithmetic operators

+	-	*	/	\	^
(dot form) →	.*	./	.\	.^	
add	sub	mult	right div	left div	power

If A and B are vectors then it is not always true that
>>A op B
is defined when ***op*** is one of the operations in the table above.
The size of an vector is its number of rows and columns.
When multiple operators are used in an expression , use the rules of precedence.
We will consider some operators on a case by case basis.

Vectors - Addition

A and B must be of the same size or a scalar.

```
>> A = [2 3 4] ;
>> B = 3;
>> A + B
ans    =
       5       6       7

>> B = [1 2 3] ;
>> A + B
ans    =
       3       5       7
```


Vectors - Subtraction

A and B must be of the same size or a scalar.

```
>> A = [2 3 4] ;
>> B = 3;
>> A - B
ans    =
      -1       0       1

>> B = [1 2 3] ;
>> A - B
ans    =
       1       1       1
```


Vectors - Scalar Product

In Cartesian coordinates, we can express the dot or scalar product of two vectors as a matrix multiplication:

$$\mathbf{F} \bullet \mathbf{s} = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} \bullet \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = f_1 s_1 + f_2 s_2 + f_3 s_3$$

```
>> s = [1, 1, 1] ' ;
>> F1 = [1  2.5  3.5];
>> F1 * s                    (dot or scalar product)
ans =
     7
```

Vectors - Scalar Product

In order for the scalar product of vectors A * B to be defined, the number of columns in the row vector A must equal the number of rows in the column vector B.

```
>> A = [2 3 4] ;
>> B = [1 ; 2 ; 3] ;
>> A * B
ans    =
       20
```


Vectors - (dot) Multiplication

The size of A and B must be the same or either could be a scalar .

```
>> A = [2 3 4] ;
>> B = [1 2 3] ;
>> A .* B
ans    =
       2      6      12
```

Note: the result is the product of the individual elements of the vectors.

Matlab Built-in Functions

Exponential Function

```
>> exp(0)                    (exponential function e^x )
ans =
       1

>> x = [-1 0 1];
>> exp(x)
ans =
       0.3679   1.0000   2.7183   ( [exp(-1), exp(0), exp(1)] )

>> dot(F1,s)
ans =
       7
```


Linear Systems of Equations

To solve a system of equations A*B = C , i.e. to find the point of intersection of the three planes, use the " \ " operator.

```
>> A = [100 10 1; 225 15 1; 400 20 1] ;
>> C = [1 ; 20 ; 10] ;
>> B = A \ C
B      =
      -0.5800
      18.3000
    -124.0000
```

A is called the "coefficient matrix" for the system of linear equations.

The three planes intersect at the point (-0.5800, 18.3000, -124.0000)

You can test this is the solution by typing

```
>>A*B
       1.0000
      20.0000
      10.0000
```

Creating a Matrix

A matrix is an array of numbers. To type a matrix into MATLAB you must

- begin with a square bracket, [
- separate elements in a row with spaces or commas (,)
- use a semicolon (;) to separate rows
- end the matrix with another square bracket,].

Example: For a general 3 x 3 matrix the math notation is,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Creating a Matrix

In MATLAB the above matrix is created by

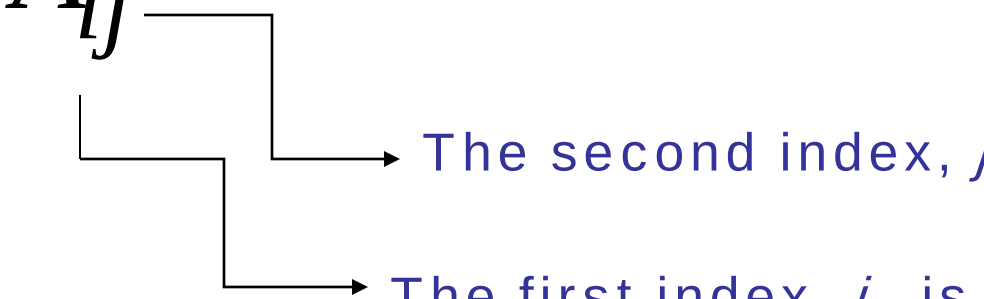
```
>> A = [1 2  3 ; 4 5 6 ; 7 8 9]
```

MATLAB then displays the 3 × 3 matrix

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

Matrix indexing

A_{ij}



The second index, *j*, is the column number

The first index, *i*, is the row number

For example, A(1,3) is an element of first row and third column.

```
>> A(1,3)

ans =

     3
```

Matrix indexing

Correcting any entry is easy through indexing.

Here we substitute A(3,3)=9 by A(3,3)=0.

The result is

>> ***A(3,3)=0***

A =

1	2	3
4	5	6
7	8	0

Selecting elements from a Matrix - Subscripting

Example

```
>> mat1 = [1 -2 ;5 3 ] ;
>>  mat1(1,1) ( first row ,first column)
ans =
        1
>> mat1(1,2) ( first row ,second column)
ans =
       -2
>> mat1(2,1) (second row, first column)
ans =
        5
>> mat1(2,2) (second row, second column)
ans =
        3
```

Colon operator

You can select an entire row or column of a matrix by using the colon " : " symbol.

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

>> ***A(1,:)*** **(first row, any column)**

ans =

1 2 3

>> ***A(2, :)*** **(second row, any column)**

ans =

4 5 6

Colon operator

You can select an entire row or column of a matrix by using the colon " : " symbol.

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

>> A(: ,1)  (any row, first column)
ans =
        1
        4
        7

>> A(: ,2) (any row, second column)
ans =
        2
        5
        8
```

Colon operator

The colon operator can also be used to extract a sub-matrix from A.

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

>> A(:,2:3)

ans =

     2     3
     5     6
     8     9
```

A(:,2:3) is a sub-matrix with the last two columns of A.

Creating a sub-matrix

To extract a sub-matrix B consisting of rows 2 and 3 and columns 1 and 2 of the matrix A

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

>> B = A([2 3],[1 2])

B =

    4    5
    7    8
```

Creating a sub-matrix

To interchange rows 1 and 2 of A, use the vector of row indices together with the colon operator.

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

>> C = A([2 1 3],:)

C =

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

Creating a sub-matrix

To interchange column 1 and 2 of A, use the vector of column indices together with the colon operator.

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

>> C = A(:,[2 1 3])

C =

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

Creating a sub-matrix

To create a vector version of matrix A,

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

ans =

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

Creating a sub-matrix

```
>> A = [1 2  3 ; 4 5 6 ; 7 8 9];
>> A(2:3,2:3)
ans =
        5 6
        8 9
>> A(end:-1:1,end)
ans =
        9
        6
        3
>> A([1 3],[2 3])
ans =
        2 3
        8 9
```

Deleting row or column

To delete a row or column of a matrix, use the empty vector operator, []

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

Third row of matrix A is now deleted.

To restore the third row, we use a technique for creating a matrix

```
>> A = [A(1,:);A(2,:);[7 8  9]]
A =

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

Deleting row or column

```
>> A(:,1) = []
A =

     2     3
     5     6
     8     9
```

First column of matrix A is now deleted.

To restore the first column, we use a technique for creating a matrix

```
>> A = [[1;4;7] A(:,1) A(:,2)]
A =

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

Dimension

To determine the dimensions of a matrix or vector, use the command size. For example,

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

    3    4    5    6
    2    3    4    5

>> size(A)

ans =

    2    4
```

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

m =

    2


n =

    4
```

Matrix generators

Arrays can be constructed from built-in Matlab functions

Matrix	Description
`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Matrix generators

```
>> A  = eye(3)
A =
     1     0     0
     0     1     0
     0     0     1
```

(eye(n) constructs an n x n “identity” matrix)

```
>> A = ones(3,2)
A =
     1     1
     1     1
     1     1
```

(ones(m,n) constructs an m x n array of 1’s)

```
>> A = zeros(2,3)
A =
     0     0     0
     0     0     0
```

(zeros(m,n) constructs an m x n array of 0’s)

Matrix generators

```
>> K = [1 2 3];
>> diag(K,0)
```

(diag(K,0) constructs a diagonal matrix with values of K down the diagonal)

```
1   0   0
0   2   0
0   0   3
```

(green line represents the "diagonal" of a matrix")

```
K = [2 3];
>> diag(K,1)
```

(diag(K,1) constructs a matrix with values of K above the diagonal)

```
0   2   0
0   0   3
0   0   0
```

Matrix generators

```
>> K = [3 4];
>> diag(K,-1)
```

(diag(K,-1) constructs a diagonal matrix with values of K below the diagonal)

```
     0     0     0
     3     0     0
     0     4     0
```

```
>> rand(2,5)

ans =

    0.9501    0.6068    0.8913    0.4565    0.8214
    0.2311    0.4860    0.7621    0.0185    0.4447
```

Matrix generators

```
>> C = [1 2;3 4]

C =

     1     2
     3     4

>> D = [C zeros(2); ones(2) eye(2)]

D =

     1     2     0     0
     3     4     0     0
     1     1     1     0
     1     1     0     1
```

Matrix generators

magic(N) is an N-by-N matrix constructed from the integers 1 through N^2 with equal row, column, and diagonal sums. (Except N=2 !!!)

```
>> magic(2)

ans =

     1     3
     4     2
```

```
>> magic(3)

ans =

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

Matrix generators

vander(v) is an N-by-N the Vandermonde matrix whose columns are powers of the vector v, that is

A(i,j) = v(i)^(n-j)

```
>> v=[1 2 3];
>> F=vander(v)

F =

     1     1     1
     4     2     1
     9     3     1
```

```
>> v=[0.2 0.4 0.6 1];
>> F=vander(v)

F =

    0.0080    0.0400    0.2000    1.0000
    0.0640    0.1600    0.4000    1.0000
    0.2160    0.3600    0.6000    1.0000
    1.0000    1.0000    1.0000    1.0000
```

Arithmetic operations

MATLAB has two different types of arithmetic operations:

- Matrix arithmetic operations
- Array arithmetic operations (element-by-element)

Operations	Matrix	Array
Addition	+	+
Subtraction	-	-
Multiplication	*	.*
Division	/	./
Left Division	\	.\
Exponentiation	^	.^

Matrix arithmetic operations

A+B or B+A	A and B must be same size
A*B	Number of column of A must be equal number of rows of B
A^2	A is square and equals A*A
c*A or A*c	Multiply each element of A by c
A/B	$A*B^{-1}$
A\B	$A^{-1}*B$

Array arithmetic operations

A and B are two matrices of the same size with elements $A = [a_{ij}]$ and $B = [b_{ij}]$, then

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$A.*B = \begin{bmatrix} a_{11} * b_{11} & a_{12} * b_{12} \\ a_{21} * b_{21} & a_{22} * b_{22} \end{bmatrix} \qquad A.\hat{} B = \begin{bmatrix} a_{11} \hat{} b_{11} & a_{12} \hat{} b_{12} \\ a_{21} \hat{} b_{21} & a_{22} \hat{} b_{22} \end{bmatrix}$$

$$A./B = \begin{bmatrix} a_{11} / b_{11} & a_{12} / b_{12} \\ a_{21} / b_{21} & a_{22} / b_{22} \end{bmatrix} \qquad A.\backslash B = \begin{bmatrix} b_{11} / a_{11} & b_{12} / a_{12} \\ b_{21} / a_{21} & b_{22} / a_{22} \end{bmatrix}$$

Addition

```
>> A = [2 3 4; 5 6 7] ;
>> B = 3;
>> A + B
ans     =
        5        6        7
        8        9       10
```

```
>> B = [1 2 3; 4 5 6] ;
>> A + B
ans     =
        3        5        7
        9       11       13
```

Subtraction

```
>> A = [2 3 4; 5 6 7; 8 9 10] ;
>> B = 3;
>> A - B
ans     =
        -1      0       1
        2       3       4
        5       6       7
>> B = [1 2 3; 4 5 6; 7 8 9] ;
>> A - B
ans     =
        1       1       1
        1       1       1
        1       1       1
```

Arrays - (dot) Multiplication

The size of A and B must be the same or either could be a scalar .

```
>> A = [2 3 4; 5 6 7] ;
>> B = [1 2 3; 4 5 6] ;
>> A .* B
ans    =
        2     6    12
       20    30    42
```

Note: the result is the product of the individual elements of the array.

Matrix Multiplication

If A has size (m x n) and B has size (n x k)
then there are m x k possible combinations.

```
>> A = [2 3 4; 5 6 7] ;
>> B = [1 2 3; 4 5 6; 7 8 9];
>> C = A * B
C      =
        42      51      60
        78      96     114
```

Note: the size of A * B is computed by:
size(A) is 2 x 3
size(B) is 3 x 3
size(A * B) is 2 x 3

Note: In general
A * B does not equal B * A.

In fact A * B is defined only when the number of columns of A equals the number of rows of B.

Matrix Multiplication

Example:

```
>> A = [2 3 4; 5 6 7] ;
>> B = [1 ; 2 ; 3] ;
>> C = A * B
C      =
          20
          38
```

Note: again, the size of A * B is computed by:
size(A) is 2 x 3
size(B) is 3 x 1
size(A * B) is 2 x 1

Matrix Function

MATLAB provides many matrix functions for various matrix/vector manipulations

det	Determinant
diag	Diagonal matrices and diagonals of a matrix
sum	Sum of elements
eig	Eigenvalues and eigenvectors
inv	Matrix inverse
norm	Matrix and vector norm
trace	Sum of diagonal elements
rank	Number of linearly independent rows or columns

Matrix Function

```
>> A = [1 2  3 ; 4 5 6 ; 7 8 0];
>> det(A)

ans =

   27

>> inv(A)

ans =

   -1.7778    0.8889   -0.1111
    1.5556   -0.7778    0.2222
   -0.1111    0.2222   -0.1111
```

```
>> A = [2  3 ; 5  8]
A =

     2     3
     5     8

>> inv(A)

ans =

    8.0000   -3.0000
   -5.0000    2.0000
```

Matrix Function

```
>> A

A =

     1     2     3
     4     5     6
     7     8     0
```

```
>> diag(A)

ans =

     1
     5
     0
```

```
>> diag(A,1)

ans =

     2
     6
```

```
>> diag(A,2)

ans =

     3
```

```
>> diag(A,3)

ans =

  Empty matrix: 0-by-1
```

```
>> diag(A,-1)

ans =

     4
     8
```

```
>> diag(A,-2)

ans =

     7
```

Matrix generators

```
>> A=magic(4)

>> A

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>>sum(A)
ans =
    34    34    34    34
```

```
>> sum(A,1)
ans =
    34    34    34    34

>> sum(A,2)

ans =
    34
    34
    34
    34
>> sum(diag(A)) % or trace(A)

ans =
    34
```

Matrix Function

```
>> A = [1 2  3 ; 4 5 6 ; 7 8 0]

A =

     1     2     3
     4     5     6
     7     8     0

>> rank(A)

ans =

     3

>> det(A)

ans =

    27
```

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

A =

     1     2     3
     4     5     6
     7     8     9

>> rank(A)

ans =

     2

>> det(A)

ans =

     0
```

Matrix Function

For matrices

NORM(X) is the largest singular value of X, max(svd(X)).

NORM(X,2) is the same as NORM(X).

NORM(X,1) is the 1-norm of X, the largest column sum,
= max(sum(abs(X))).

NORM(X,inf) is the infinity norm of X, the largest row sum,
= max(sum(abs(X')))
= max(sum(abs(X),2)) .

NORM(X,'fro') is the Frobenius norm,
=sqrt(sum(diag(X'*X))).

Matrix Function

```
>>  A=rand(3)

A =
    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169

>> norm(A,1)
ans =
    2.2581

>> sum(abs(A))
ans =
    1.8521    1.8363    2.2581

>>  max(sum(abs(A)))
ans =
    2.2581

>> norm(A,inf)
ans =
    2.2891

>> sum(abs( A' ))
ans =
    1.7722    2.2891    1.8851

>>  max(sum(abs( A' )))
ans =
    2.2891
```

$$Norm_1 = \max(abs(sum(A)))$$

$$Norm_{\inf} = \max(abs(sum(A')))$$

Matrix Function

```
>> norm(A,'fro')
ans =

    2.1236
```

```
>> diag(A'*A)
ans =

    1.2037
    1.4258
    1.8804
```

```
>> sum(diag(A'*A))
ans =

    4.5099
```

```
>> sqrt(sum(diag(A'*A)))
ans =

    2.1236
```

$$Norm_{fro} = sum(diag(A' * A))$$

Matrix Function

For vectors

NORM(V,P) = sum(abs(V).^P)^(1/P).

NORM(V) = norm(V,2).

NORM(V,inf) = max(abs(V)).

NORM(V,-inf) = min(abs(V)).

Matrix Function

```
>> A=linspace(-3,3,7)

A =

   -3    -2    -1     0     1     2     3

>> norm(A,inf)

ans =

     3

>> norm(A,-inf)

ans =

     0

>> norm(A,2)

ans =

    5.2915

>> sum(abs(A).^2)^(1/2)

ans =

    5.2915
```

The find command

The find command can be used to find nonzeros element of any vector or matrix.

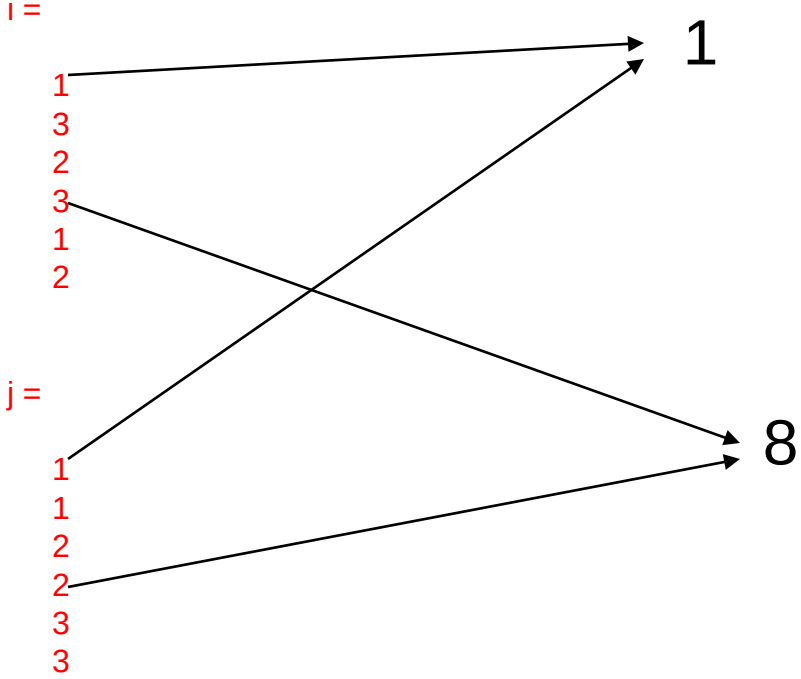
i = find(x) returns the indices of the vector x that are non-zero.

i = find(X) returns the row and column indices of the nonzero entries in the matrix X

The find command

```
>> A=[3 0 5 1 0 7 8]

A =

    3    0    5    1    0    7    8

>> i=find(A)

i =

    1    3    4    6    7
```

```
>> A = [1 0 3 ; 0 5 6 ; 7 8 0]

A =

    1    0    3
    0    5    6
    7    8    0

>> [i,j]=find(A)

i =

    1
    3
    2
    3
    1
    2

j =

    1
    1
    2
    2
    3
    3
```



The find command

```
>> A=[3 0 5 1 0 7 8]

A =

    3    0    5    1    0    7    8

>> i=find(A>=5)

i =

    3    6    7
```

```
>> i=find(A==min(A))

i =

    2    5

>> i=find(A<=3)

i =

    1    2    4    5

>> i=find(A~=0)

i =

    1    3    4    6    7
```

Linear Systems of Equations

Consider the following system of linear equations:

$$\begin{aligned} x + 2y + 3z &= 1 \\ 4x + 5y + 6z &= 1 \\ 7x + 8y &= 1 \end{aligned}$$

This system of three equations is called linear since the powers on all the variables are one and each variable occurs separately (i.e. no xy or xyz terms).

Linear Systems of Equations

To solve a system of equations A*x = b

A is called the "coefficient matrix" for the system of linear equations.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This equation can be solved for x using linear algebra.

The result is $x = A^{-1} * b$.

Linear Systems of Equations

There are typically two ways to solve for x in MATLAB:

The first one is to use the matrix inverse, inv.

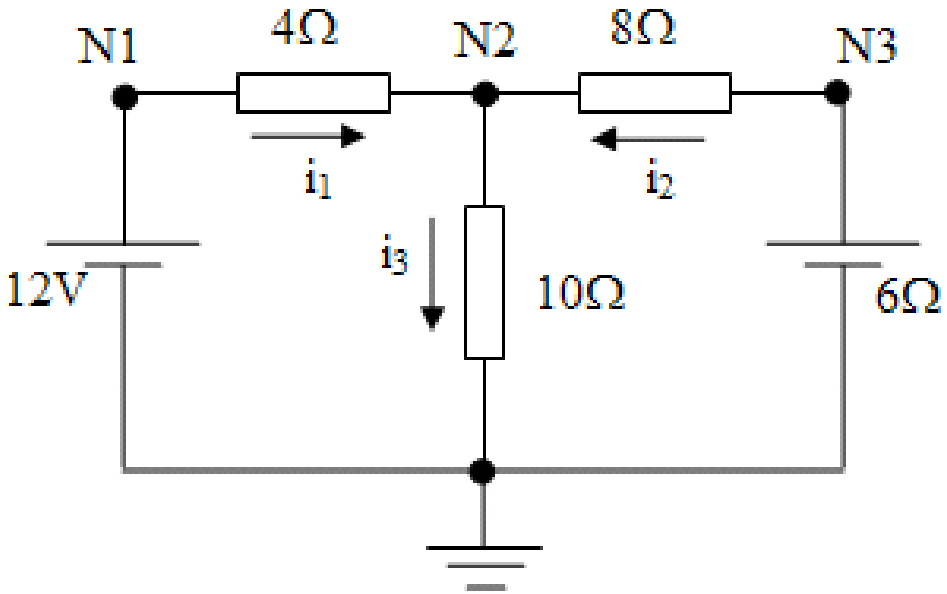
```
>> A = [1 2  3 ; 4 5 6 ; 7 8 0];
>> b=[1;1;1];
>> x=inv(A)*b

x =

   -1.0000
    1.0000
   -0.0000
```

The second one is to use the backslash (\) operator.

```
>> A = [1 2  3 ; 4 5 6 ; 7 8 0];
>> b=[1;1;1];
>> x=A\b

x =

   -1.0000
    1.0000
   -0.0000
```

Linear Systems of Equations

Example: Find i_1,i_2, and i_3 by using Node equation method.



Linear Systems of Equations

Example:

$$4i_1 + 10i_3 = 12$$
$$8i_2 + 10i_3 = 6$$
$$i_1 + i_2 - i_3 = 0$$

$$A = \begin{bmatrix} 4 & 0 & 10 \\ 0 & 8 & 10 \\ 1 & 1 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 12 \\ 6 \\ 0 \end{bmatrix}$$

Linear Systems of Equations

Example:

```
>> A = [4 0 10 ; 0 8 10 ; 1 1 -1];
>> b=[12;6;0];
>> x=inv(A)*b

x =

    1.0263
   -0.2368
    0.7895
```

➔

$$i_1 = 1.0263A$$

$$i_2 = -0.2368A$$

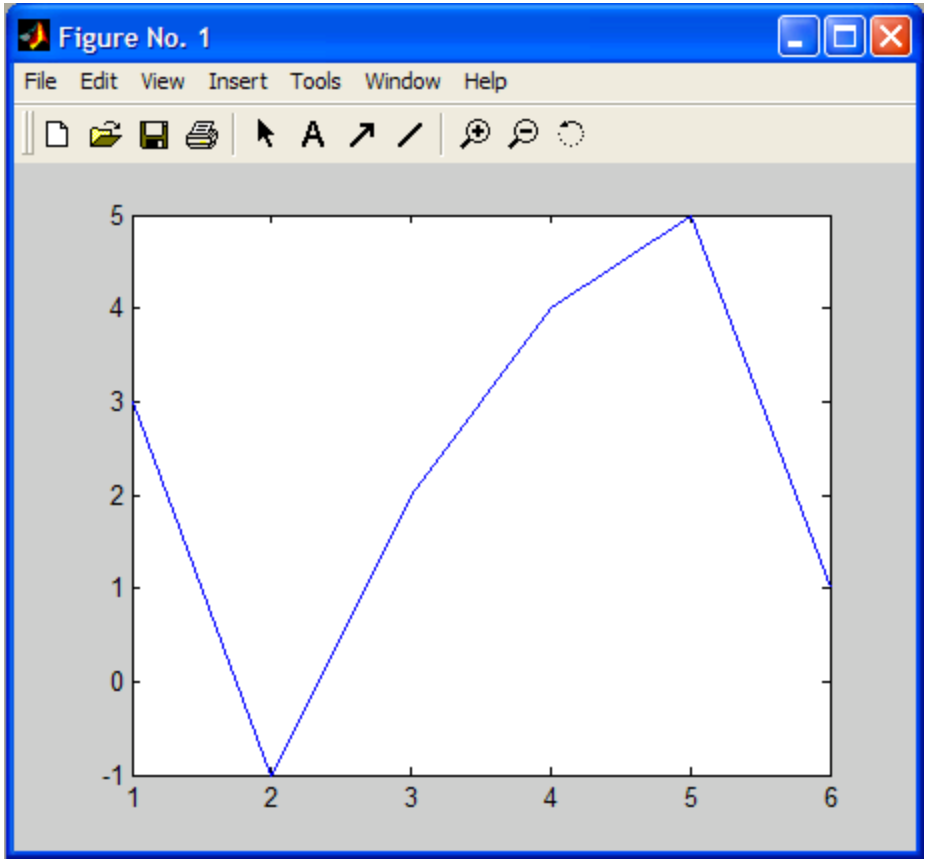
$$i_3 = 0.7895A$$

Creating Simple Plots

The MATLAB command to plot a graph is plot(x,y).

The vectors x = (1 2 3 4 5 6) and y = (3 -1 2 4 5 1) produce

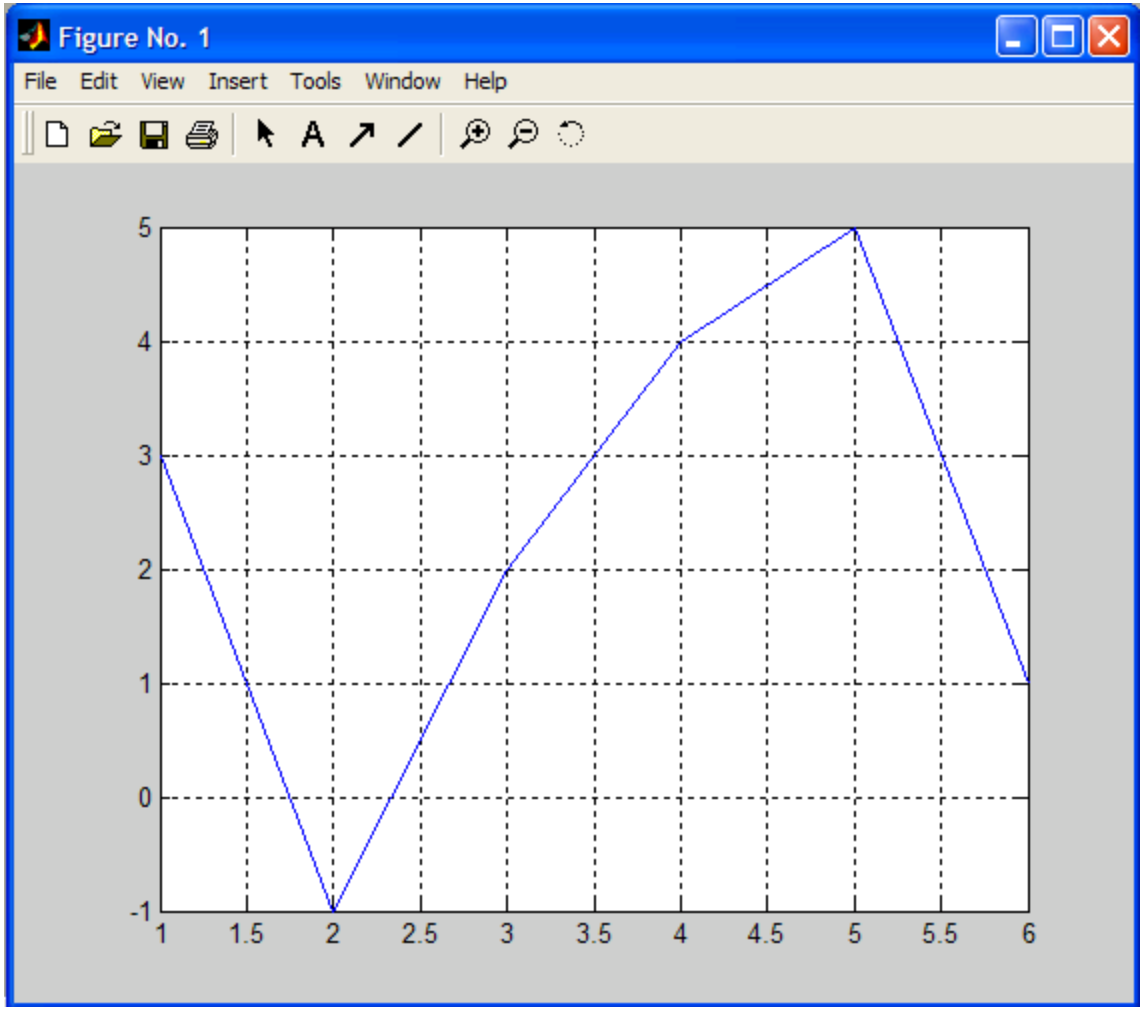
```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```



