King Fahd University of Petroleum & Minerals
College of Engineering Sciences
Mechanical Engineering Department

ME 413: System Dynamics & Control

Name: _____

ID #: _____

Section #: _____

Date: _____

Instructor _____

MATLAB TUTORIAL

1. MATLAB ENVIRONMENT

MATLAB (an abbreviation for **MAT**rix **LAB**oratory) is a matrix-based system for mathematical and engineering calculations.

MATLAB is interactive software that is used in various areas of engineering and scientific applications. It is not a computer language in the normal sense but it does most of the work of a computer language. The power of MATLAB is represented by the length and simplicity of the code. For instance, one page of MATLAB code may be equivalent to many pages of other computer language source codes. Numerical calculations in MATLAB use collections of written subroutine libraries such as LINPACK and EISPACK. MATLAB provides Graphical User Interface (GUI) as well as three dimensional graphical animations.

The default view of MATLAB desktop is shown in Figure 1 while the basic Matlab Environment is summarized in Table 1.

On the other hand, Figure 2 shows a figure window while Figure 3 shows the MATLAB editor window.

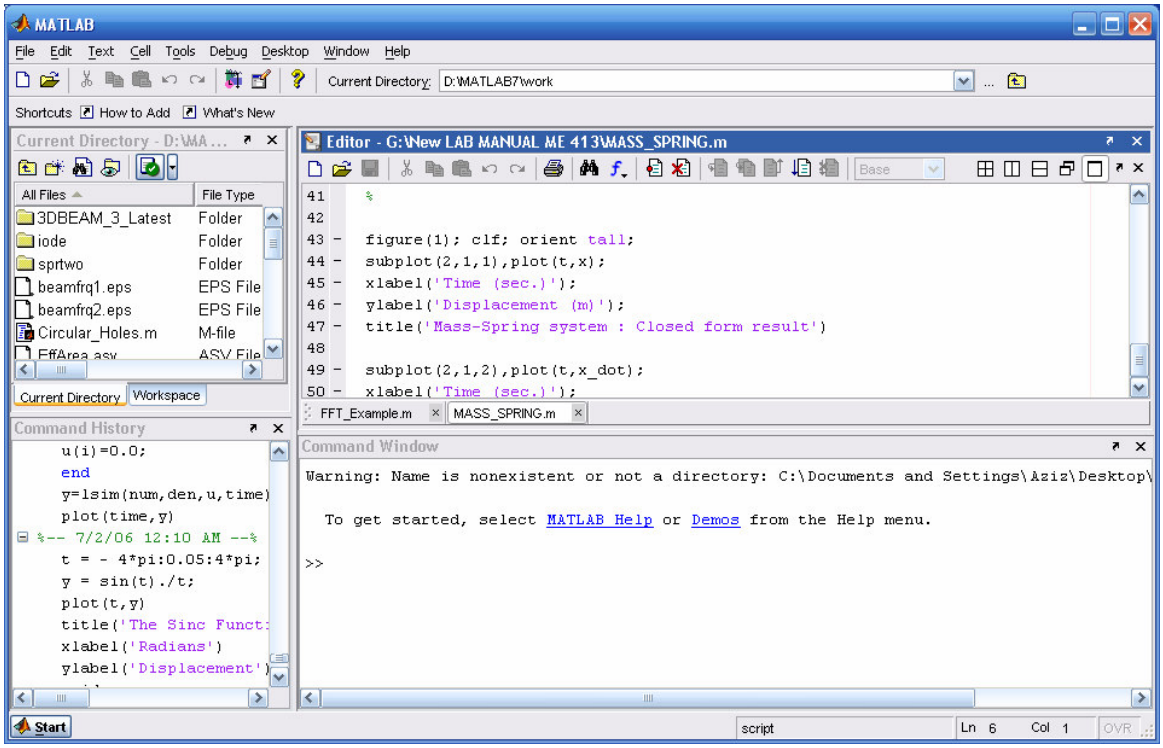


Figure 1 The default view of MATLAB desktop

Table 1 MATLAB Windows

Window	Purpose
Command Window	Main Window, enters variables, runs programs
Figure Window	Contains output from graphic commands
Editor Window	Creates and debugs script and function files
Help Window	Provides help information
Launch Pad Window	Provides access to tools, demos, and documentation
Command History Window	Logs commands entered in the command window
Workspace Window	Provides information about the variables that are used
Current Directory Window	Shows the files in the current directory