A figure window opens when a plot is requested

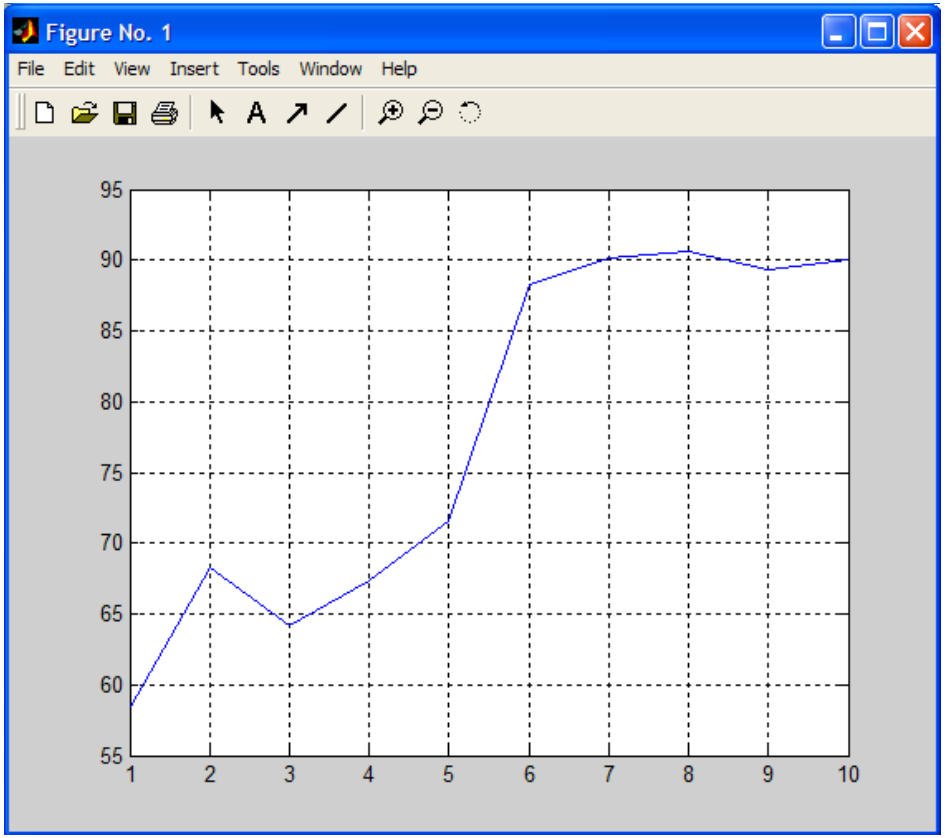
Creating Simple Plots

```
>> grid
```



Creating Simple Plots

```
>> x=[1:10];
>> y=[58.5 68.3 64.2 67.3 71.5 88.3 90.1 90.6 89.3 90];
>> plot(x,y)
>> grid
```



Creating Simple Plots

The plot functions 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 we specify two vectors, as mentioned above, plot(x,y) produces a graph of y versus x.

Creating Simple Plots

For example, to plot the function sin (x) on the interval [0, 2π],

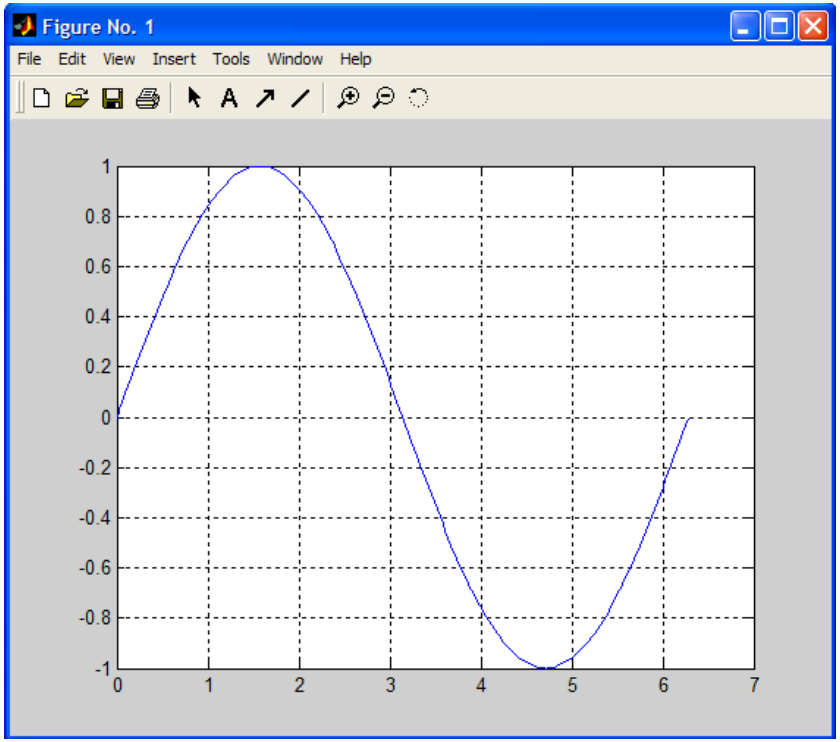
we first create a vector of x values ranging from 0 to 2π, then compute the sine of these values, and finally plot the result:

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
>> grid
```

Creating Simple Plots

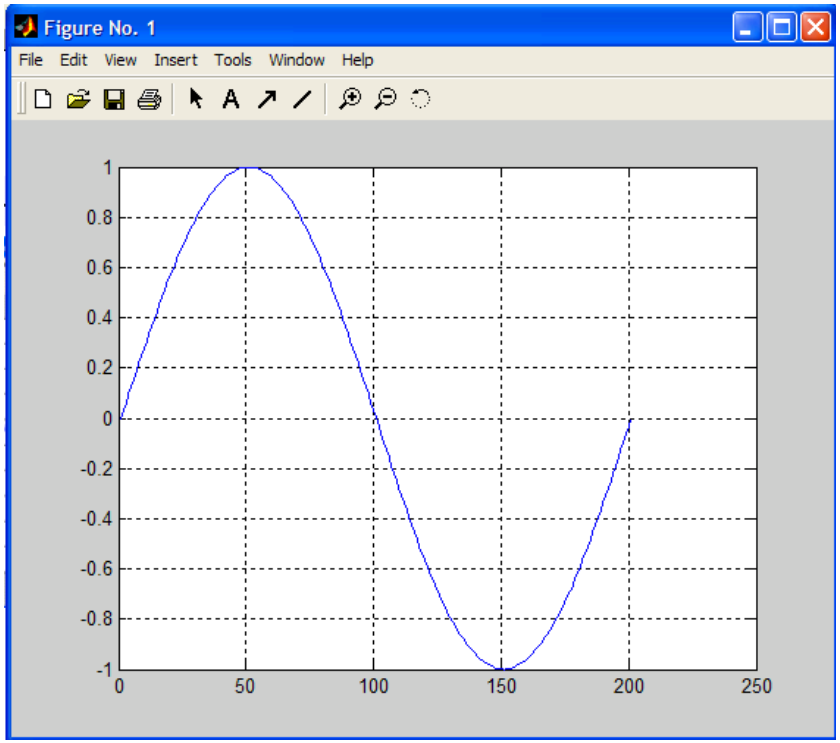
- 0:pi/100:2*pi yields a vector that
  - starts at 0,
  - takes steps (or increments) of π/100,
  - stops when 2π is reached.
- If you omit the increment, MATLAB automatically increments by 1.

Creating Simple Plots




```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
>> grid
```

x → 0-6.28




```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(y)
>> grid
```

x → 1-201

Example

1) create a vector x, with 50 equally spaced entries from 0 to 1.0

```
>> x = linspace(0,1,50);
```

2) create a vector f, with entries of cosine evaluated at entries of 2*pi*x

```
>> f = cos(2*pi*x);
```

3) plot f as a function of x, using + symbols

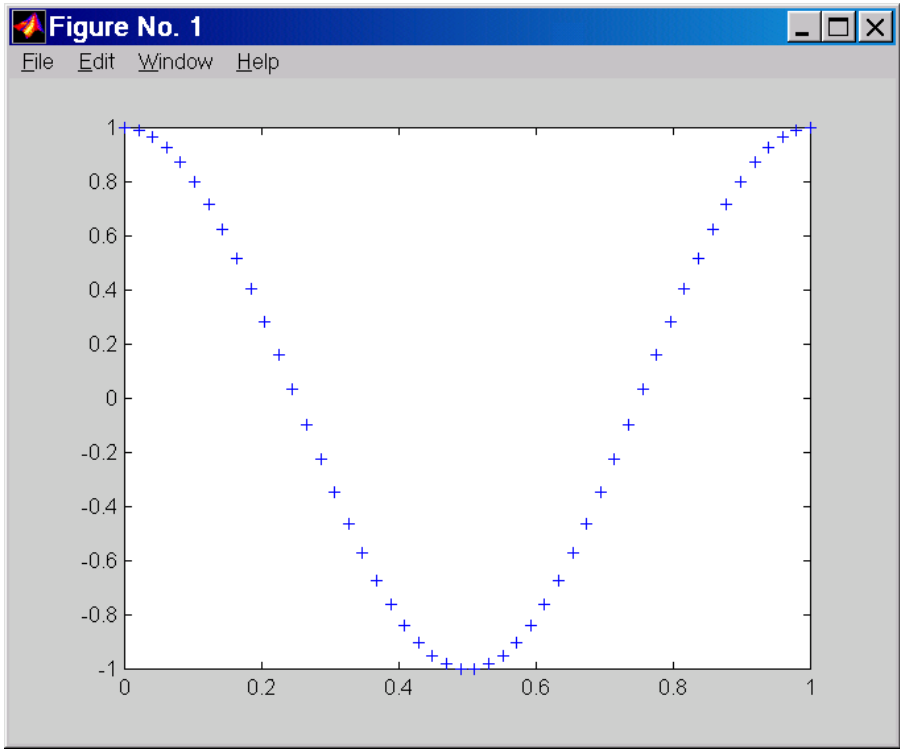
```
>>  plot(x,f,'+')
```

Example

MATLAB Command Window

File Edit Window Help

```
» x = linspace( 0, 1, 50)';
» f  = cos(2*pi*x);
» plot( x,  f, '+');
»
```



Example

1) create a vector x, with 50 equally spaced entries from 0 to 1.0

```
>> x = linspace(0,1,50);
```

2) create a vector f, with entries of cos evaluated at entries of x

```
>> f = cos(2*pi*x);
```

3) plot f as a function of x, using a blue line

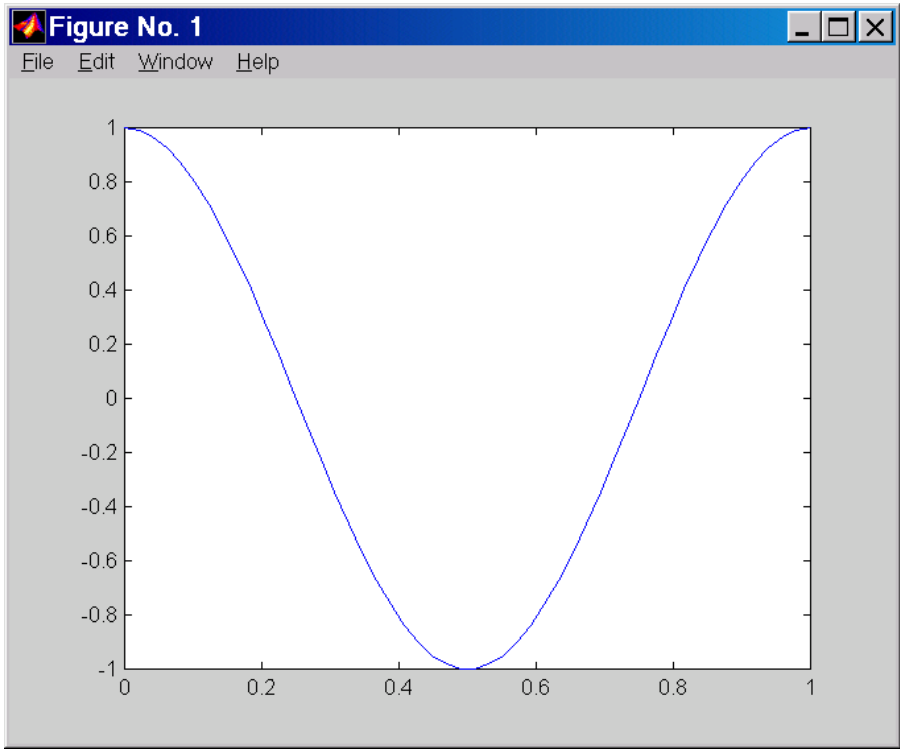
```
>>  plot(x,f,'b')
```

Example

MATLAB Command Window

File Edit Window Help

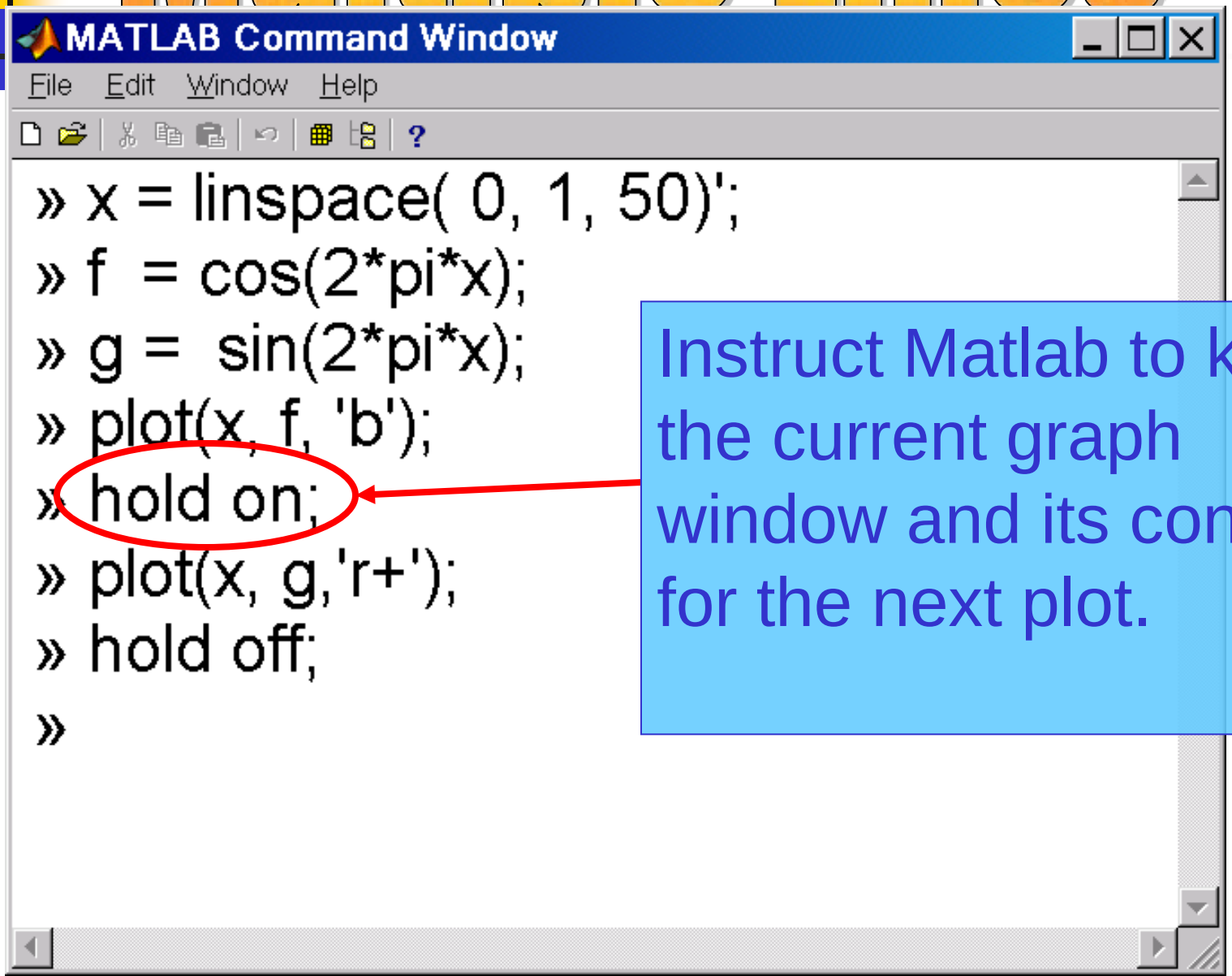
```
» x = linspace( 0, 1, 50)';
» f  = cos(2*pi*x);
» plot( x,  f, 'b');
»
```



Multiple Lines

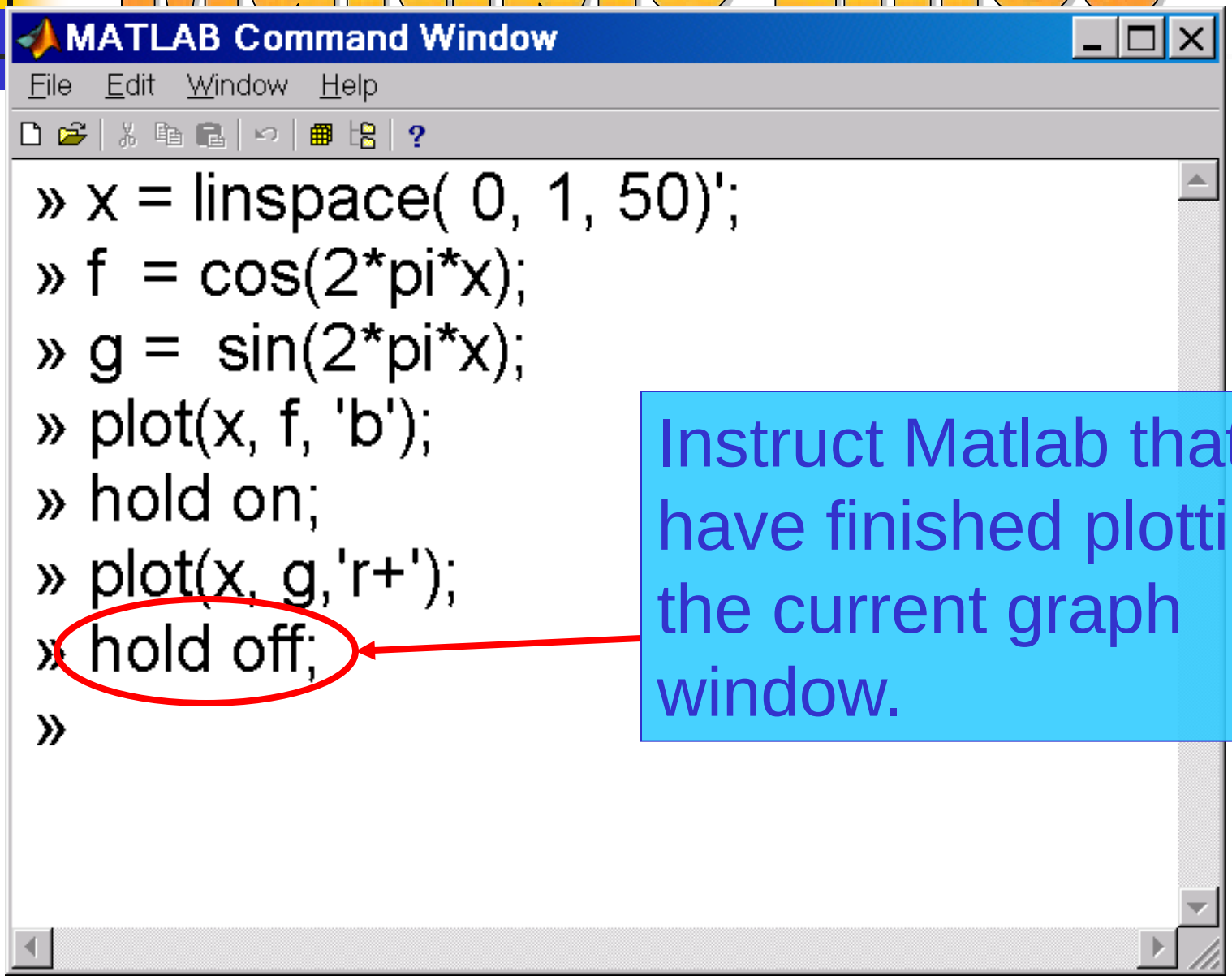
- The figure window is erased every time a new graph is drawn
- Use the hold on command draw an additional line onto an existing graph

Multiple Lines



Instruct Matlab to keep the current graph window and its contents for the next plot.

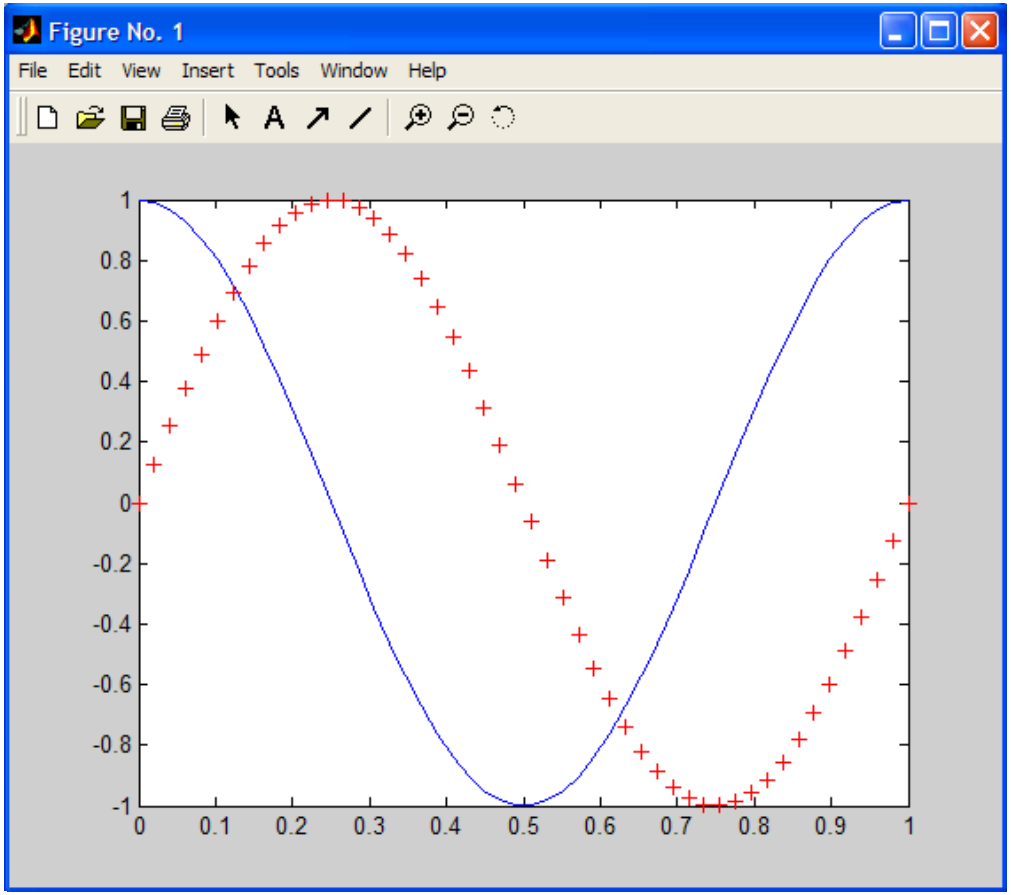
Multiple Lines



Instruct Matlab that you have finished plotting in the current graph window.

Multiple Lines

```
>> x = linspace(0,1,50);
>> f = cos(2*pi*x);
>> g=sin(2*pi*x);
>> plot(x,f,'b')
>> hold on
>> plot(x,g,'r+')
>> hold off
```



MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Bytes
x	<1x201 double>	1x201	16
y1	<1x201 double>	1x201	16
y2	<1x201 double>	1x201	16

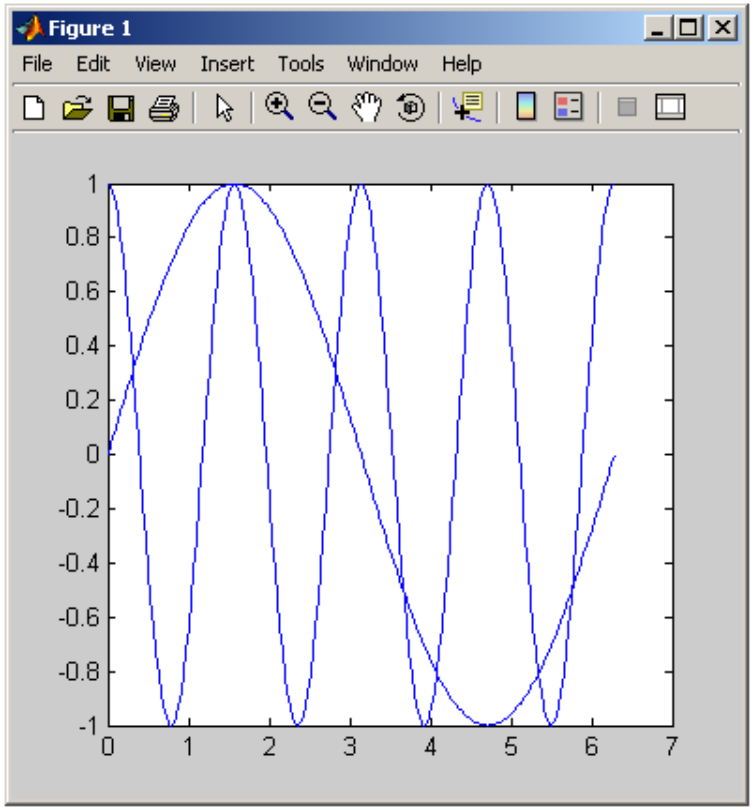
Current Directory | Workspace

Command History

```
xlabel('Trial')
ylabel('Distance, ft')
grid on
x = 0:pi/100:2*pi;
y1 = cos(x*4);
plot(x,y1)
y2 = sin(x);
hold on;
plot(x, y2)
clear, clc
```

Command Window

```
>> x = 0:pi/100:2*pi;
>> y1 = cos(x*4);
>> plot(x,y1)
>> y2 = sin(x);
>> hold on;
>> plot(x, y2)
>>
```



Start

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Bytes
x	<1x201 double>	1x201	16
y1	<1x201 double>	1x201	16
y2	<1x201 double>	1x201	16

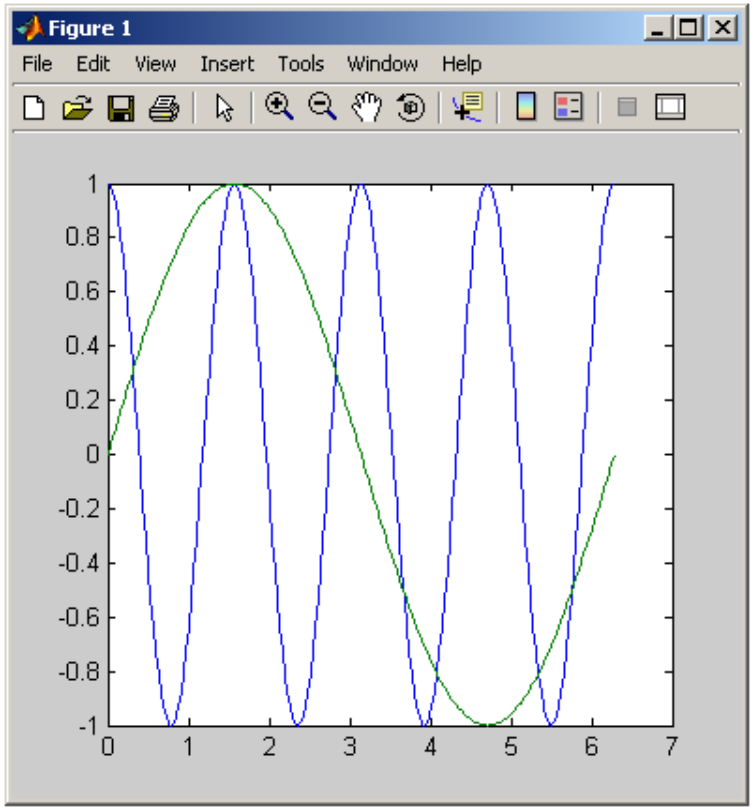
Current Directory | Workspace

Command History

```
y1 = cos(x*4);
y2 = sin(x);
plot(x,y1,x,y2)
plot(x,[y1,y2])
clear, clc
x = 0:pi/100:2*pi;
y1 = cos(x*4);
y2 = sin(x);
plot(x,y1,x,y2)
plot(x,[y1;y2])
```

Command Window

```
>> x = 0:pi/100:2*pi;
>> y1 = cos(x*4);
>> y2 = sin(x);
>> plot(x,y1,x,y2)
>> plot(x,[y1;y2])
>>
```



Start

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Bytes
X	<1x201 double>	1x201	16
Y1	<1x201 double>	1x201	16
Y2	<1x201 double>	1x201	16
Y3	<1x201 double>	1x201	16
Y4	<1x201 double>	1x201	16
Z	<4x201 double>	4x201	64

Current Directory | Workspace

Command History

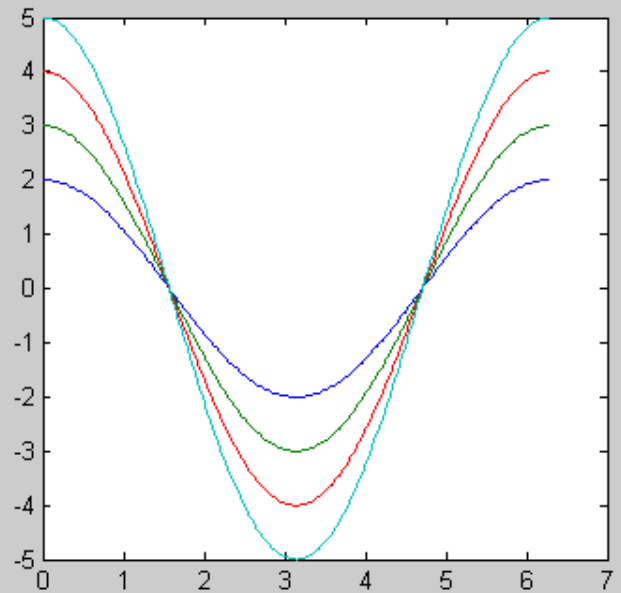
```
Y4 = cos(X)*5;
Z = [Y1; Y2; Y3; Y4];
clear, clc
X = 0:pi/100:2*pi;
Y1 = cos(X)*2;
Y2 = cos(X)*3;
Y3 = cos(X)*4;
Y4 = cos(X)*5;
Z = [Y1; Y2; Y3; Y4];
plot(X,Z)
```

Command Window

```
>> X = 0:pi/100:2*pi;
>> Y1 = cos(X)*2;
>> Y2 = cos(X)*3;
>> Y3 = cos(X)*4;
>> Y4 = cos(X)*5;
>> Z = [Y1; Y2; Y3; Y4];
>> plot(X,Z)
>>
```

Figure 1

File Edit View Insert Tools Window Help



Start 7:12 PM

Line, Color and Mark Style

y yellow	. point	- solid
m magenta	o circle	: dotted
c cyan	x x-mark	-. dashdot
r red	+ plus	-- dashed
g green	* star	
b blue	s square	
w white	d diamond	
k black	v triangle (down)	
	^ triangle (up)	
	< triangle (left)	
	> triangle (right)	
	p pentagram	
	h hexagram	

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Byte:
x	[1 2 3 4 5 6 7 8 ...	1x10	
y	[58.5 63.8 64.2 ...	1x10	

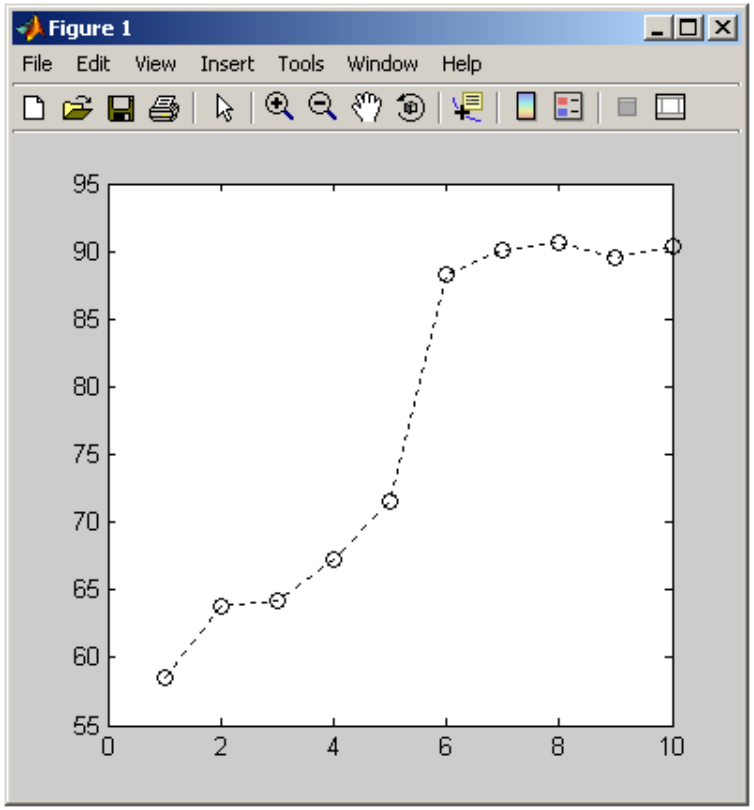
Current Directory | Workspace

Command History

```
z = [Y1; Y2; Y3; Y4];
plot(X,Z)
clear, clc
plot(peaks(100))
clear, clc
help plot
clear, clc
x = [1:10];
y = [ 58.5, 63.8, 64.2, 67.3
plot(x,y,':ok')
```

Command Window

```
>> x = [1:10];
>> y = [ 58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89
>> plot(x,y,':ok')
>>
```



Start

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Byte:
x	[1 2 3 4 5 6 7 8 ...	1x10	
y	[58.5 63.8 64.2 ...	1x10	

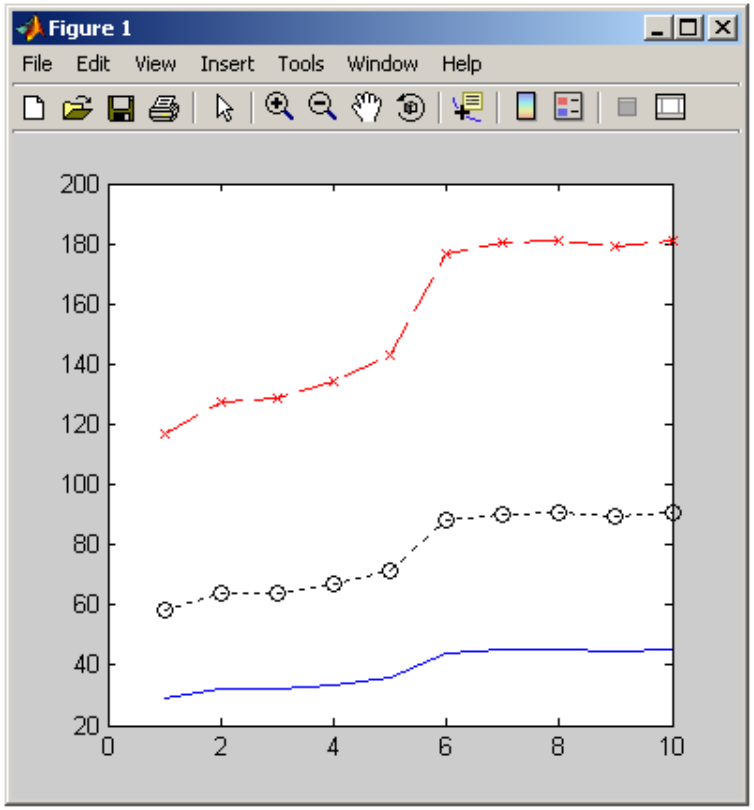
Current Directory | Workspace

Command History

```
plot(x,2)
clear, clc
plot(peaks(100))
clear, clc
help plot
clear, clc
x = [1:10];
y = [ 58.5, 63.8, 64.2, 67.3
plot(x,y,':ok')
plot(x,y,':ok',x,y*2,'--xr'
```

Command Window

```
>> x = [1:10];
>> y = [ 58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89
>> plot(x,y,':ok')
>> plot(x,y,':ok',x,y*2,'--xr',x,y/2,'-b')
>>
```



Start

Title, Labeling Axes and Legends

- Set up the legend box with descriptions of each curve:

  ```
  >> legend('f=cos(2*pi*x)', 'g=sin(2*pi*x)');
  ```

- Label the x-axis

  ```
  >> xlabel('x');
  ```

- Label the y-axis

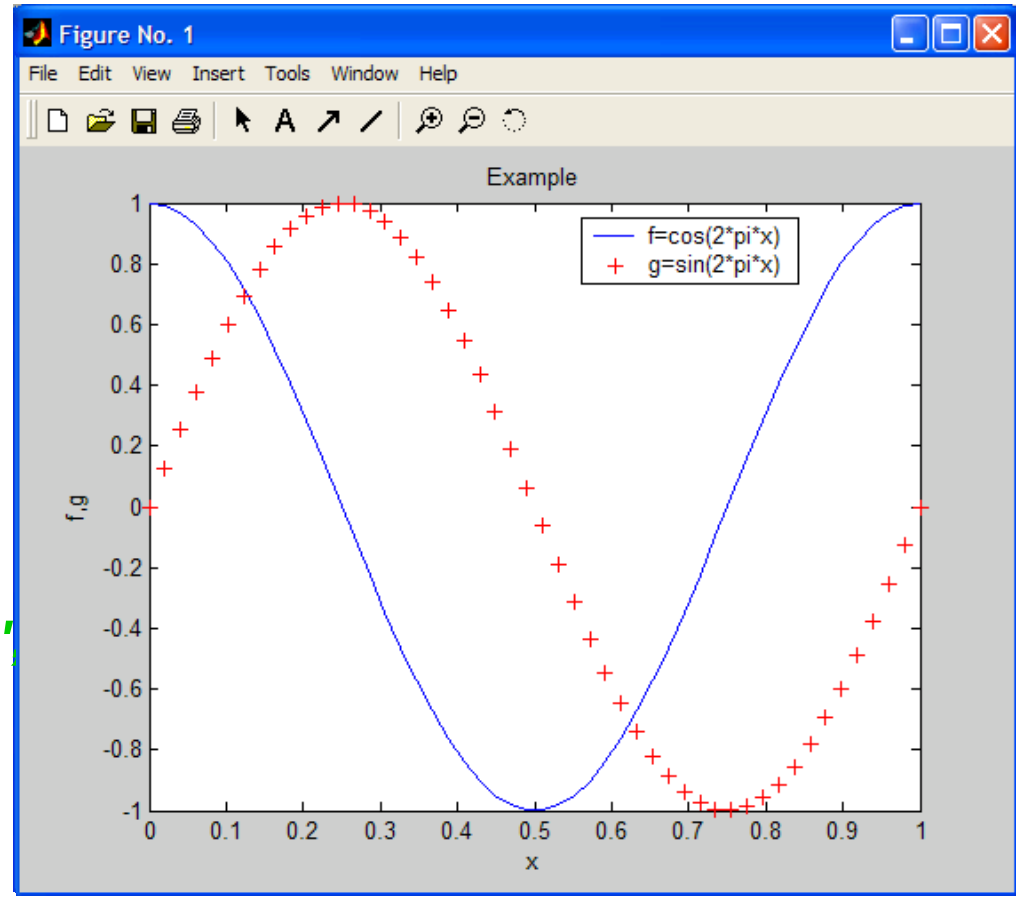
  ```
  >> ylabel('f,g');
  ```

- Give the graph a title

  ```
  >> title('Example');
  ```

Title, Labeling Axes and Legends

```
>> x = linspace(0,1,50);
>> f = cos(2*pi*x);
>> g=sin(2*pi*x);
>> plot(x,f,'b')
>> hold on
>> plot(x,g,'r+')
>> hold off
>> legend('f=cos(2*pi*x)'
>> xlabel('x')
>> ylabel('f,g')
>> title('Example')
```



MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Stack: Base

Name	Value	Size	Byte:
x	[1 2 3 4 5 6 7 8 ...	1x10	
y	[58.5 63.8 64.2 ...	1x10	

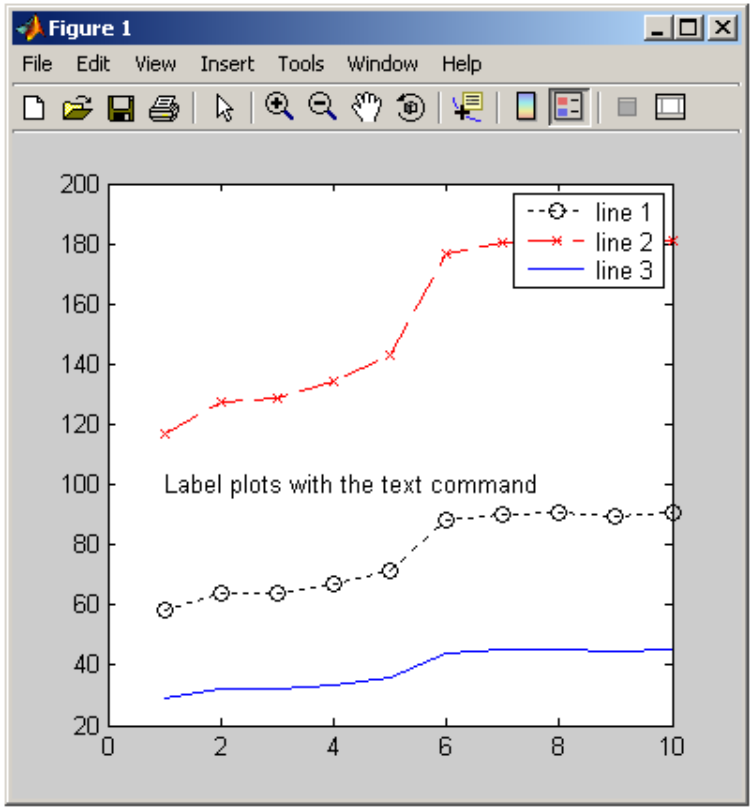
Current Directory Workspace

Command History

```
plot(peaks(100))
clear, clc
help plot
clear, clc
x = [1:10];
y = [ 58.5, 63.8, 64.2, 67.3
plot(x,y,':ok')
plot(x,y,':ok',x,y*2,'--xr',
legend('line 1', 'line 2',
text(1,100, 'Label plots wit
```

Command Window

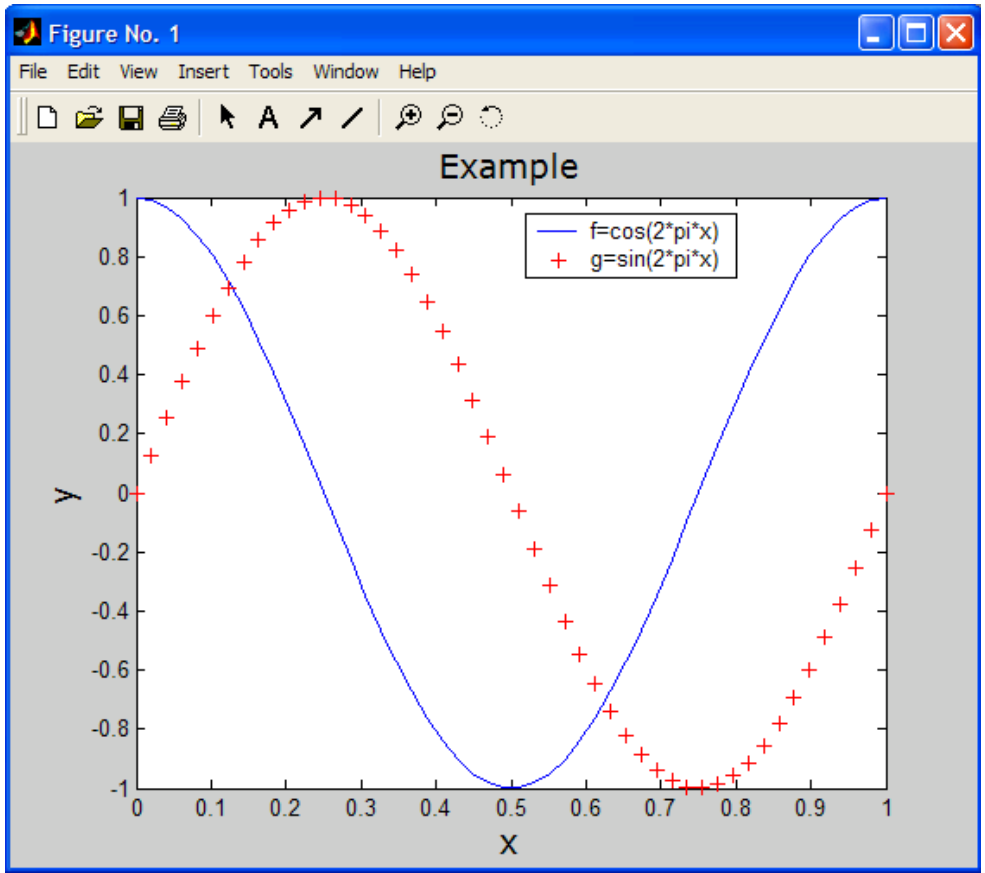
```
>> x = [1:10];
>> y = [ 58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89
>> plot(x,y,':ok')
>> plot(x,y,':ok',x,y*2,'--xr',x,y/2,'-b')
>> legend('line 1', 'line 2', 'line 3')
>> text(1,100, 'Label plots with the text command')
>>
```



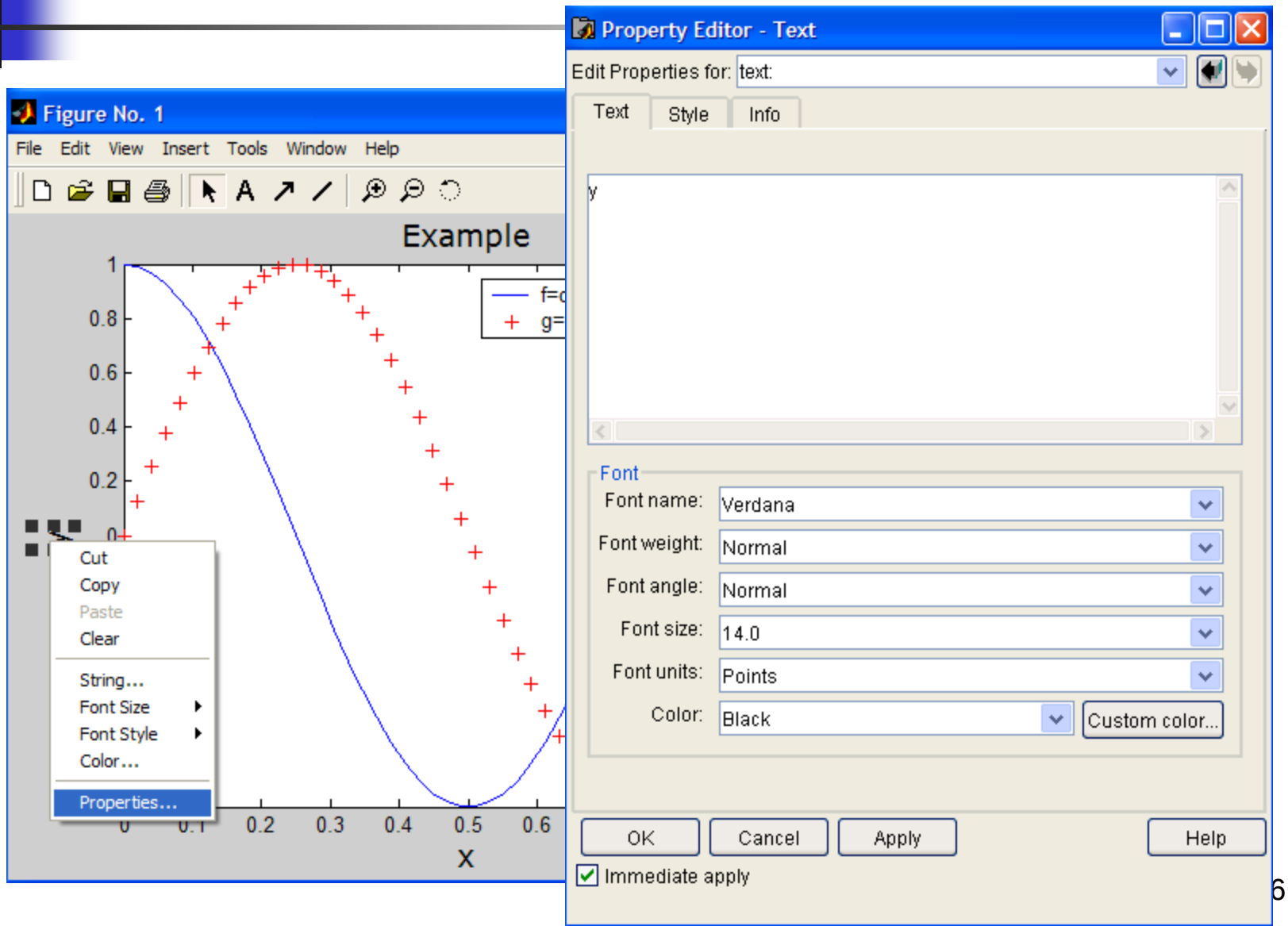
Start

Specifiy the fonts and colors