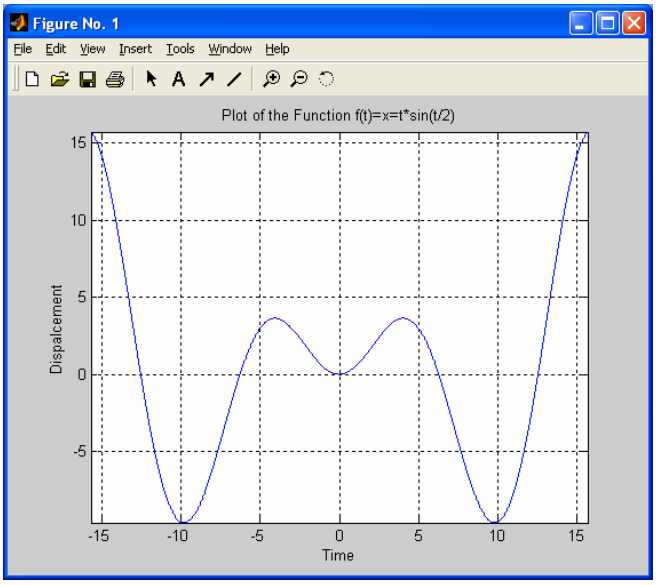


Figure 2 Example of a Figure Window

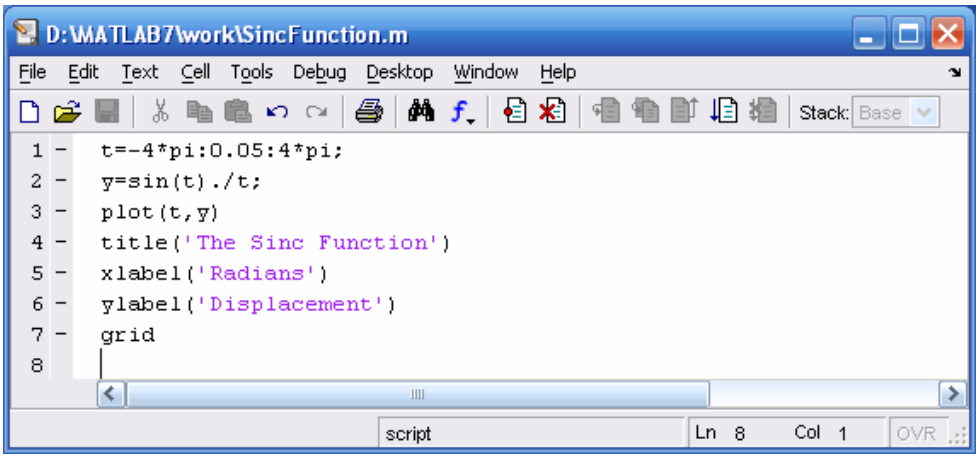


Figure 3 Example of an Editor Window

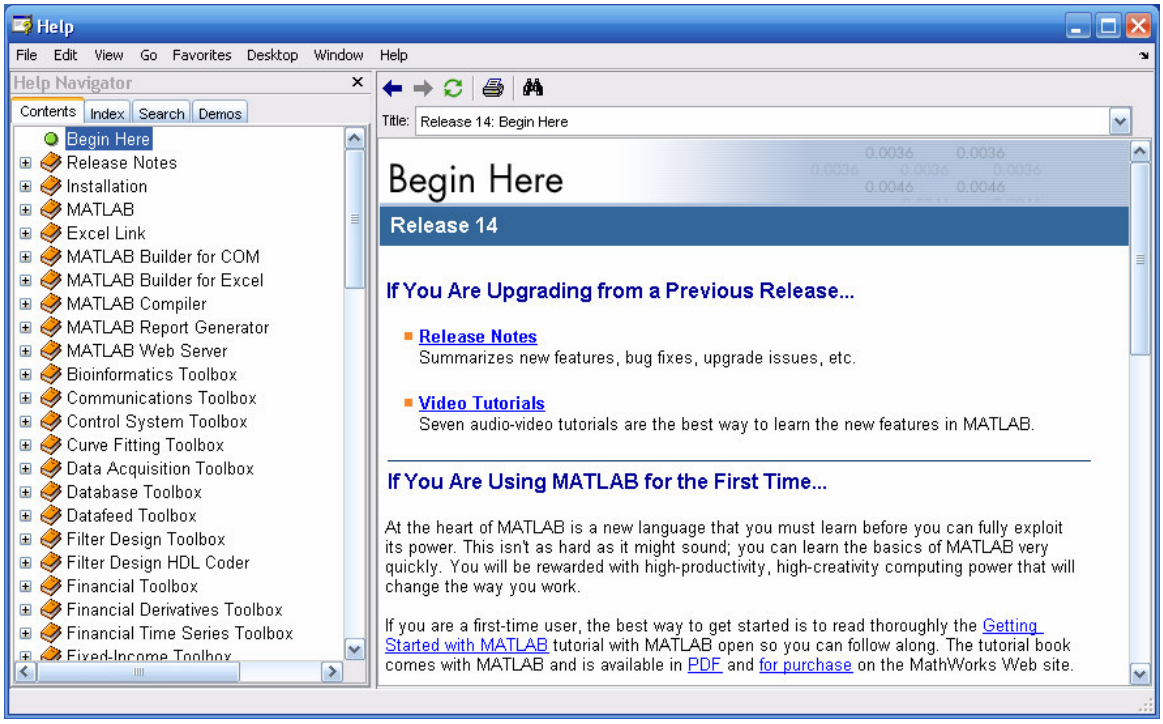


Figure 4 The Help Window

Notes for working in the Command Window

- To type a command the cursor must be placed next to the command prompt (**>>**).
- Once a command is typed and the Enter key is pressed, the command is executed. However, only the last command is executed. Everything executed previously is unchanged.
- Several commands can be typed in the same line. This is done by typing a comma between the commands. When the Enter key is pressed the commands are executed in order from left to right.
- It is not possible to go back to a previous line in the command window, make a correction, and then re-execute the command.
- A previously typed command can be recalled to the command prompt with the up-arrow key **(↑)**. When the command is displayed at the command prompt, it can be modified if needed and executed. The down-array key **(↓)** can be used to move down the previously typed commands.
- I f a command is too long to fit in one line, it can be continued to the next line by typing three periods … (called an ellipsis) and pressing the Enter key. The continue line after line up to a total of 4096 characters.

- The semicolon **(;)**: When a command is typed in the command window and the Enter key is pressed, the command is executed. Any output that the command generates is displayed in the command window. If a semicolon (;) is typed at the end of a command the output of the command is not displayed. Typing a semicolon is useful when the result is obvious or known, or when the output is very large.

- MATLAB has an on-line help facility that may be invoked whenever need arises. The command help will display a list of predefined functions and operator for which on-line help is available.

  >> help 'function name': gives information on the specific function.

2. COMMANDS AND MATRIX FUNCTIONS USED IN MATLAB

2.1 Matrix Operators

Table 2 shows the different matrix notations that are used in the matrix operations.

Table 2 MATLAB Operators

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
.*	Element by element multiplication of arrays
/	Right division
\	Left division
^	Exponentiation
=	Assignment operator
'	Conjugate transpose

2.2 Relational and Logical Operators

Table 3 shows the various relational and logical operators that are used in MATLAB.