```
>> xlabel('x','FontSize',14,'FontName','Verdana');
>> ylabel('y','FontSize',14,'FontName','Verdana');
>> title('Example','FontSize',14,'FontName','Verdana')
```



Specifiy the fonts and colors



Individual Exercise

- Create two curves on the same graph:

  F = 1/(1+x^2)

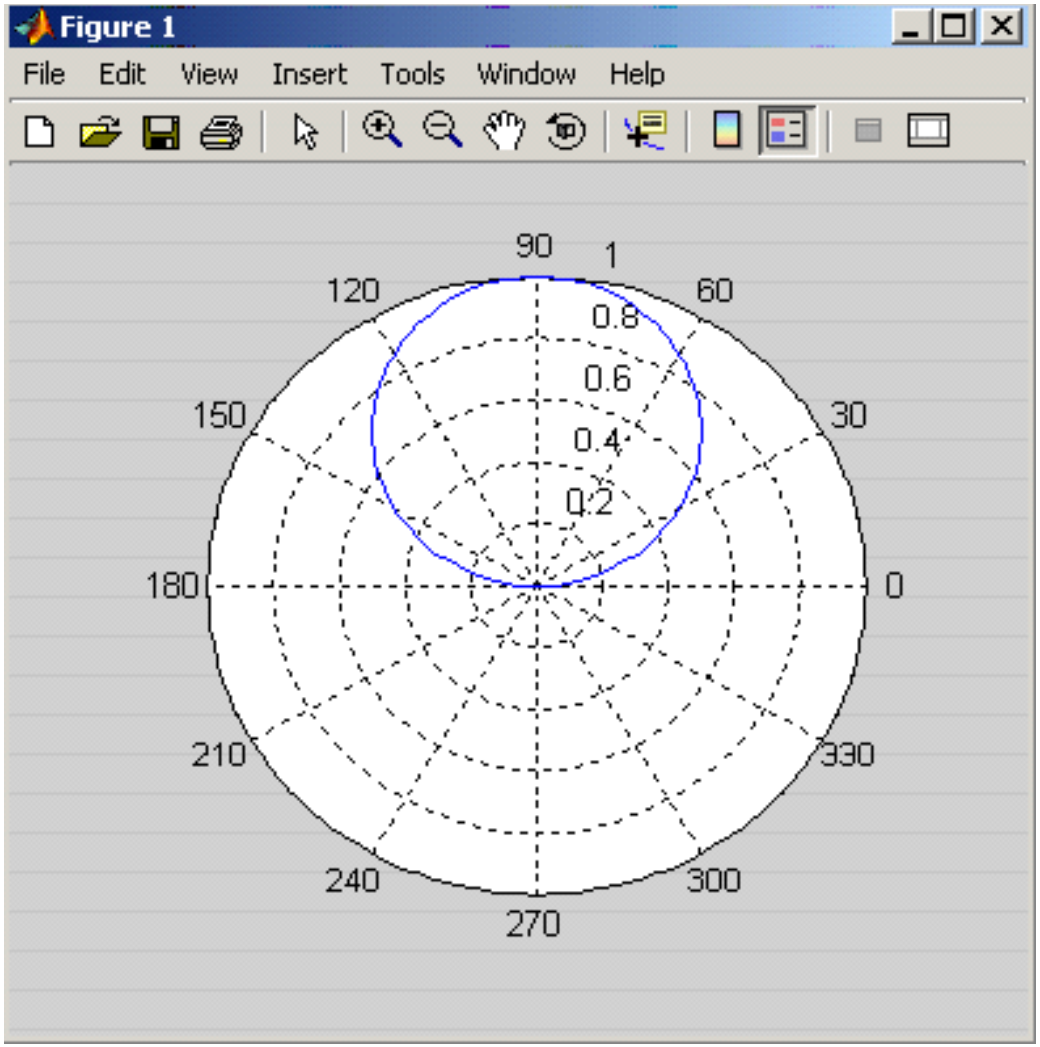
  G = exp(x^3)
- Plot the points at 33 points equally spaced between 0 and 1
- Use black *'s for F and green o's for G
- Label the horizontal and vertical axes
- Create a title (including your name)
- Create a legend

Other kinds of 2-D plots

- Polar
- Pie
- Bar
- Stem

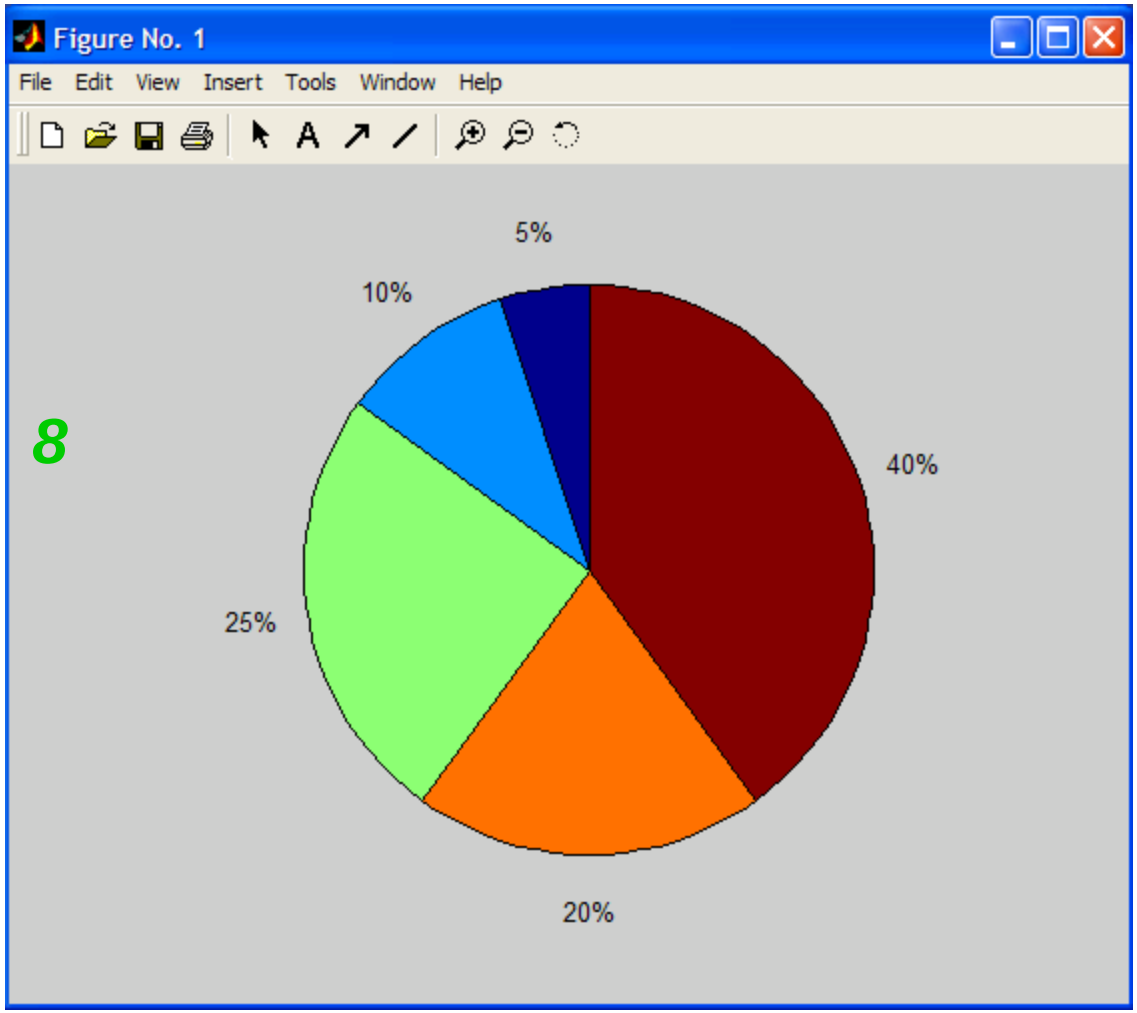
Polar

```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> polar (x,y)
```



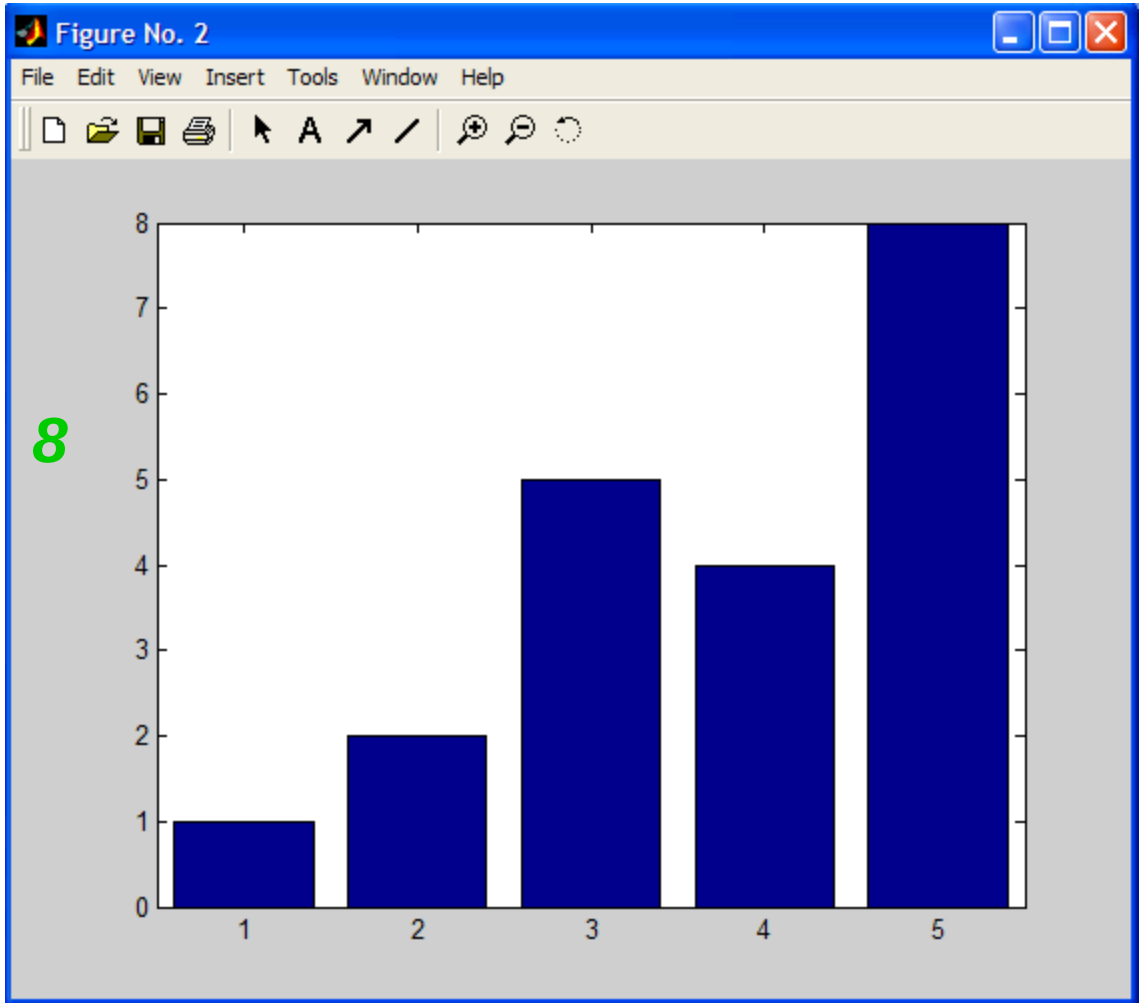
Pie

```
>> A=[1 2 5 4 8]

A =

     1     2     5     4     8

>> pie(A)
```



Bar

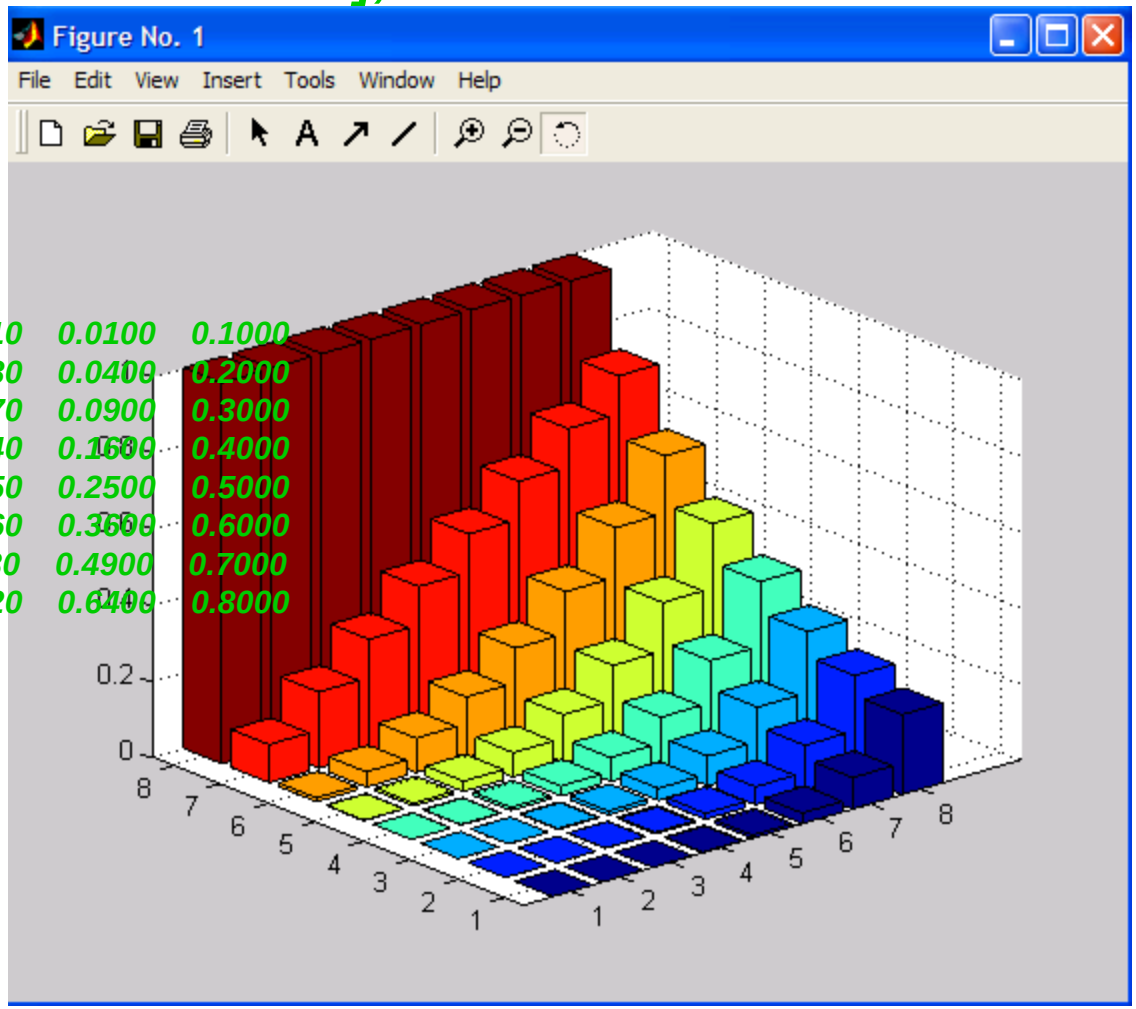
```
>> A=[1 2 5 4 8]

A =

     1     2     5     4     8

>> pie(A)
>> figure
>> bar(A)
```



3-D Bar plots

```
>> v=[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8];
>> A=vander(v);
>> bar3(A)
```

```
A =

  Columns 1 through 7

    0.0000    0.0000    0.0000    0.0001    0.0010    0.0100    0.1000
    0.0000    0.0001    0.0003    0.0016    0.0080    0.0400    0.2000
    0.0002    0.0007    0.0024    0.0081    0.0270    0.0900    0.3000
    0.0016    0.0041    0.0102    0.0256    0.0640    0.1600    0.4000
    0.0078    0.0156    0.0313    0.0625    0.1250    0.2500    0.5000
    0.0280    0.0467    0.0778    0.1296    0.2160    0.3600    0.6000
    0.0824    0.1176    0.1681    0.2401    0.3430    0.4900    0.7000
    0.2097    0.2621    0.3277    0.4096    0.5120    0.6400    0.8000

  Column 8

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

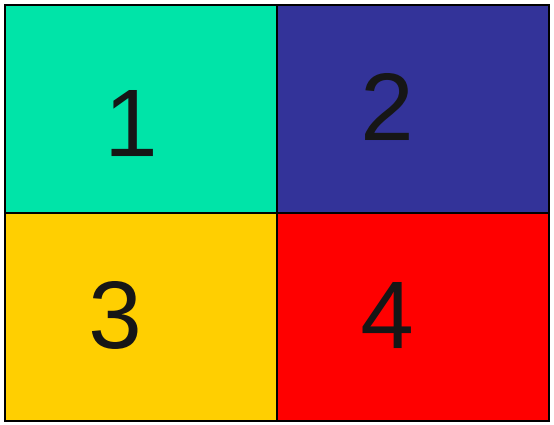


Logarithmic plots

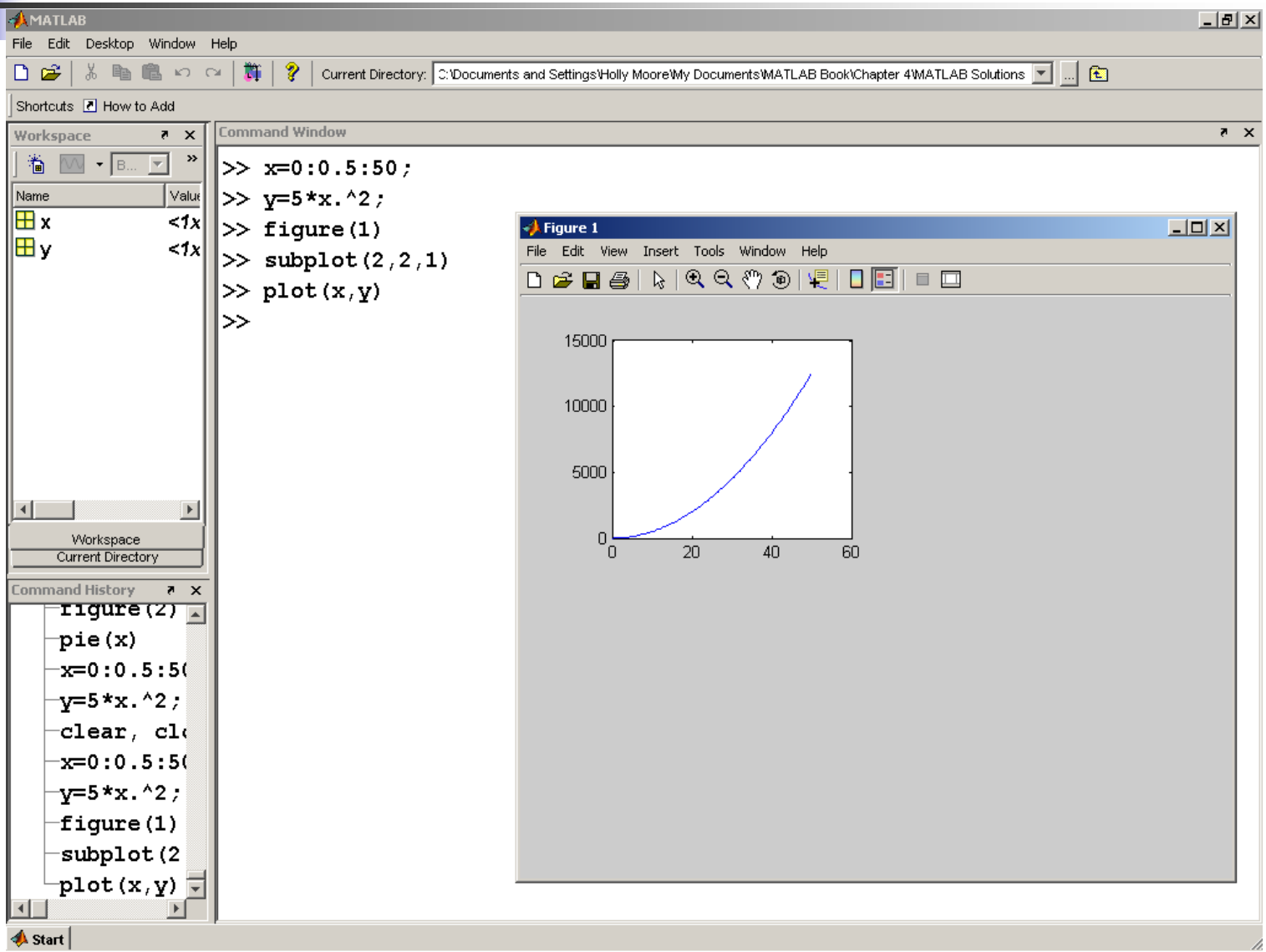
- semilogx
- semilogy
- loglog

Subplots

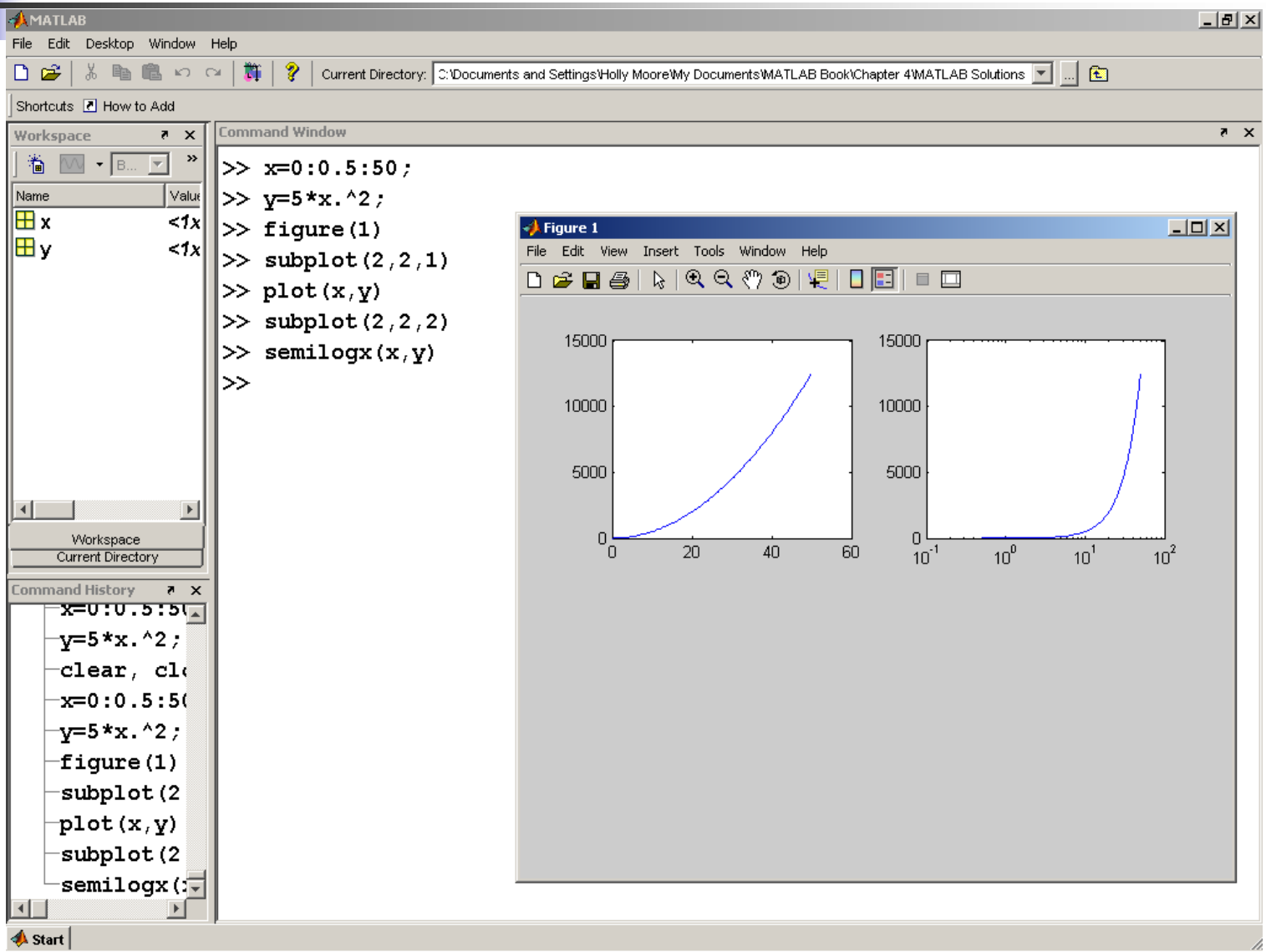
- subplot(2,2,1)



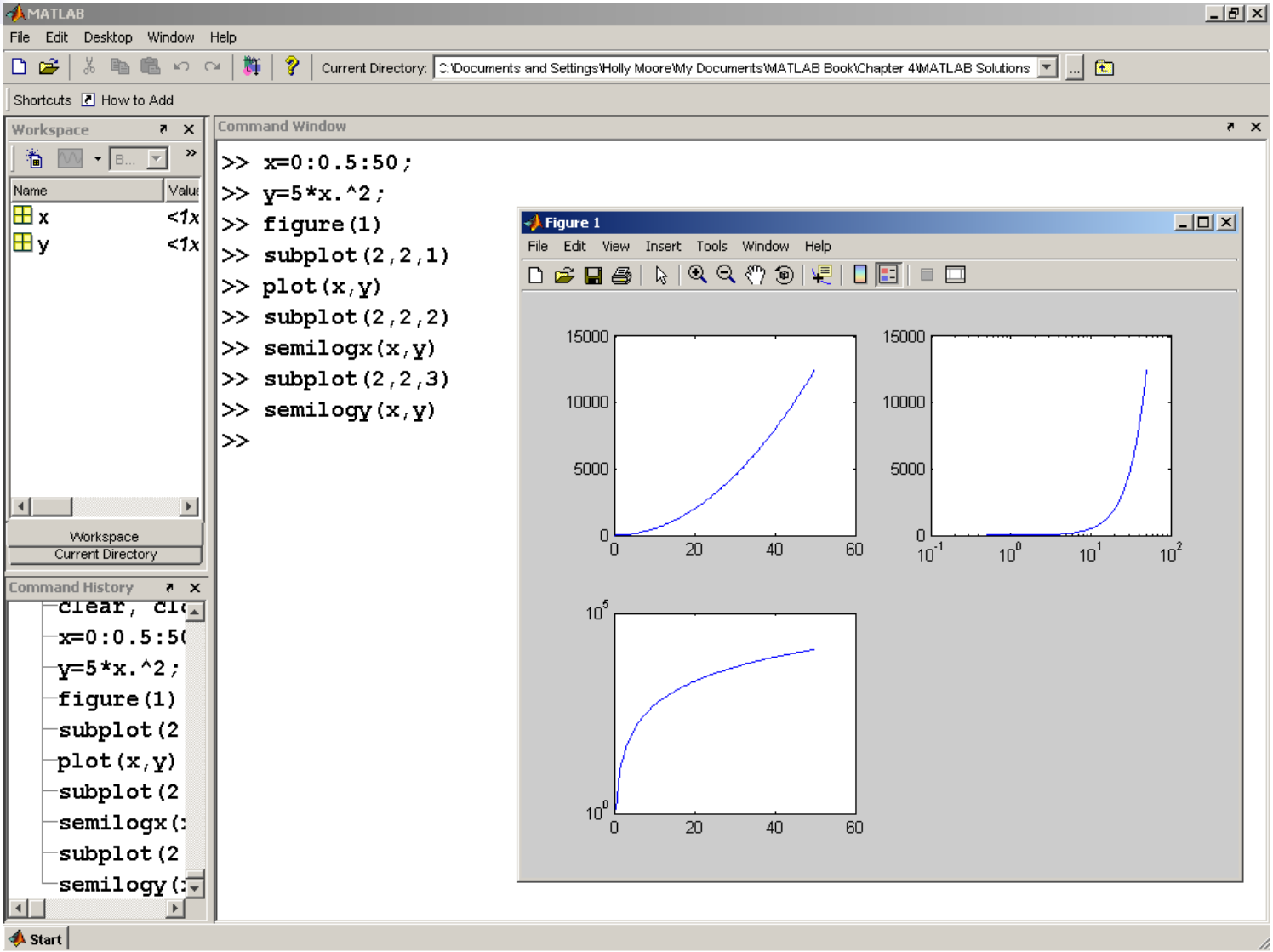
Subplots


MATLAB
File Edit Desktop Window Help
Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions
Shortcuts How to Add
Workspace
Name Value
x <1x
y <1x
Workspace
Current Directory
Command History
figure(2)
pie(x)
x=0:0.5:5(
y=5*x.^2;
clear, cl(
x=0:0.5:5(
y=5*x.^2;
figure(1)
subplot(2
plot(x,y)
Command Window
>> x=0:0.5:50;
>> y=5*x.^2;
>> figure(1)
>> subplot(2,2,1)
>> plot(x,y)
>>
Figure 1
File Edit View Insert Tools Window Help
15000
10000
5000
0
0
20
40
60
Start

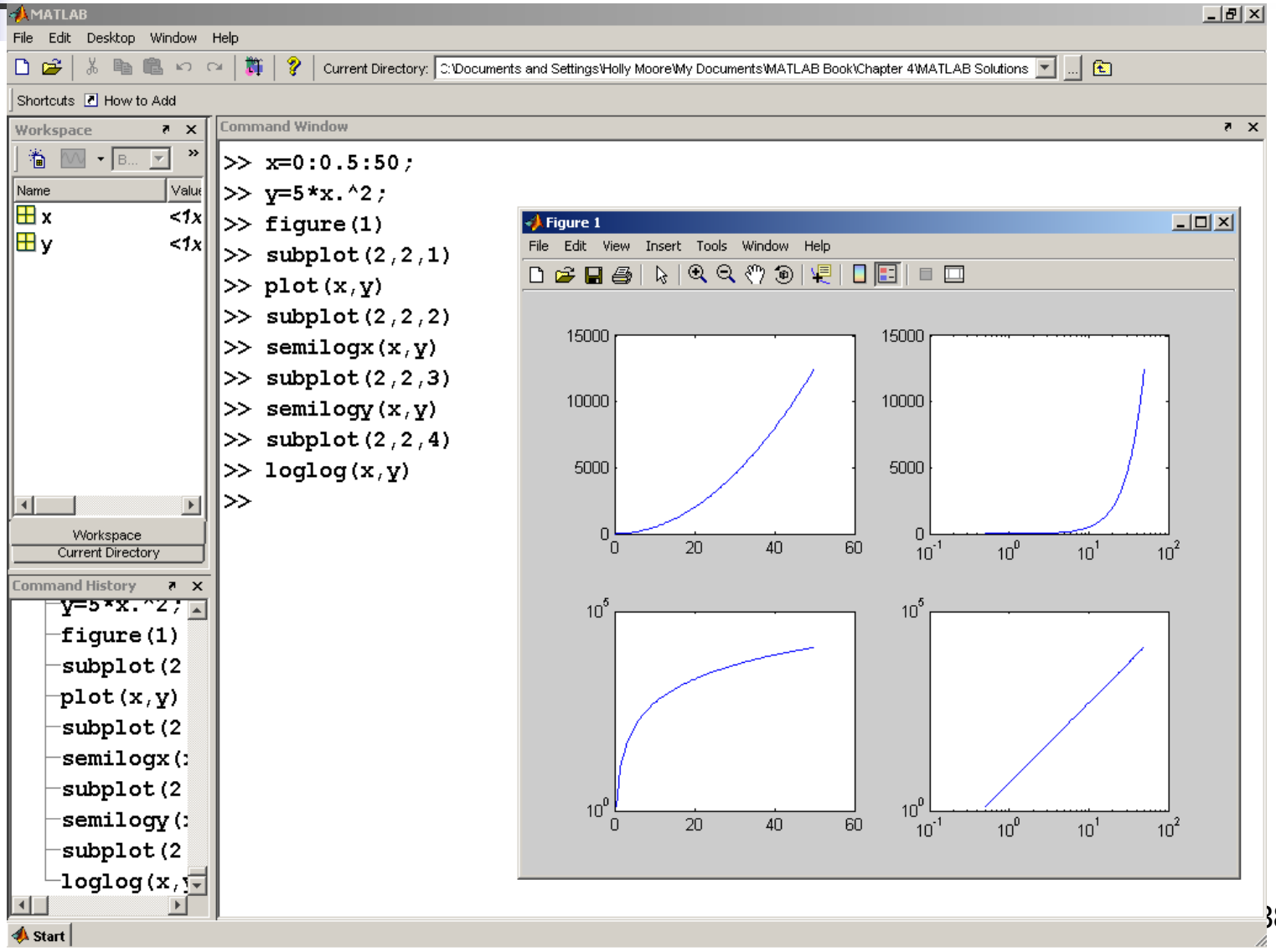
Subplots


MATLAB
File Edit Desktop Window Help
Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions
Shortcuts How to Add
Workspace
Name
x
y
Command Window
>> x=0:0.5:50;
>> y=5*x.^2;
>> figure(1)
>> subplot(2,2,1)
>> plot(x,y)
>> subplot(2,2,2)
>> semilogx(x,y)
>>
Command History
y=5*x.^2;
clear, cl
x=0:0.5:5(
y=5*x.^2;
figure(1)
subplot(2
plot(x,y)
subplot(2
semilogx(:
Figure 1
File Edit View Insert Tools Window Help
15000
10000
5000
0
0 20 40 60
15000
10000
5000
0
10^-1 10^0 10^1 10^2
Start

Subplots


MATLAB
File Edit Desktop Window Help
Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions
Shortcuts How to Add
Workspace
Name
x
y
Workspace
Current Directory
Command History
clear, cl
x=0:0.5:5(
y=5*x.^2;
figure(1)
subplot(2
plot(x,y)
subplot(2
semilogx(:
subplot(2
semilogy(:
Command Window
>> x=0:0.5:50;
>> y=5*x.^2;
>> figure(1)
>> subplot(2,2,1)
>> plot(x,y)
>> subplot(2,2,2)
>> semilogx(x,y)
>> subplot(2,2,3)
>> semilogy(x,y)
>>
Figure 1
File Edit View Insert Tools Window Help
15000
10000
5000
0
0 20 40 60
15000
10000
5000
0
10^-1 10^0 10^1 10^2
10^5
10^0
0 20 40 60
Start

Subplots


MATLAB
File Edit Desktop Window Help
Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions
Shortcuts How to Add
Workspace
Name
x
y
Workspace
Current Directory
Command History
figure(1)
subplot(2
plot(x,y)
subplot(2
semilogx(
subplot(2
semilogy(
subplot(2
loglog(x,
Command Window
>> x=0:0.5:50;
>> y=5*x.^2;
>> figure(1)
>> subplot(2,2,1)
>> plot(x,y)
>> subplot(2,2,2)
>> semilogx(x,y)
>> subplot(2,2,3)
>> semilogy(x,y)
>> subplot(2,2,4)
>> loglog(x,y)
>>
Figure 1
File Edit View Insert Tools Window Help
15000
10000
5000
0
0 20 40 60
15000
10000
5000
0
10^-1 10^0 10^1 10^2
10^5
10^0
0 20 40 60
10^5
10^0
10^-1 10^0 10^1 10^2
Start

Histograms

- Organizes data into “bins’
- Default is 10
- hist(x)
- hist(x, 25) would create 25 bins

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Name Value

A <1x

Workspace

Current Directory

Command History

semilogy(

subplot(2

loglog(x,

clear, cl

randn(1,1(

clear,clc

randn(1,1(

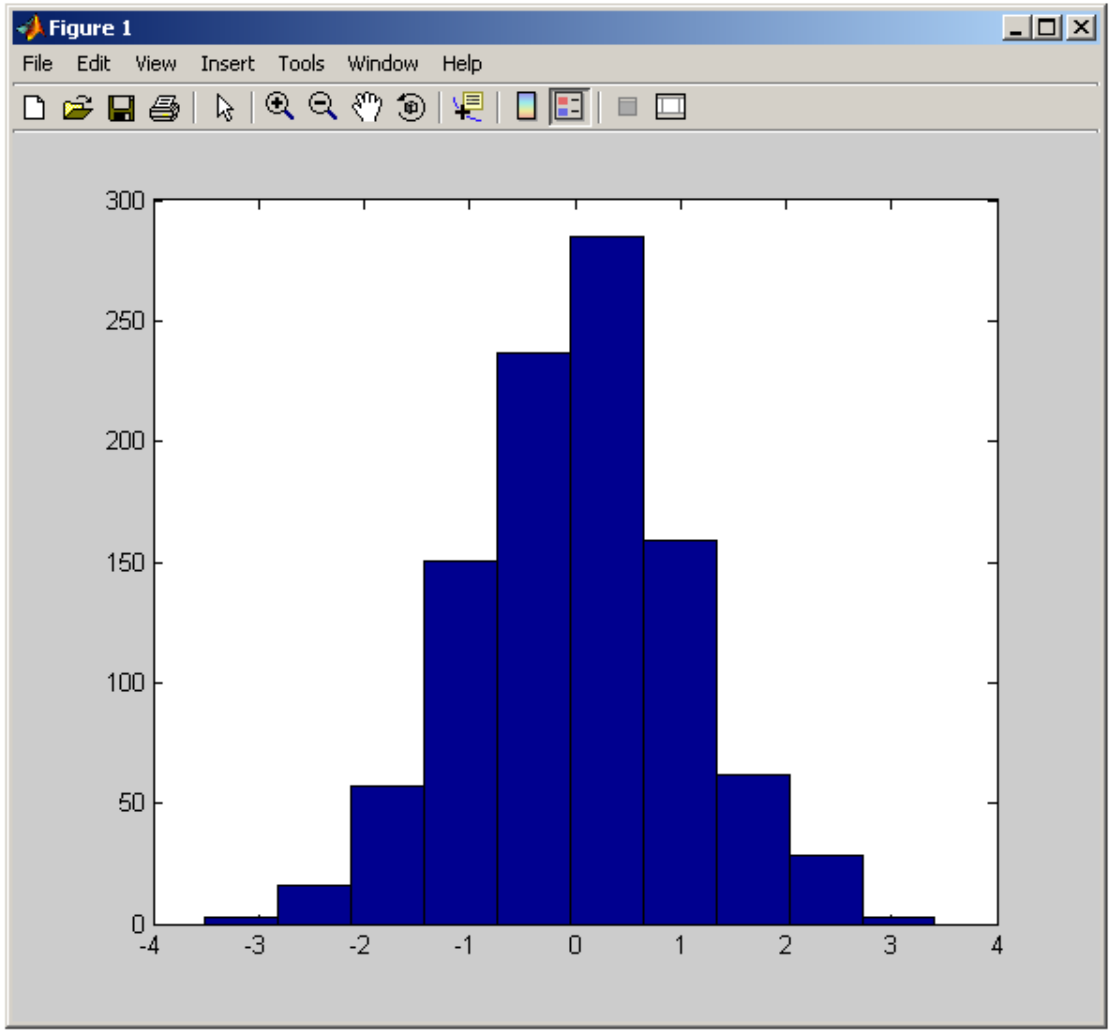
clear, cl

A=randn(1

hist(A)

Command Window

```
>> A=randn(1,1000);
>> hist(A)
>>
```



Start

Three dimensional plotting

- Line Plots
  - plot3
- Surface Plots
  - mesh
  - surf

MATLAB

File Edit Desktop Window Help

Current Directory: C:\Documents and Settings\Holly Moore\My Documents\MATLAB Book\Chapter 4\MATLAB Solutions

Shortcuts How to Add

Workspace

Name	Value
x	<1x
y	<1x
z	<1x

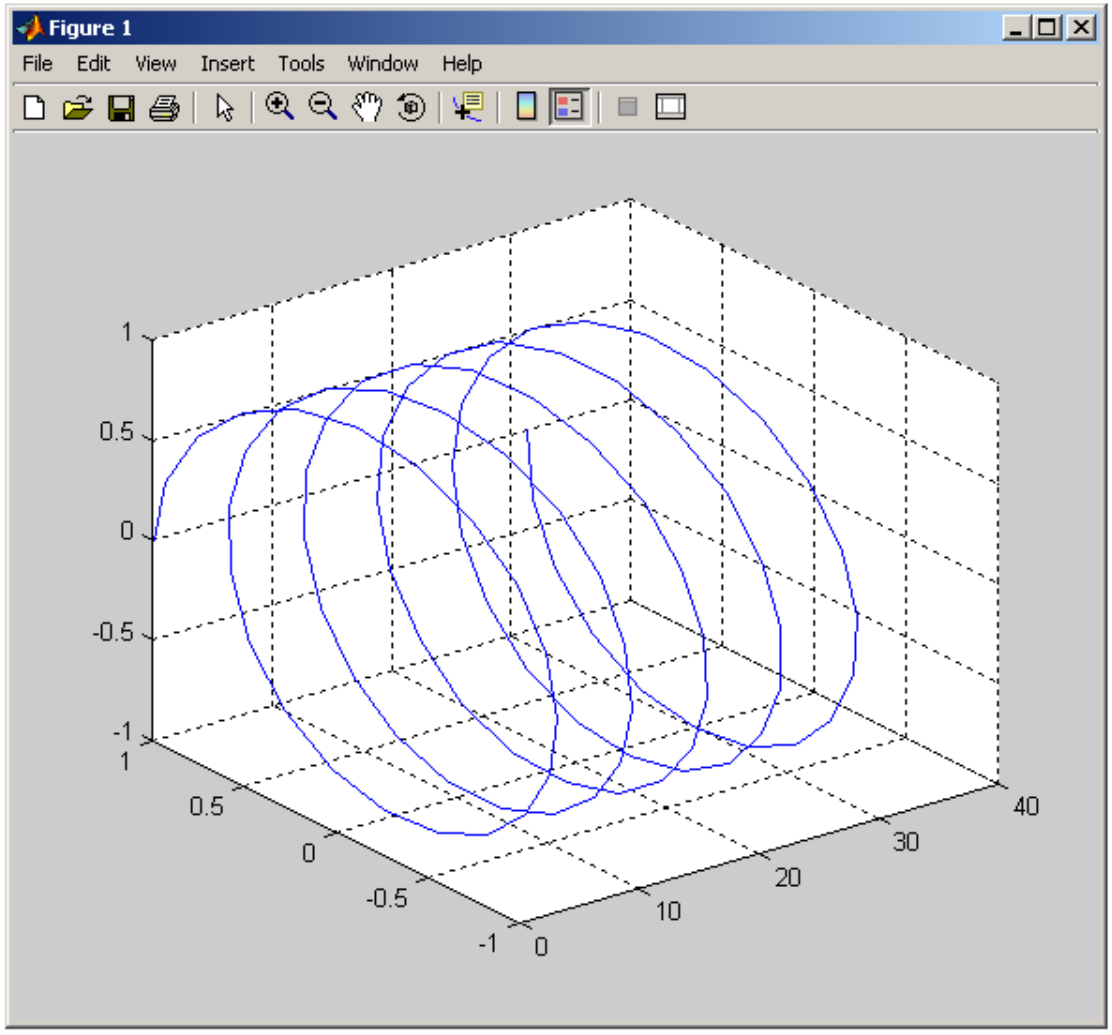
Workspace
Current Directory

Command History

```
clear, cl
x=linspac
y = cos(x
z = sin(x
clear, cl
x=linspac
y = cos(x
z = sin(x
plot3(x,y
grid
```

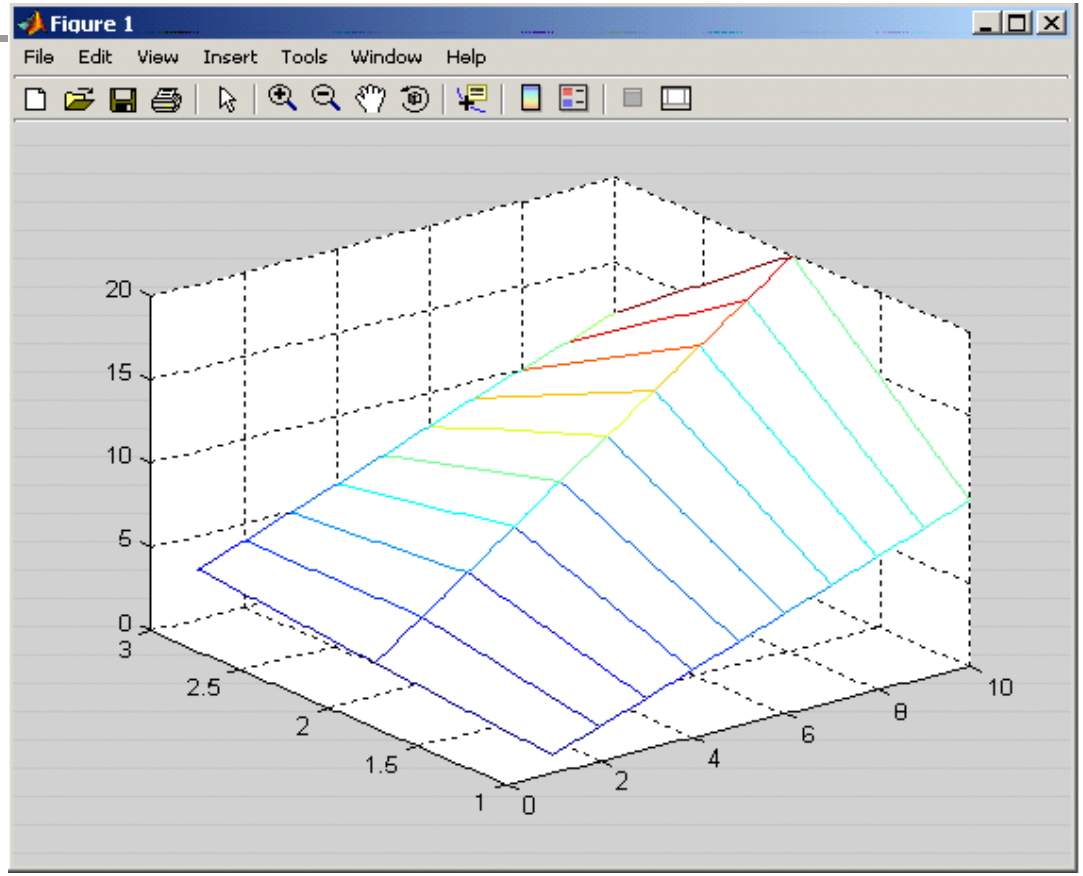
Command Window

```
>> x=linspace(0,10*pi,100);
>> y = cos(x);
>> z = sin(x);
>> plot3(x,y,z)
>> grid
>>
```



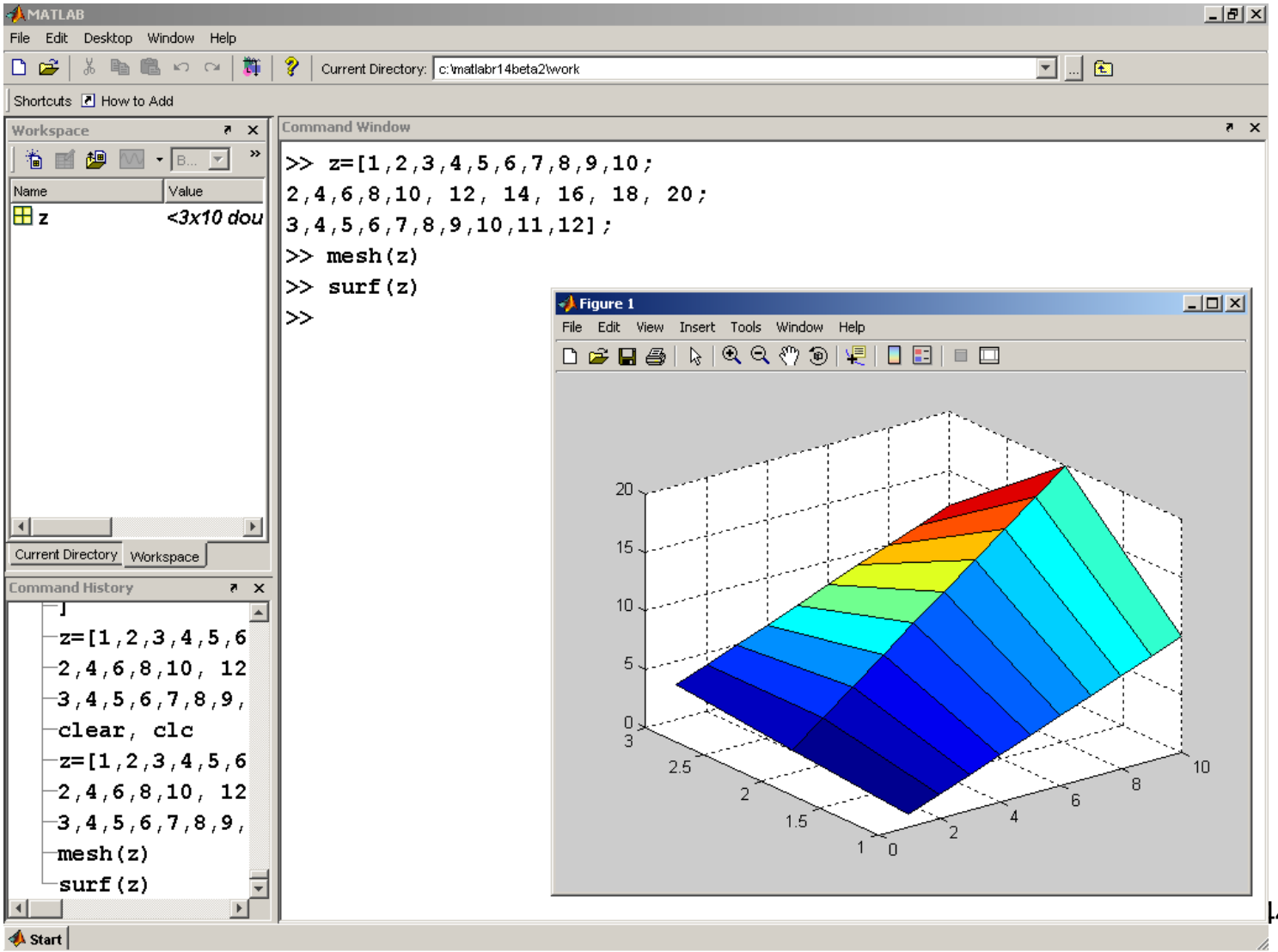
Start

Mesh




```
>> z=[1 2 3 4 5 6 7 8 9 10;
2 4 6 8 10 12 14 16 18 20;
 3 4 5 6 7 8 9 10 11 12]
z =

     1     2     3     4     5     6     7     8     9    10
     2     4     6     8    10    12    14    16    18    20
     3     4     5     6     7     8     9    10    11    12

>> mesh(z)
```

Surf


MATLAB
File Edit Desktop Window Help
Current Directory: c:\matlabr14beta2\work
Shortcuts How to Add
Workspace
Name Value
z <3x10 dou
Current Directory Workspace
Command History
z=[1,2,3,4,5,6
2,4,6,8,10, 12
3,4,5,6,7,8,9,
clear, clc
z=[1,2,3,4,5,6
2,4,6,8,10, 12
3,4,5,6,7,8,9,
mesh(z)
surf(z)
Command Window
>> z=[1,2,3,4,5,6,7,8,9,10;
2,4,6,8,10, 12, 14, 16, 18, 20;
3,4,5,6,7,8,9,10,11,12];
>> mesh(z)
>> surf(z)
>>
Figure 1
File Edit View Insert Tools Window Help
20
15
10
5
0
3
2.5
2
1.5
1
0
2
4
6
8
10
Start

Three dimensional plotting

$$f2d = e^{(5*(x2d^2+y2d^2))}$$

x2d=-1, -0.5, 0, 0.5, 1 ; y2d=-1, -0.5, 0, 0.5, 1

Three dimensional plotting

```
>> x1d=linspace(-1,1,5)

x1d =

   -1.0000   -0.5000         0    0.5000    1.0000

>> y1d=linspace(-1,1,5)

y1d =

   -1.0000   -0.5000         0    0.5000    1.0000

>> [x2d,y2d]=meshgrid(x1d,y1d);
```

Three dimensional plotting

x2d =

-1.0000	-0.5000	0	0.5000	1.0000
-1.0000	-0.5000	0	0.5000	1.0000
-1.0000	-0.5000	0	0.5000	1.0000
-1.0000	-0.5000	0	0.5000	1.0000
-1.0000	-0.5000	0	0.5000	1.0000

y2d =

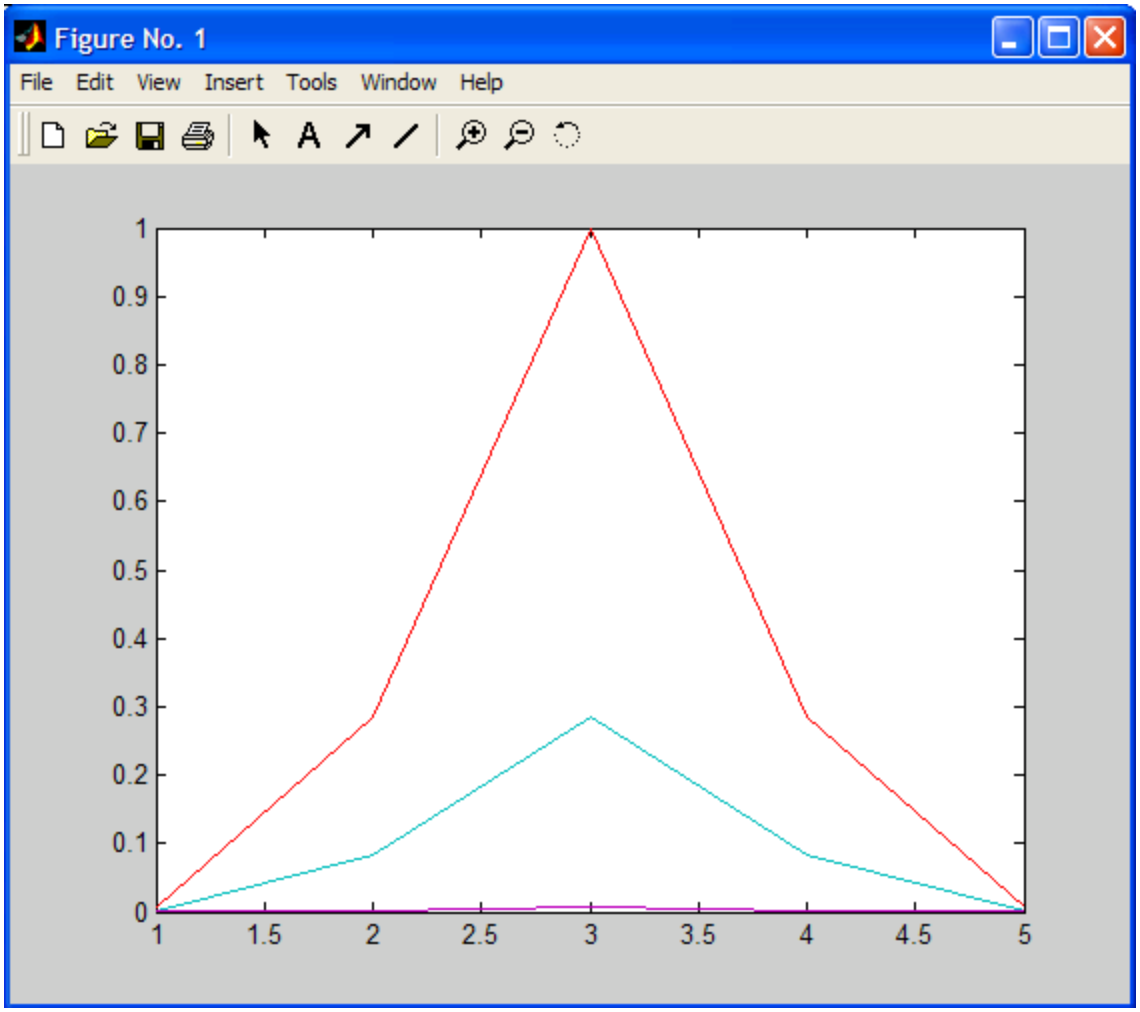
-1.0000	-1.0000	-1.0000	-1.0000	-1.0000
-0.5000	-0.5000	-0.5000	-0.5000	-0.5000
0	0	0	0	0
0.5000	0.5000	0.5000	0.5000	0.5000
1.0000	1.0000	1.0000	1.0000	1.0000