Table 3 MATLAB Relational and Logical Operators

Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
~=	Not equal
&	Logical AND
|	Logical OR
~	Logical NOT

2.3 Special Characters, Managing Commands and Predefined Variables

Tables 4, 5 and 6 show respectively the different special characters, managing commands, and predefined variables that are used in MATLAB.

Table 4 MATLAB Special Characters

Character	Meaning
[ ]	Used to form vectors and matrices
( )	Arithmetic expression precedence
`	Separate subscripts and function arguments
;	End rows, suppress printing
:	Subscripting, vector generation
!	Execute operating system command
%	Comments
…	Continuation of line

Table 5 MATLAB Managing Commands

Command	Description
cd	Changes current directory
clc	Clears the command window
clear all	Removes all variables from the memory
clear x y z	Removes variables x, y, and z from the memory
fclose	Close a file
fopen	Opens a file
global	Declares global variables
help	Displays help for MATLAB functions
lookfor	Search for specified word in all help entries
who	Displays variables currently in the memory
whos	Displays information on variables in the memory

Table 6 MATLAB Predefined Variables

Variable	Description
ans	Value of last expression
eps	The smallest difference between two numbers
i	$\sqrt{-1}$
inf	Infinity
j	Same as i
NaN	Not a number
pi	The number π

3. BASIC CALCULATIONS

3.1 Simple Math

MATLAB can be used as a calculator.

■ **Example** Calculate the expression $3+\sqrt{7}-(5/2)+3^2$

>> 3+sqrt(7)–(5/2)+3^2

ans = 12.1458

The variable **ans** is assigned to the previous result if no assignment is made. Normally variables are used and assigned values or results.

■ **Example** As an alternative, the above can be solved by storing information in MATLAB variables

>> erasers=4; pads=6;tapes=2;
>> items=erasers+pads+tapes

items =12

>> cost=erasers*25.75+pads*22+tapes*99.5
>> average_cost=cost/items

cost = 434

average_cost = 36.1667

Here we created three MATLAB variables erasers, pads and tapes to store the number of each item. After entering each statement, MATLAB displayed the results except in the case of tapes, pads and tapes.

>> erasers=4; pads=6;tapes=2;

3.2 Number Display Format

When MATLAB displays numerical results, it follows several rules. By default, if a result is an integer, MATLAB displays it as an integer. Likewise, when a result is a real number, MATLAB displays it with approximately 4 digits to the right of the decimal point. You can override this default behavior by specifying a different format using the Numeric Format menu item in the Options menu if available or by typing the appropriate MATLAB command at the prompt. Using the variable average_cost from the above example, some of these numerical formats are

3.3 About Variables

Like any other computer language, MATLAB has rules about variables and names. Earlier it was noted that variable names must be a single word containing no spaces. More specifically, MATLAB variables naming rules are

Table 7 MATLAB Formats

MATLAB Command	average_cost	Comments
format long	36.16666666666666	16 digits
format short e	3.6167e+001	5 digits plus exponent
format long e	3.616666666666666e+001	16 digits plus exponent
format bank	36.17	2 decimal digits
format rat	217/6	rational approximation
format short	36.1667	default display

Table 8 Variables in MATLAB

Variable Naming Rules	Comments/Examples
Variable names are case sensitive	Items, items, itEms and ITEMS are all different MATLAB variables
Variable names can contain up to 19 characters; characters beyond the 19th are ignored	how_about_this_variable_name
Variable names must start with a letter followed by any number of letters, digits, or underscores.	X51483
Punctuation characters are not allowed since many have special meaning to MATLAB.	a_b_c_d

3.4 Complex Numbers

The imaginary unit $\sqrt{-1}$ is predefined by two variables i and j. In a program, if other values are assigned to i and j, they must be redefined as imaginary units, or other characters can be defined for the imaginary unit.

>> j = sqrt(−1) or **i = sqrt(−1)**

Once the complex unit has been defined, complex numbers can be generated.