Three dimensional plotting

```
>> f2d=exp(-5*(x2d.^2+y2d.^2));

  f2d =

    0.0000    0.0019    0.0067    0.0019    0.0000
    0.0019    0.0821    0.2865    0.0821    0.0019
    0.0067    0.2865    1.0000    0.2865    0.0067
    0.0019    0.0821    0.2865    0.0821    0.0019
    0.0000    0.0019    0.0067    0.0019    0.0000
```

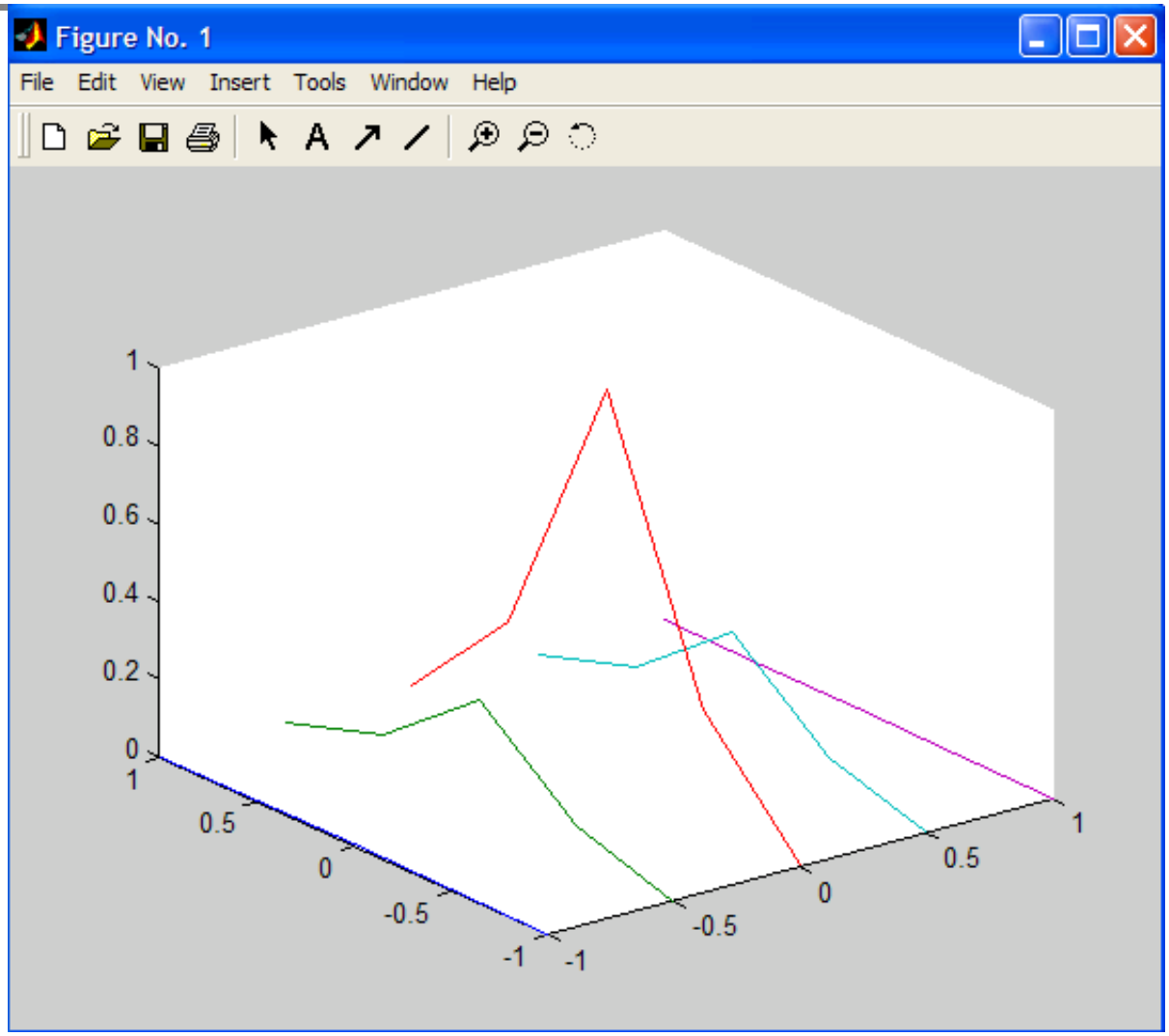
Three dimensional plotting

```
>> plot(f2d)
```



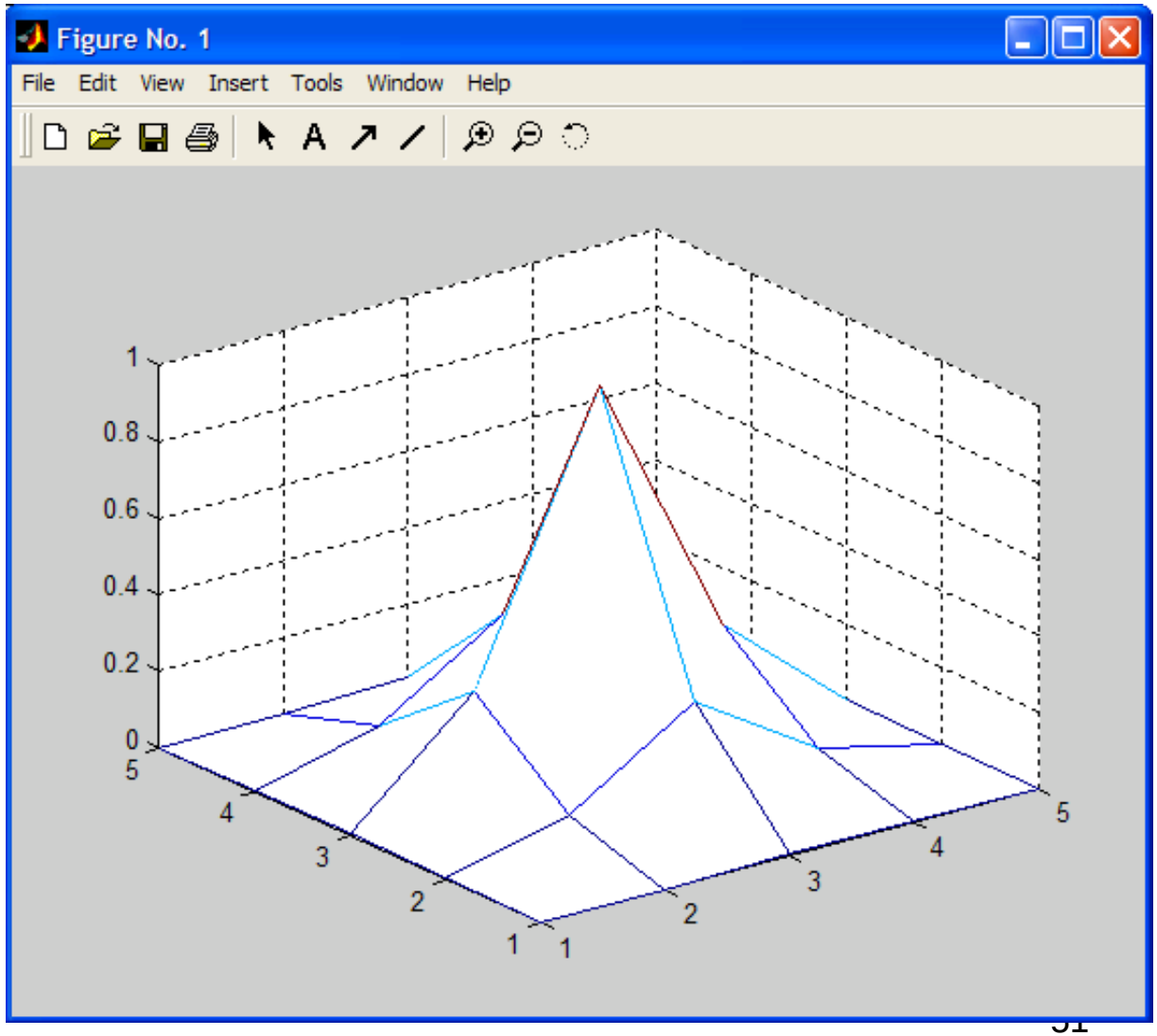
Three dimensional plotting

```
>> plot3(x2d,y2d,f2d)
```



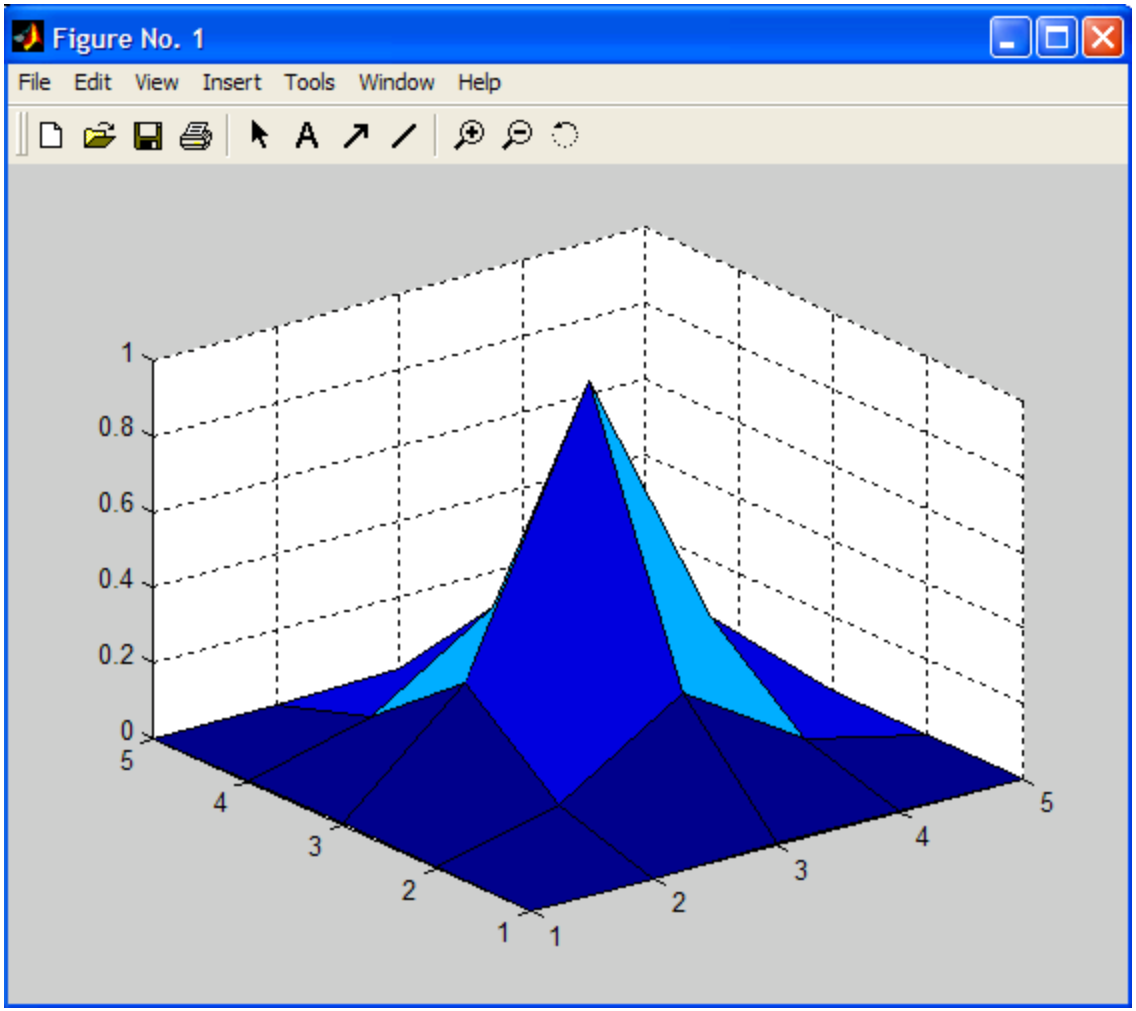
Three dimensional plotting

```
>> mesh(f2d)
```



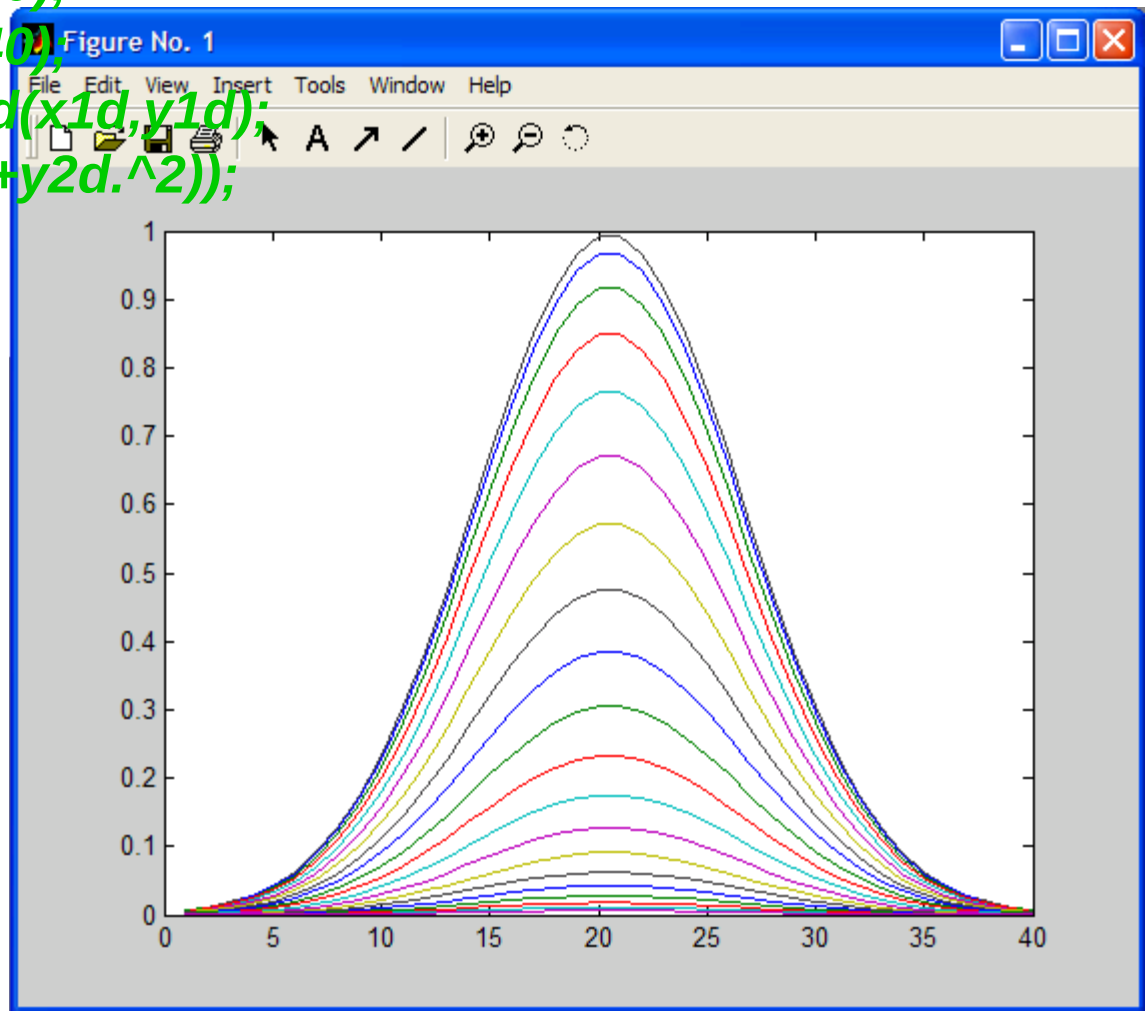
Three dimensional plotting

```
>> surf(f2d)
```



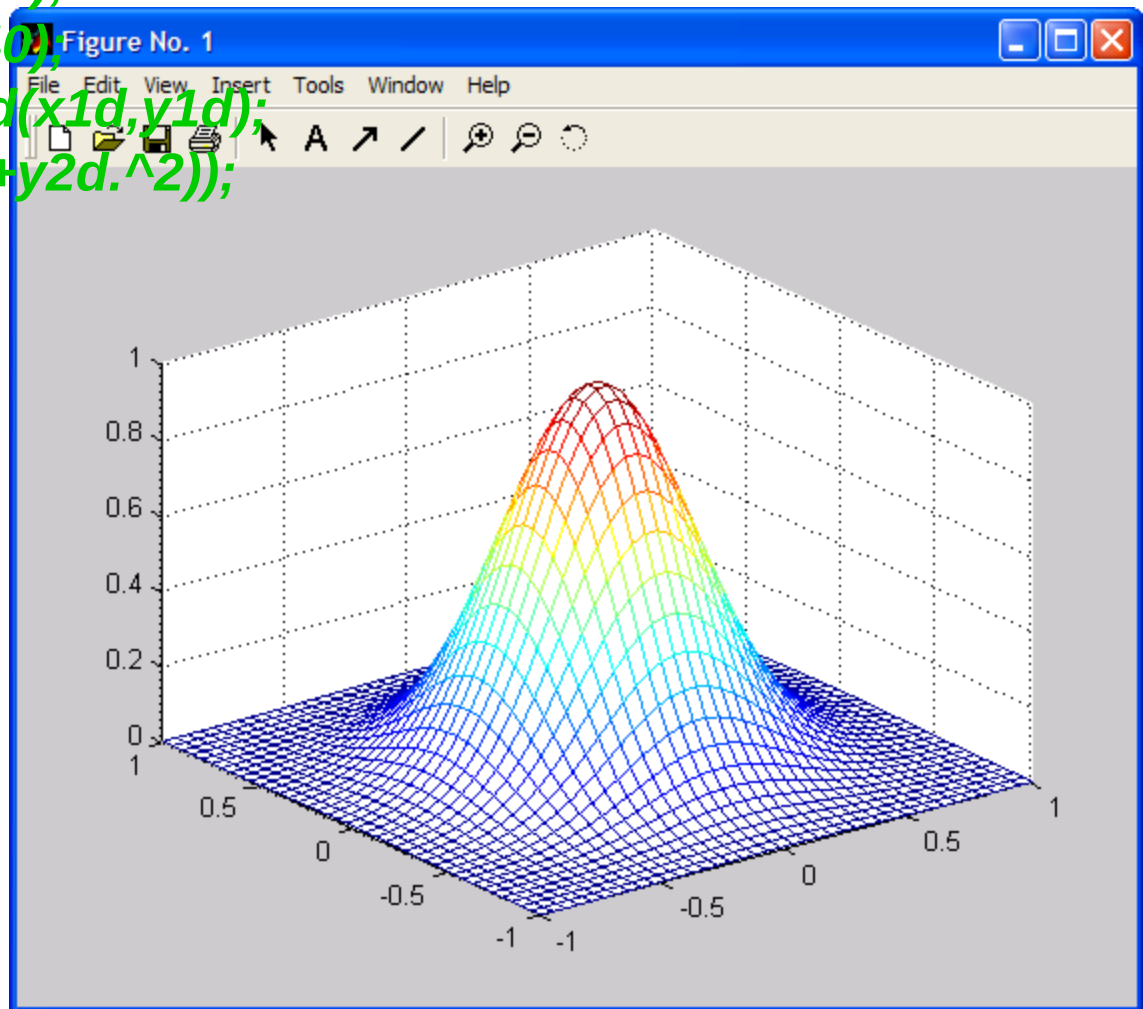
Three dimensional plotting

```
>> x1d=linspace(-1,1,40);
>> y1d=linspace(-1,1,40);
>> [x2d,y2d]=meshgrid(x1d,y1d);
>> f2d=exp(-5*(x2d.^2+y2d.^2));
>> plot(f2d)
```



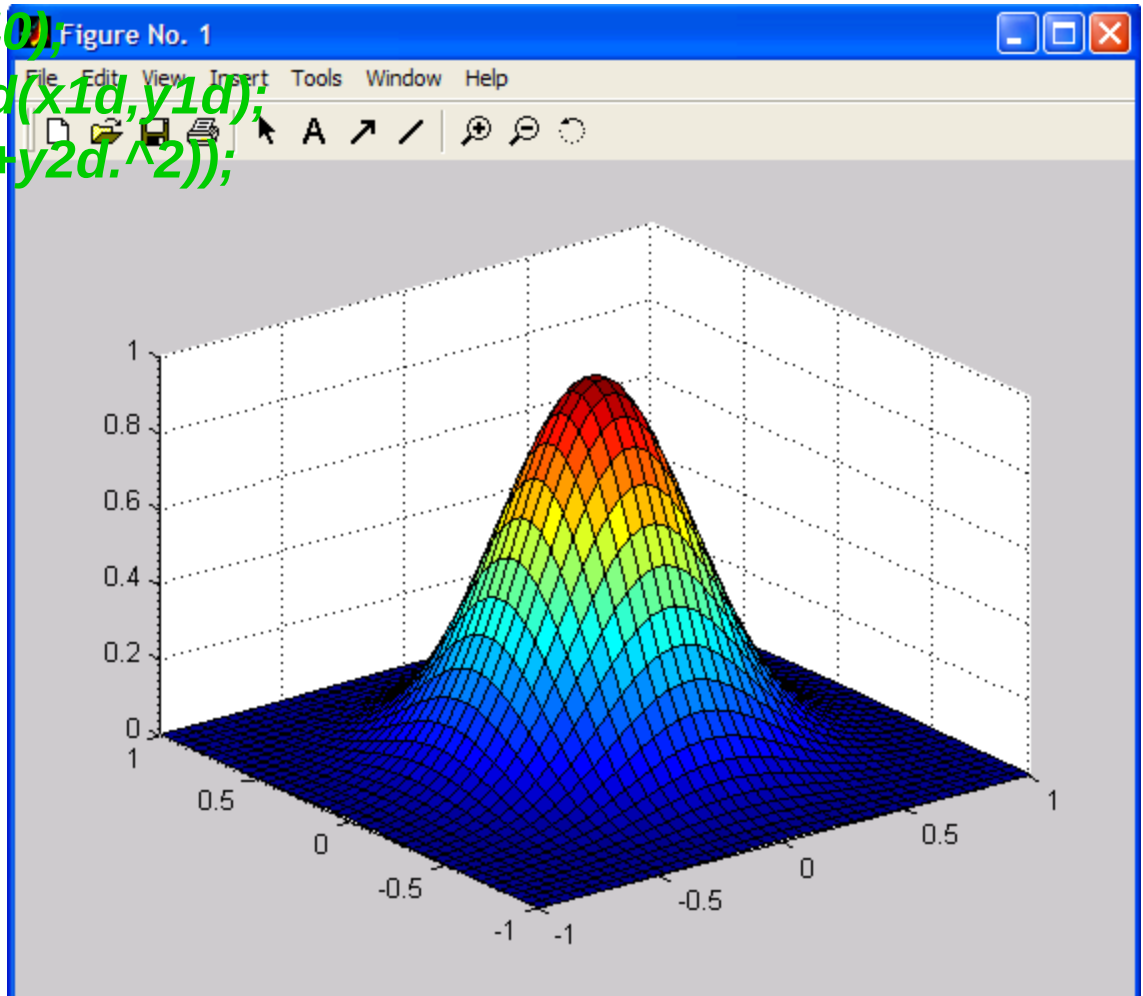
Three dimensional plotting

```
>> x1d=linspace(-1,1,40);
>> y1d=linspace(-1,1,40);
>> [x2d,y2d]=meshgrid(x1d,y1d);
>> f2d=exp(-5*(x2d.^2+y2d.^2));
>> plot(f2d)
>> mesh(x2d,y2d,f2d)
```



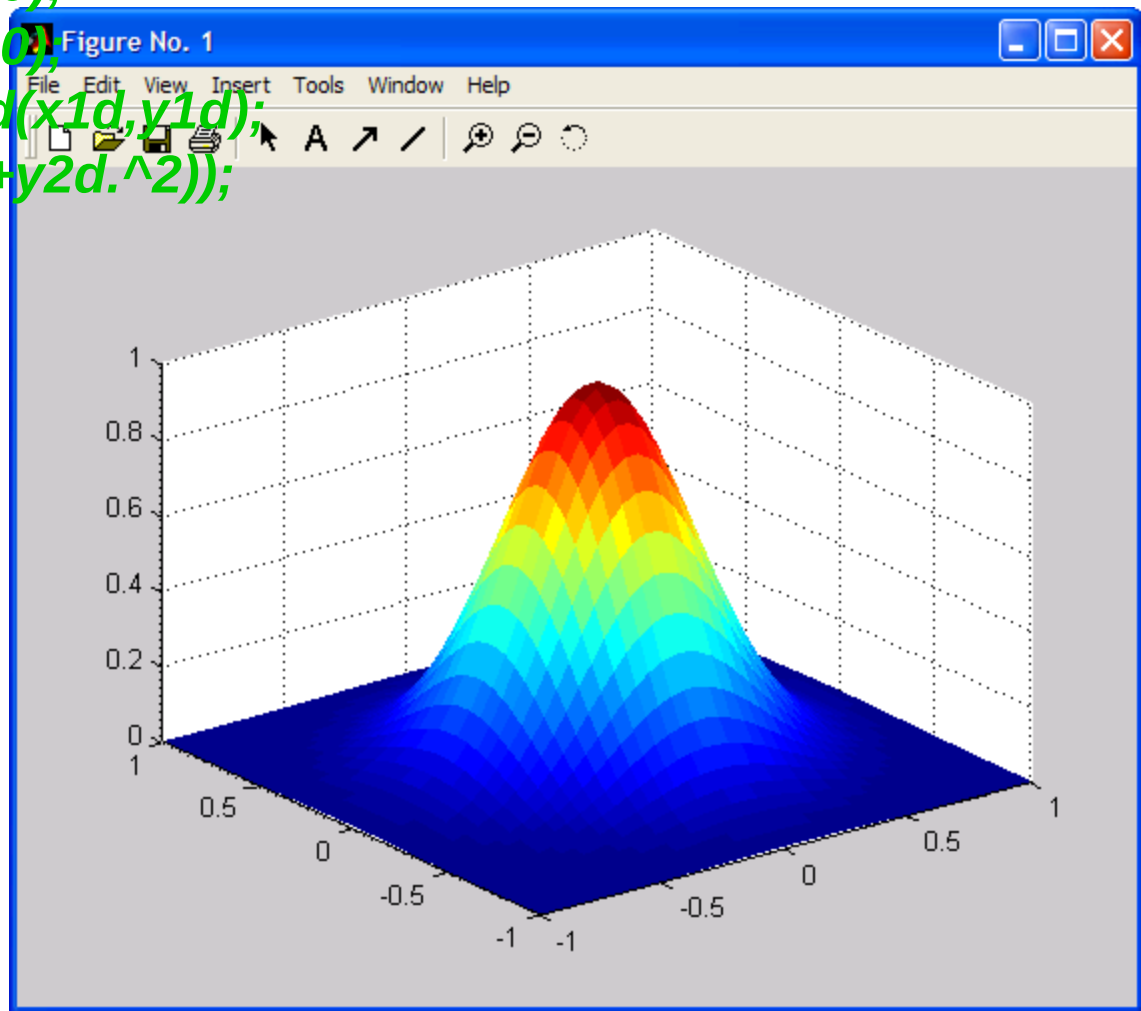
Three dimensional plotting

```
>> x1d=linspace(-1,1,40);
>> y1d=linspace(-1,1,40);
>> [x2d,y2d]=meshgrid(x1d,y1d);
>> f2d=exp(-5*(x2d.^2+y2d.^2));
>> plot(f2d)
>> mesh(x2d,y2d,f2d)
>> surf(x2d,y2d,f2d)
```



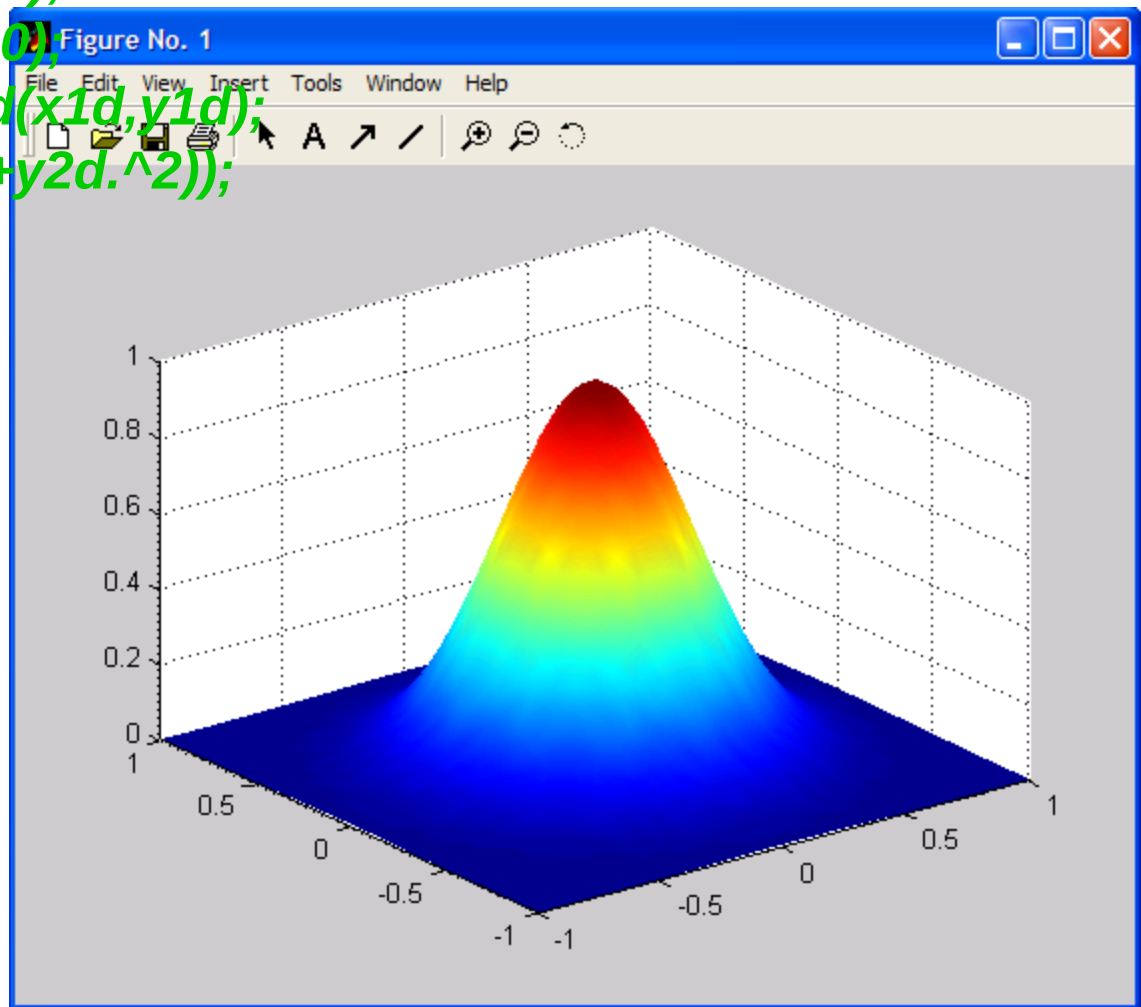
Three dimensional plotting

```
>> x1d=linspace(-1,1,40);
>> y1d=linspace(-1,1,40);
>> [x2d,y2d]=meshgrid(x1d,y1d);
>> f2d=exp(-5*(x2d.^2+y2d.^2));
>> plot(f2d)
>> mesh(x2d,y2d,f2d)
>> surf(x2d,y2d,f2d)
>> shading flat
```



Three dimensional plotting

```
>> x1d=linspace(-1,1,40);
>> y1d=linspace(-1,1,40);
>> [x2d,y2d]=meshgrid(x1d,y1d);
>> f2d=exp(-5*(x2d.^2+y2d.^2));
>> plot(f2d)
>> mesh(x2d,y2d,f2d)
>> surf(x2d,y2d,f2d)
>> shading flat
>> shading interp
```



Example-II

```
function y=function_generator(f,VM,type)

if nargin==2
   type='sin'
end

if f<1000
   t=0:0.0001:10/f;
elseif f<10000
   t=0:0.00001*1e-1:10/f;
elseif f<100000
   t=0:0.00001*1e-2:10/f;
elseif f<1000000
   t=0:0.00001*1e-3:10/f;
end
```

Example-II

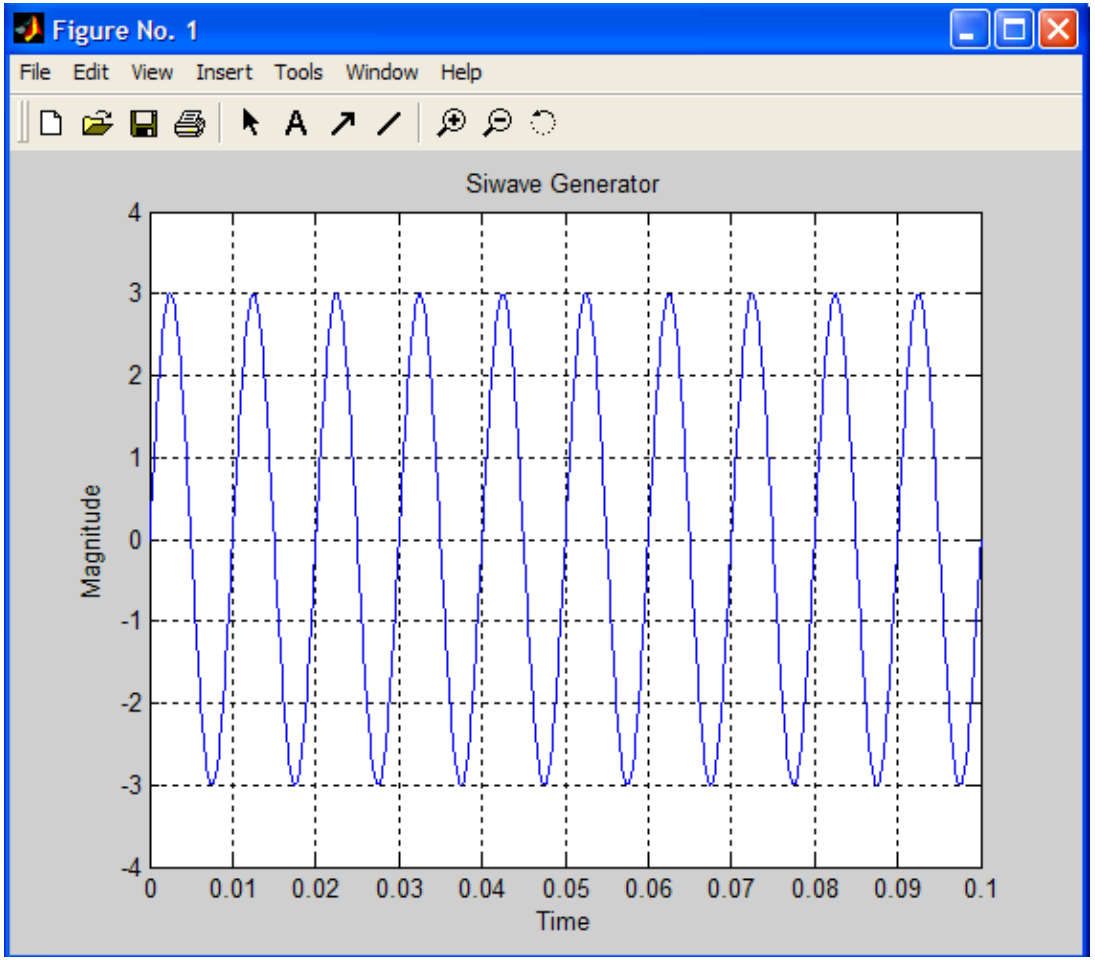
```
switch (type)
   case 'square'
      y=VM*square(2*pi*f*t);
      plot(t,y);
      axis([0 10/f -VM-1 VM+1]);
      title('Squarewave Generator');
      xlabel('Time');
      ylabel('Magnitude');
      grid

   case 'sin'
      y=VM*sin(2*pi*f*t);
      plot(t,y);
      axis([0 10/f -VM-1 VM+1]);
      title('Siwave Generator');
      xlabel('Time');
      ylabel('Magnitude');
      grid
```

Example-II

```
  case 'triangle'
      y=VM*sawtooth(2*pi*f*t,0.5);
      plot(t,y);
      axis([0 10/f -VM-1 VM+1]);
      title('Trianglewave Generator');
      xlabel('Time');
      ylabel('Magnitude');
      grid
  otherwise
      break
end
```

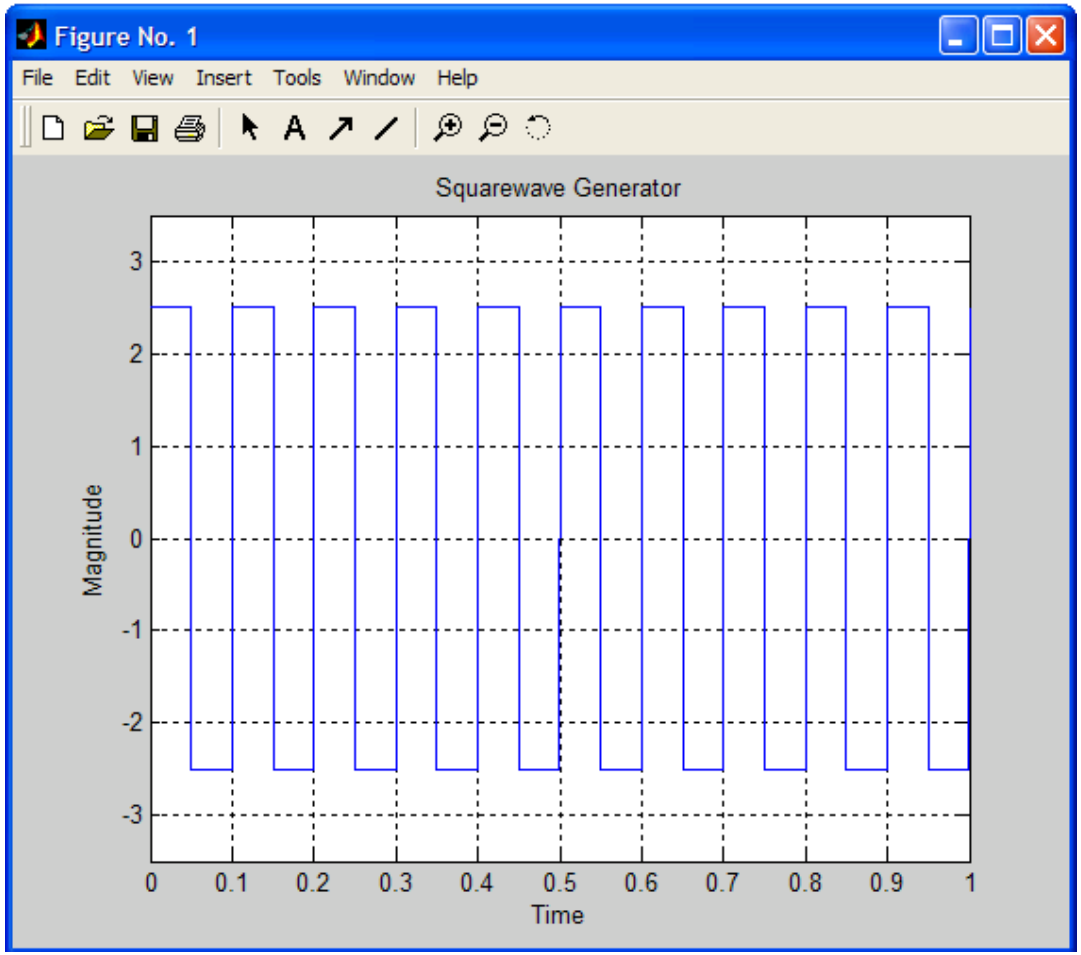
Example-II

```
>> y=function_generator(100,3,'sin');
```



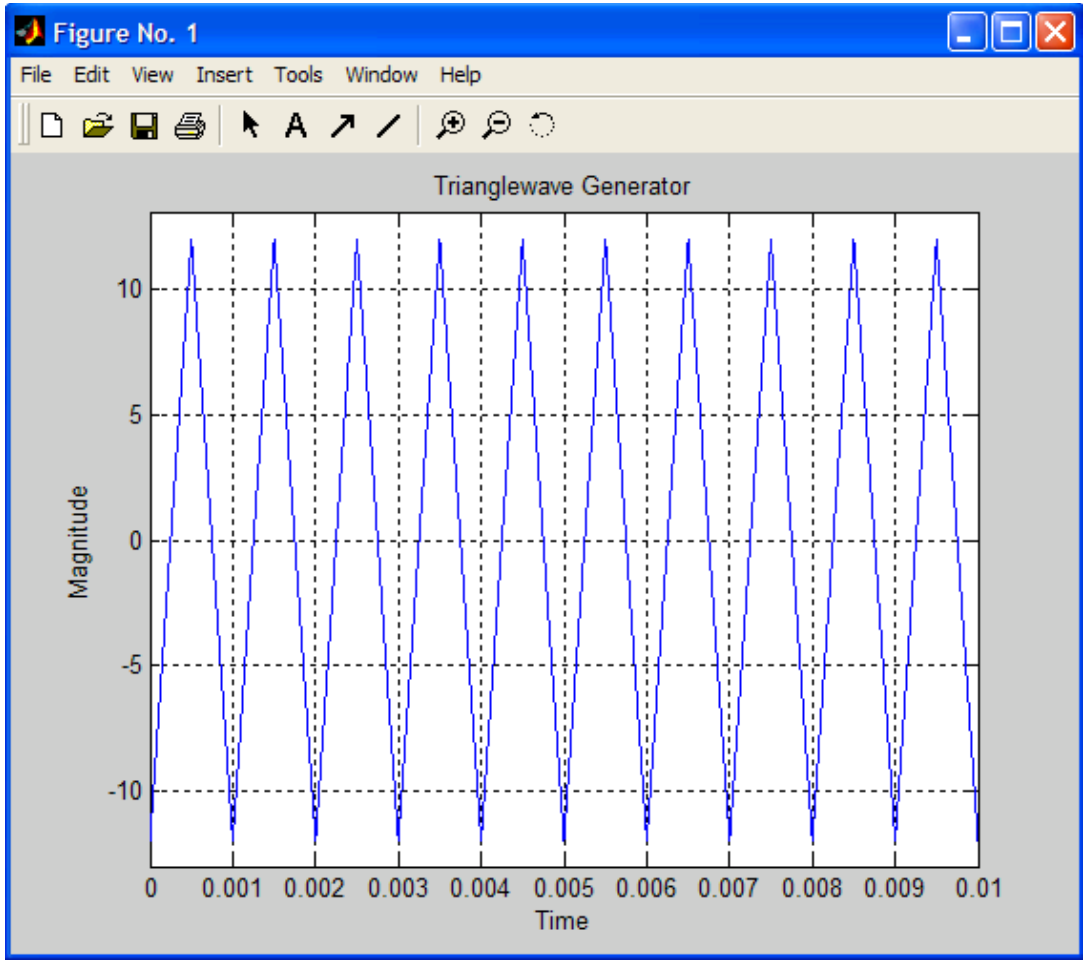
Example-II

```
>> y=function_generator(10,2.5,'square');
```



Example-II

```
>> y=function_generator(1000,12,'triangle');
```



Example-III

Use a "for" loop to compute and plot the following piecewise–defined equation over the range $x \in [-5,5]$.

$$y = \begin{cases} x^3, & x < -2 \\ 2x^2 - 16, & -2 \le x \le 3 \\ 32 - 10x, & x > 3 \end{cases}$$

Example-III

```
clear all
close all
clc
x = linspace(-5,5,100);
y = zeros(size(x));
for ii = 1:100,
  if (x(ii) < -2)
    y(ii) = x(ii)^3;
  elseif (x(ii) >= -2) & (x(ii) <= 3)
    y(ii) = 2*x(ii)^2 - 16;
  elseif (x(ii) > 3)
    y(ii) = 32 - 10*x(ii);
  end
end
plot(x,y)
xlabel('x')
ylabel('y')
```

Example-III

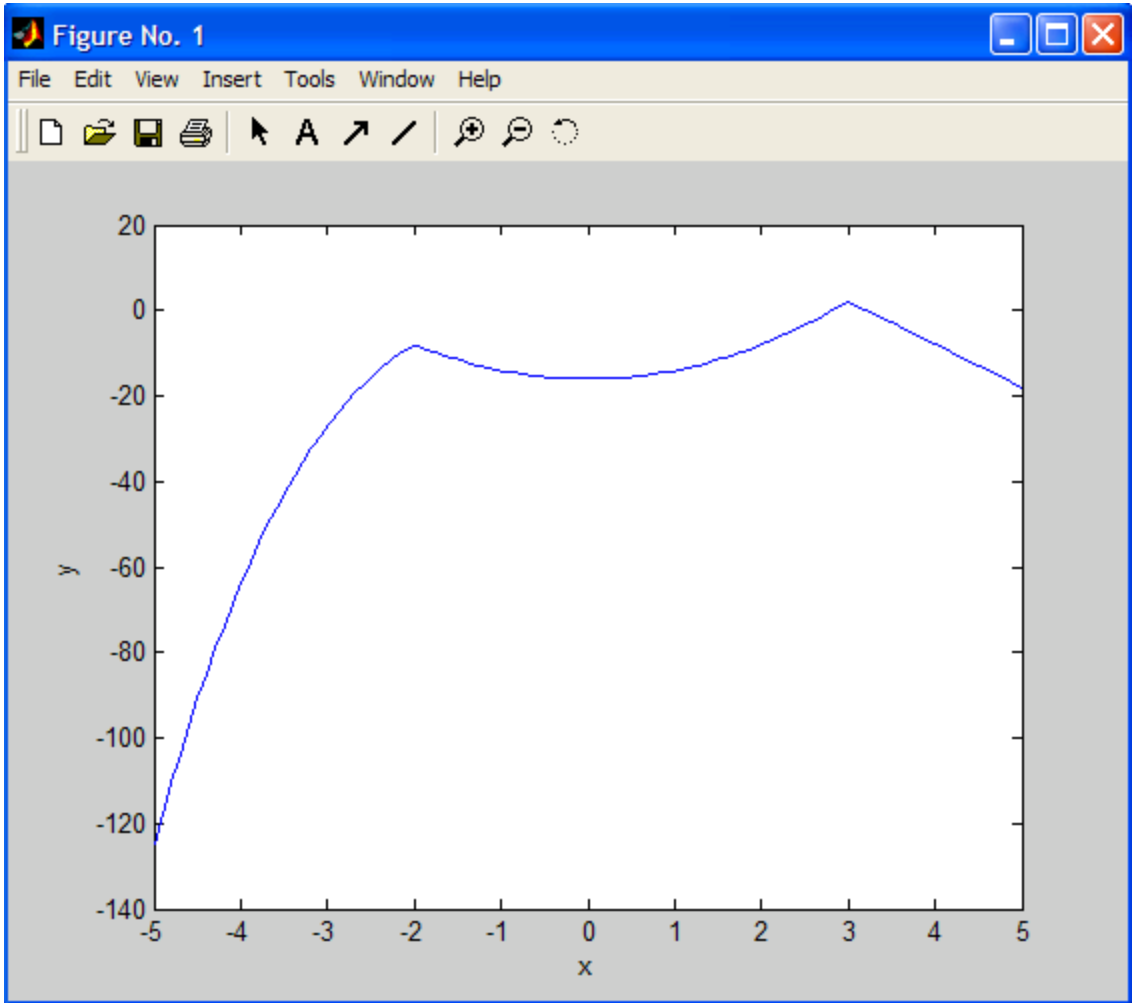


Figure No. 1
File Edit View Insert Tools Window Help
20
0
-20
-40
-60
-80
-100
-120
-140
y
-5 -4 -3 -2 -1 0 1 2 3 4 5
x

Polynomials

Polynomial are represented in MATLAB by the array of the coefficients of their term, in the order of the descending powers of the variable.

For example, the polynomial

$$x^2 + 2x + 1$$

is stored in MATLAB as the array

```
>> coeff=[1 2 1];
```

Polynomial Roots

If the polynomial is the left-hand side of the equation

$$x^2 + 2x + 1 = 0$$

The roots are obtained by

```
>> r=roots(coeff)

r=
     -1
     -1
```

This is the expected solution because the given polynomial is simply the expansion of $(x+1)^2$.

Polynomial Roots

As a more interesting example, let us solve the equation

$$x^3 - 15x = 4$$

The roots are obtained by

```
>> coeff=[1 0 -15 -4];
>> r=roots(coeff)

r=
    4.0000
   -3.7321
   -0.2679
```

Retrieving Polynomial Coefficients from Polynomial Roots

The coefficients of a polynomial can be retrieved from its roots. Thus, continuing the preceding example,

```
>> p=poly(r)

p=
    1.0000         0.0000       -15.0000       -4.000
```

Retrieving Polynomial Coefficients from Polynomial Roots

Let us find the coefficients of a polynomial that has three roots equal to one:

```
>> r=[1 1 1];
>> p=poly(r)
 p=
        1        -3         3        -1
```

These are indeed the coefficients of the expansion of

$$(x-1)^3$$

Polynomial Evaluation

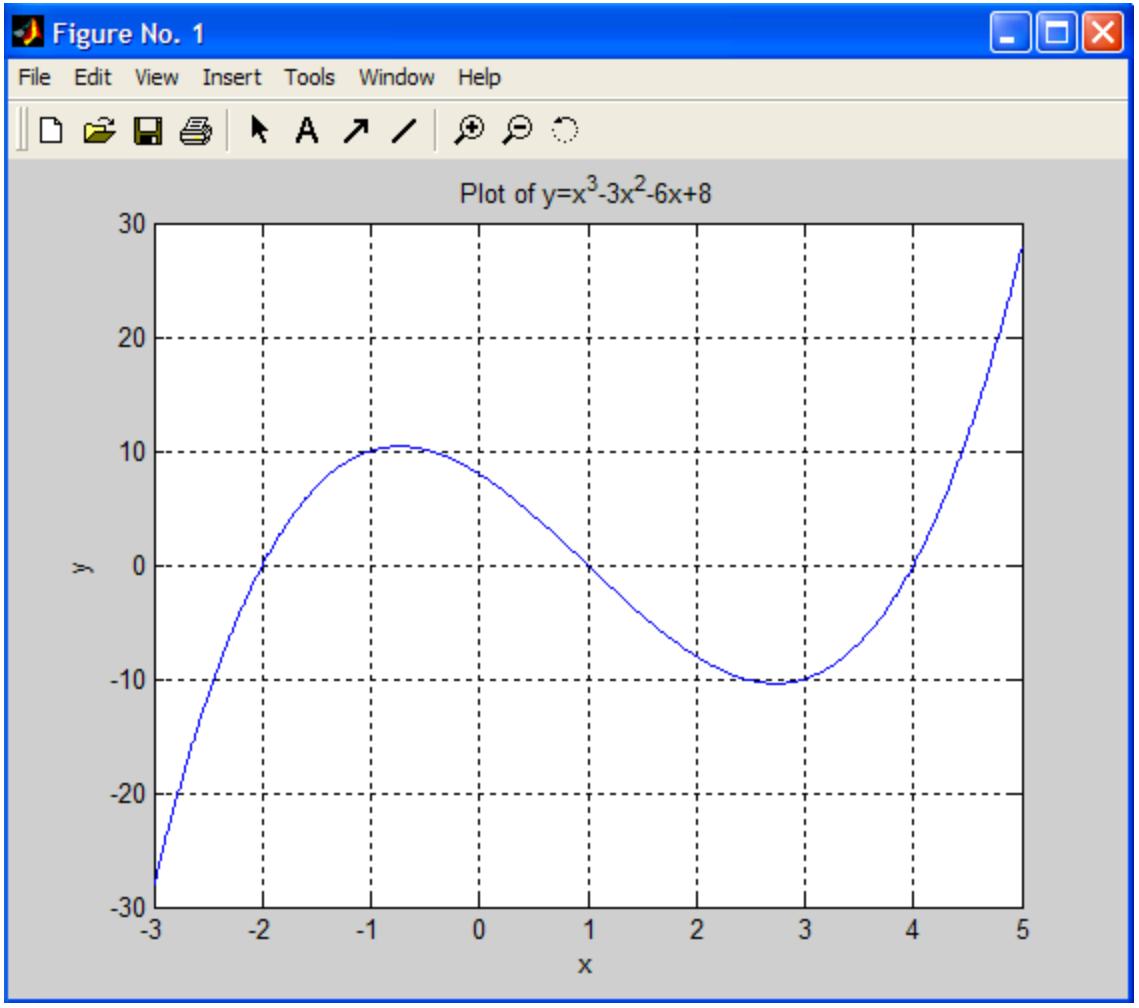
Let us suppose that we want to plot the function

$$y = x^3 - 3x^2 - 6x + 8$$

in a range that includes the zeros of y. In this case,

```
>> coeff=[1 -3 -6 8];
>> r=roots(coeff)

r=
     4.0000
    -2.0000
     1.0000

>> x=-3:0.01:5;
>> y=polyval(coeff,x);
>> plot(x,y);
>> grid
>> title('plot of y=x^3-3x^2-6x+8')
>> xlabel('x'), ylabel('y')
```

Polynomial Evaluation



Multiplication of Polynomials

The multiplication of two polynomials can easily be performed in MATLAB by operating on the arrays of their coefficients.

$$\left(2x^2+3x+1\right)\cdot\left(5x\ -2\right)=10x^3+11x^2-x-2$$

In MATLAB, we obtain the same result by convolution :

```
>> p1=[2 3 1];
>> p2=[5 -2];
>> p3=conv(p1,p2)
p3
      10   11   -1   -2
```

Division of Polynomials

The inverse operation, polynomials division, is performed in MATLAB by deconvolution :

```
>> [Q,R]=deconv(p3,p1)

Q
        5 -2
R
        0 0 0 0
```

Division of Polynomials

Let us substitude p3 to p4 defined as

$$10x^3 + 11x^2 + 2x$$

Dividing by p1 is calculated by

```
>> p4=[10 11 2 0];

>> [Q,R]=deconv(p4,p1)
Q
        5 -2
R
        0 0 3 2
```

The remainder of the division of p4 by p1 is indeed 3x+2.

Polynomial fit and interpolation

Data is often given for discrete values. In this case, you may require estimates at points between the discrete values.

One way to do this to compute value of the function at a number of discrete values along the range of interest. Then a simpler function may be derived to fit these values. Both of these applications are known as CURVE FITTING.

There are two general approaches for curve fitting:

- Least-squares regression
- Interpolation

Polynomial fit and interpolation

A second order-degree or a higher order-degree curve can be fitted to the given data by the least-square method.

Assuming that an experiment yielded m pairs of data, (x_i, y_i), and that a second order-degree curve seems to fit those data, we must seek the three coefficients c_1, c_2, c_3 of the equation:

$$y = c_1 x^2 + c_2 x + c_3$$

Polynomial fit and interpolation

That minimized the sum of squared errors

$$\varepsilon = \sum_{i=1}^{m} \left(y_i - c_1 x_i^2 - c_2 x_i - c_3 \right)^2$$

The common dimension of the arrays *x* and *y* is *m*.

Differentiating ε with respect to c_1, c_2, and c_3 and equating the derivativesa to zero

Polynomial fit and interpolation

$$\frac{\partial \varepsilon}{\partial c_1} = 2\left[\sum_{i=1}^{m}\left(yi - c_1 x_i^2 - c_2 x_i - c_3\right)\right]\cdot\left[\sum_{i=1}^{m} - x_i^2\right] = 0$$

$$\frac{\partial \varepsilon}{\partial c_2} = 2\left[\sum_{i=1}^{m}\left(yi - c_1 x_i^2 - c_2 x_i - c_3\right)\right]\cdot\left[\sum_{i=1}^{m} - x_i\right] = 0$$

$$\frac{\partial \varepsilon}{\partial c_3} = 2\left[\sum_{i=1}^{m}\left(yi - c_1 x_i^2 - c_2 x_i - c_3\right)\right]\cdot\left[\sum_{i=1}^{m} -1\right] = 0$$

Polynomial fit and interpolation

$$\frac{\partial \varepsilon}{\partial c_1} = 2\left[\sum_{i=1}^{m} - x_i^2 yi + c_1 x_i^4 + c_2 x_i^3 + c_3 x_i^2\right] = 0$$

$$\frac{\partial \varepsilon}{\partial c_2} = 2\left[\sum_{i=1}^{m} - x_i yi + c_1 x_i^3 + c_2 x_i^2 + c_3 x_i\right] = 0$$

$$\frac{\partial \varepsilon}{\partial c_3} = 2\left[\sum_{i=1}^{m} \left(- yi + c_1 x_i^2 + c_2 x_i + c_3\right)\right] = 0$$

Polynomial fit and interpolation

$$\sum_{i=1}^{m}\left(c_1 x_i^4 + c_2 x_i^3 + c_3 x_i^2\right) = \sum_{i=1}^{m} x_i^2 yi$$

$$\sum_{i=1}^{m}\left(c_1 x_i^3 + c_2 x_i^2 + c_3 x_i\right) = \sum_{i=1}^{m} x_i yi$$

$$\sum_{i=1}^{m}\left(c_1 x_i^2 + c_2 x_i + c_3\right) = \sum_{i=1}^{m} yi$$

Polynomial fit and interpolation

We obtain the system

$$\begin{bmatrix} \sum_{i=1}^{m} x_i^4 & \sum_{i=1}^{m} x_i^3 & \sum_{i=1}^{m} x_i^2 \\ \sum_{i=1}^{m} x_i^3 & \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i & m \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i x_i^2 \\ \sum_{i=1}^{m} y_i x_i \\ \sum_{i=1}^{m} y_i \end{bmatrix}$$

Polynomial fit and interpolation

$$\begin{bmatrix} \sum_{i=1}^{m} x_i^4 & \sum_{i=1}^{m} x_i^3 & \sum_{i=1}^{m} x_i^2 \\ \sum_{i=1}^{m} x_i^3 & \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i & m \end{bmatrix} = \begin{bmatrix} x_1^2 & x_2^2 & \cdots & x_m^2 \\ x_1 & x_2 & \cdots & x_m \\ 1 & 1 & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \cdots & \cdots & \cdots \\ x_m^2 & x_m & 1 \end{bmatrix}$$

$$= A'A \qquad \Rightarrow A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \cdots & \cdots & \cdots \\ x_m^2 & x_m & 1 \end{bmatrix}$$

Polynomial fit and interpolation

$$\begin{bmatrix} \sum_{i=1}^{m} y_i x_i^2 \\ \sum_{i=1}^{m} y_i x_i \\ \sum_{i=1}^{m} y_i \end{bmatrix} = \begin{bmatrix} x_1^2 & x_2^2 & \cdots & x_m^2 \\ x_1 & x_2 & \cdots & x_m \\ 1 & 1 & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$= A'Y \qquad \Rightarrow Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Polynomial fit and interpolation

$$(A'A)\cdot C = A'Y$$

$$C = (A'A)^{-1}A'Y$$

where

$$A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \cdots & \cdots & \cdots \\ x_m^2 & x_m & 1 \end{bmatrix} \qquad C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Polynomial fit and interpolation

MATLAB provides a shortcut for performing all these operation. The polyfit function builds the matrix A and solves for C and fits a least-squares nth-order polynomial to given data.

The polyfit is used with these arguments:

- The independent variable. (x)
- The dependent variable. (y)
- The degree of the curve to be find.

MATLAB Code:

```
polyfit(x,y,n)
```

Polynomial fit and interpolation

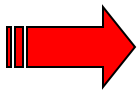
>> c=polyfit (x,y,n)

where x and y are the vectors of the independent and the dependent variables, respectively. n designates the order of the polynomial.

The function returns a vector p containinig the polynomial coefficients :

$$y = f(x) = c_1 x^{n-1} + c_2 x^{n-2} + \ldots + c_{n-1} x + c_n$$

Another function, polyval, can then be used to compute a value using the coefficients.

>> y1=polyval (c,x)  y1 is the best value at x.

Polynomial fit and interpolation

Example-1:

Fit a straight line (or a first-order polynomial) to the following data using polyfit function.

x	y
10	25
20	70
30	380
40	550
50	610
60	1220
70	830
80	1450

Polynomial fit and interpolation

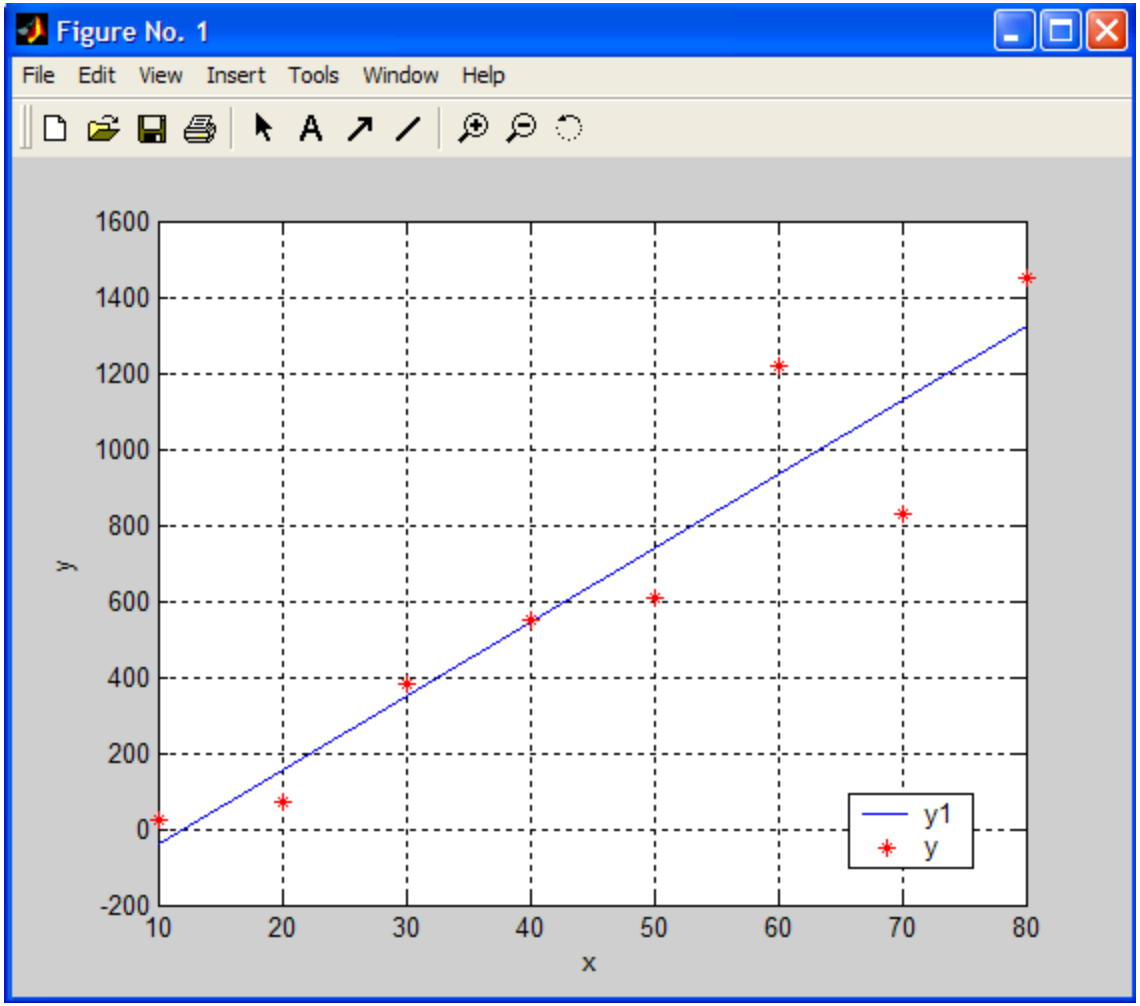
```
x=10:10:80;
y=[25 70 380 550 610 1220 830 1450];
c=polyfit(x,y,1)
y1=polyval(c,x)
plot(x,y1,x,y,'r*')
grid
xlabel('x')
ylabel('y')
legend('y1','y')
```

Polynomial fit and interpolation

```
c =

   19.4702 -234.2857

y1 =

   1.0e+003 *

   Columns 1 through 7

   -0.0396    0.1551    0.3498    0.5445    0.7392    0.9339    1.1286

   Column 8

    1.3233
```

$$y = 19.4702x - 234.2875$$

Polynomial fit and interpolation



Polynomial fit and interpolation

Example-2:

Fit a second-order polynomial to the following data using polyfit function.

x	y
0	2.1
1	7.7
2	13.6
3	27.2
4	40.9
5	61.1

Polynomial fit and interpolation

```
x=0:1:5;
y=[2.1 7.7 13.6 27.2 40.9 61.1];
c=polyfit(x,y,2)
x1=0:0.1:5;
y1=polyval(c,x1)
plot(x1,y1,x,y,'r*')
grid
xlabel('x')
ylabel('y')
legend('y1','y')
```

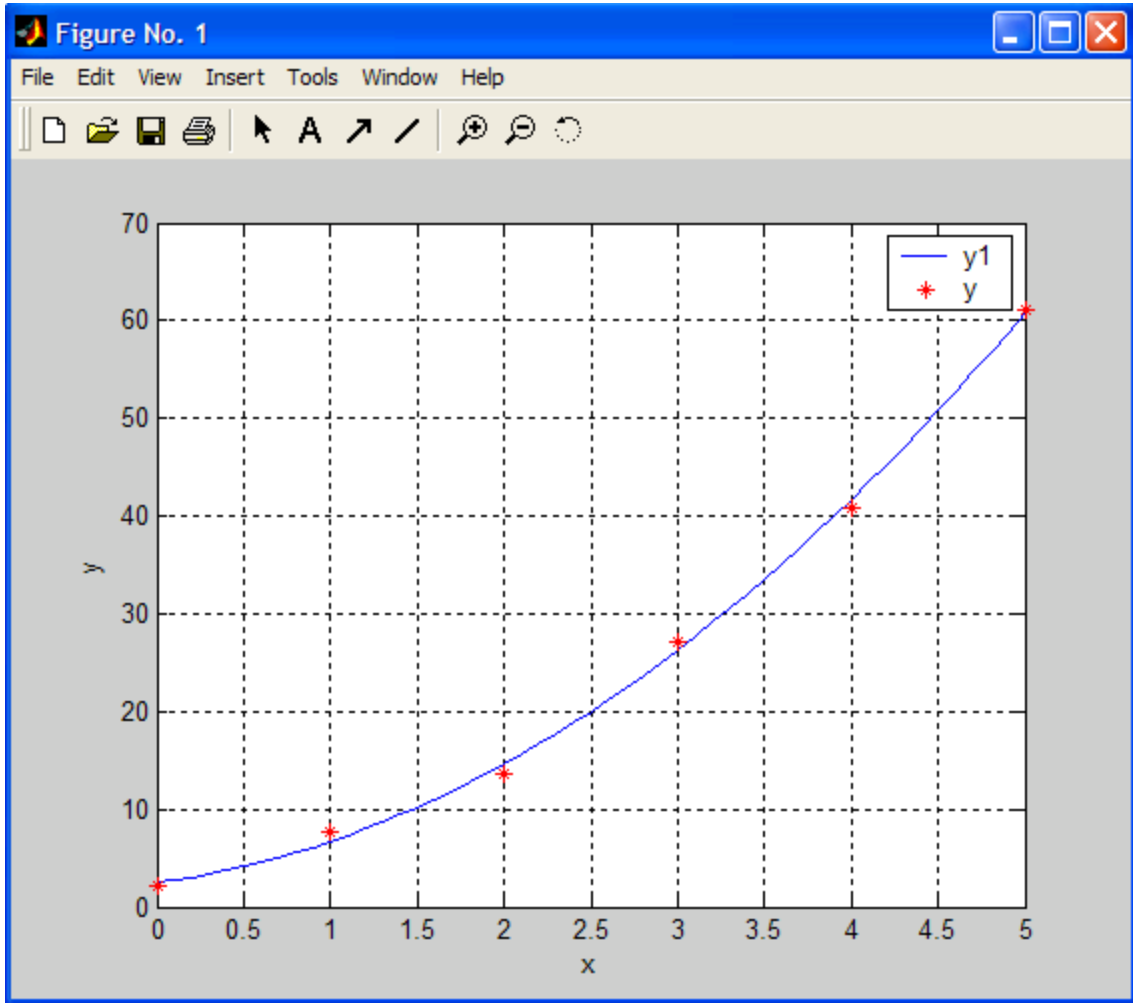
Polynomial fit and interpolation

```
c =

    1.8607    2.3593    2.4786
```

$$y = 1.8607\,x^2 + 2.3593\,x + 2.4786$$

```
y1 =

  Columns 1 through 7

    2.4786    2.7331    3.0249    3.3538    3.7200    4.1234    4.5640

  ..............

  Columns 50 through 51

   58.7148   60.7929
```

Polynomial fit and interpolation



Polynomial fit and interpolation

- In these applications, the number of data points is greater than the number of coefficients being estimated.
- Consequently, the least-square fit line does not necessarily pass through any of the points, but rather follows the general trend of the data.
- For the case where the number of data points equals the number of coefficients, polyfit performs interpolation. That is, it returns the coefficients of the polynomial that pass directly through the data points.
- Because, for n data points, there is one and only one polynomial of order (n-1) that passes through all the points.

Polynomial fit and interpolation

Example-3:

Determine the coefficients of the second-order polynomial that passes through the following data using polyfit function.

x	y
0.0000	0.0000
0.6283	0.5878
1.2566	0.9511
1.8850	0.8432
2.5133	0.3459
3.1416	0.0122

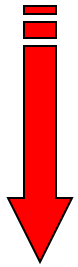
Polynomial fit and interpolation

```
x=[0 0.6283 1.2566 1.8850 2.5133 3.1416];
y=[0 0.5878 0.9511 0.8432 0.3459 0.0122];
c=polyfit(x,y,5)
x1=0:0.1:4;
y1=polyval(c,x1);
plot(x1,y1,x,y,'r*')
grid
xlabel('x')
ylabel('y')
legend('y1','y')
```

Polynomial fit and interpolation

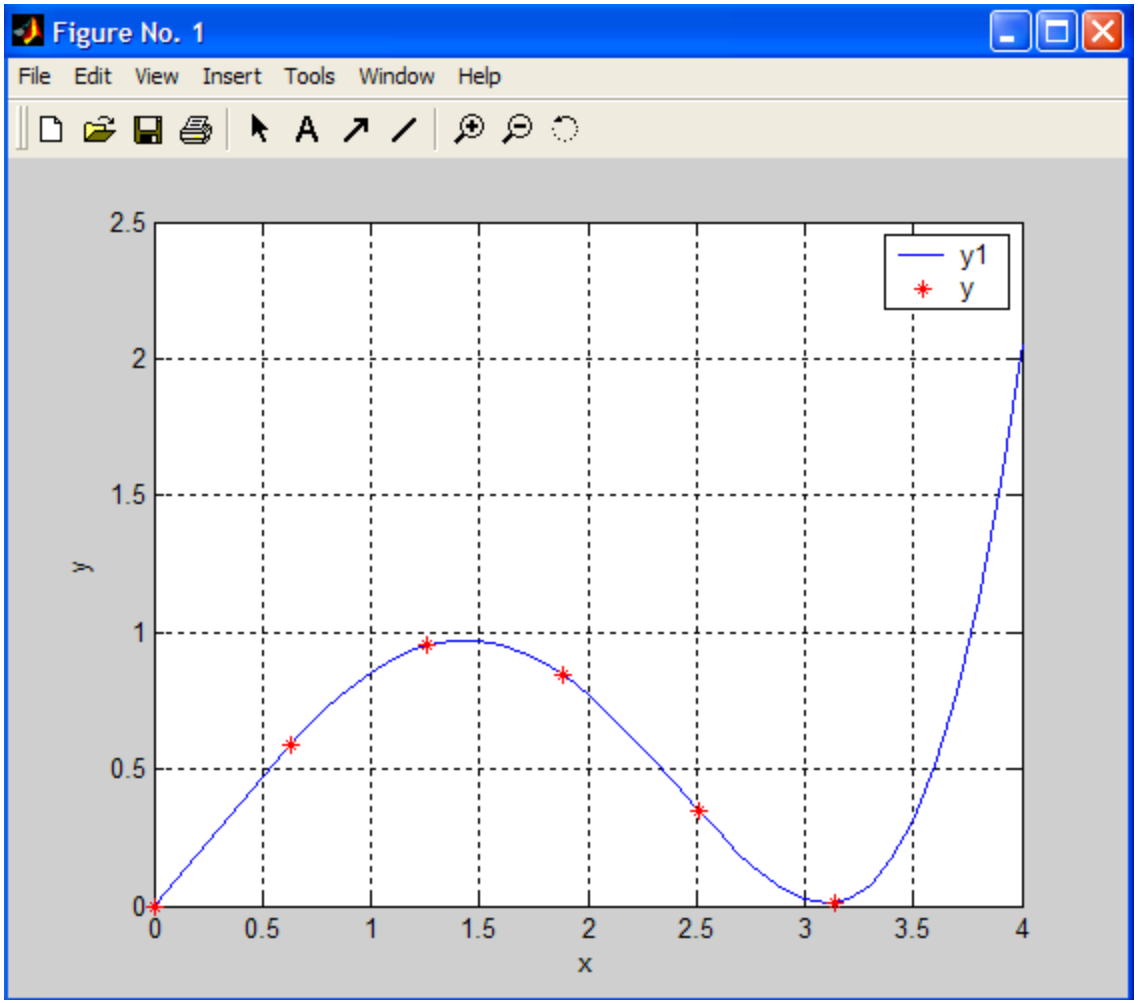
c =

0.0122 0.0114 -0.3288 0.2586 0.8981 0.0000



$$y = 0.0122x^5 + 0.0114x^4 - 0.3288x^3 + 0.2586x^2 + 0.8981x$$

Polynomial fit and interpolation

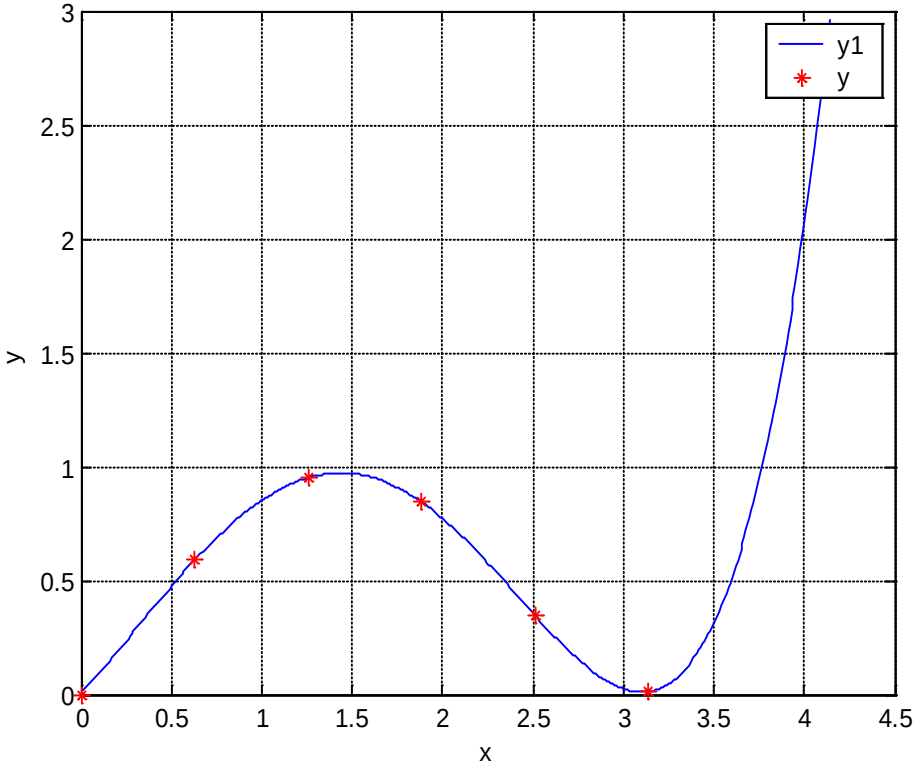


Polynomial fit and interpolation

```
function yint=polinter(x,y,n)
if nargin==2
        n=length(x);
end
c=polyfit(x,y,n)
x1=min(x):0.01:max(x);
y1=polyval(c,x1);
plot(x1,y1,x,y,'r*')
grid
xlabel('x')
ylabel('y')
legend('y1','y')
```

Polynomial fit and interpolation

```
>>x=[0 0.6283 1.2566 1.8850 2.5133 3.1416];
>>y=[0 0.5878 0.9511 0.8432 0.3459 0.0122];
>>polinter(x,y,5)
```



Lagrange Interpolating Polynomial

Linear lagrange interpolating polynomial:

$$f(x) = L_1 f(x_1) + L_2 f(x_2)$$

where

$$L_1 = \frac{x - x_2}{x_1 - x_2} \qquad L_2 = \frac{x - x_1}{x_2 - x_1}$$

$$f(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$

Lagrange Interpolating Polynomial

Second-order lagrange interpolating polynomial:

$$f(x) = L_1 f(x_1) + L_2 f(x_2) + L_3 f(x_3)$$

where

$$L_1 = \frac{(x - x_2)\cdot(x - x_3)}{(x_1 - x_2)\cdot(x_1 - x_3)} \qquad L_2 = \frac{(x - x_1)\cdot(x - x_3)}{(x_2 - x_1)\cdot(x_2 - x_3)}$$

$$L_3 = \frac{(x - x_1)\cdot(x - x_2)}{(x_3 - x_1)\cdot(x_3 - x_2)}$$

Lagrange Interpolating Polynomial

Higher-order lagrange interpolating polynomial:

$$f_{n-1}(x) = \sum_{i=1}^{n} L_i(x) f(x_i)$$

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

where n designates the number of data points.

Write a function to implement lagrange interpolation.

Lagrange Interpolating Polynomial

```
function yint=lagrange(x,y)
n=length(x);
xx=min(x):1:max(x);
for k=1:length(xx)
   s=0;
   for i=1:n
      product=y(i);
      for j=1:n
         if i~=j
            product=product*(xx(k)-x(j))/(x(i)-x(j));
         end
      end
      s=s+product;
   end
   yy(k)=s;
end
```

Lagrange Interpolating Polynomial

```
yint=yy;
plot(xx,yy,x,y,'r*')
grid
xlabel('x')
ylabel('y')
legend('lagrange
```