■ **Example** Evaluate the function $V = Z\cosh(g) + \sinh(g)/Z$ where $Z = 200 + j300$ and $g = 0.02 + j1.5$

```
>> j = sqrt(−1);
>> Z = 200 + 300*j;
>> g = 0.02+1.5*j;
>> V = Z*cosh(g) + sinh(g)/Z

V = 8.1672 +25.2172j
```

3.5 Mathematical Functions

A partial list of the common functions that MATLAB supports is shown in the Table of the Appendix. Most of these functions are used in the same way you would write them mathematically.

■ **Example** The commands below find the angle x where $\sin(x) = \sqrt{2}/2$.

```
>> x=sqrt(2)/2
       x =0.7071
>> y=asin(x)
       y =0.7854
>> y_deg=y*180/pi      %     converts the angle from radian to degrees
       y_deg = 45.0000
```

Some of the functions, like **sqrt** and **sin** are built-in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like **gamma** and **sinh**, are implemented in M-files. You can see the code and even modify it if you want. Several special functions provide values of useful constants. (See the Table of the Appendix for the listing of some MATLAB functions).

Infinity is generated by dividing a nonzero value by zero, or by evaluating well-defined mathematical expressions that overflow, i.e., exceed realmax. Not-a-number is generated by trying to evaluate expressions like 0/0 or Inf-Inf that do not have well defined mathematical values.

4. MATRICES

A matrix is a set of numbers arranged in a rectangular grid of rows and columns. When we use a matrix, we need a way to refer to individual elements or numbers in the matrix. A simple method for specifying an element in the matrix uses the row and the column number. The size of a matrix is specified by the number of rows and columns. If a matrix has the same number of rows and columns, it is called a square matrix. Table 7 depicts the basic Matrix functions.

Table 9 Basic Matrix Functions

Symbol	Explanation
inv	Inverse of a matrix
det	Determinant of a matrix
rank	Rank of a matrix
cond	Condition number of a matrix
Eye(n)	The n X n identity matrix
trace	Summation of diagonal elements of a matrix
zeros(m,n)	The m X n matrix whose elements are zeros
Ones(m,n)	The m X n matrix whose elements are ones

4.1 Entering Matrices in Matlab

In MATLAB a matrix is created with a rectangular array of numbers surrounded by square brackets. The elements in each row are separated by blanks or commas. The end of each row, except the last row, is indicated by a semicolon. Matrix elements can be any MATLAB expression. The statement

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

results in the output

```
A=
     6    1    2
    -1    8    3
     2    4    9
```

If a semicolon is not used, each row must be entered in a separate line as shown below

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

The entire row or column of a matrix can be addressed by means of the symbol (:). For example: The command to display the third raw would be

```
>> r3 = A(3,:)
```

results in

```
r3 =
 2  4  9
```

Similarly, the statement A (:,2) addresses all elements of the second column in A. For example

```
>> c2 = A (:,2)
```

results in

```
c2 =
    1
    8
    4
```

4.2 Determinant of a Matrix

A determinant is a scalar computed from the entries in a square matrix. Determinants have various applications in engineering, including computing inverses and solving system of simultaneous Equations. For a 2×2 matrix A, the determinant is

$$|A| = a_{11}a_{22} - a_{21}a_{12}$$

For a 3×3 matrix A, the determinant is

$$|A| = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

The determinant of a matrix A can be carried out by MATLAB through the command **det(A)**

>> det(A)

ans = 335

4.3 Transpose of a Matrix

The transpose of a matrix is a new matrix in which the rows of the original matrix are the columns of the new matrix. We use a superscript T after a matrix name to refer to the transpose, **B = A^T**. In MATLAB, the apostrophe (prime) ' denotes the transpose of a matrix. If B is the transpose of A then

>> B = A'

will produce the following matrix

B =

6	**-1**	**2**
1	**8**	**4**
2	**3**	**9**

4.4 Inverse of a Matrix

By definition, the inverse of a square matrix A is the matrix **A^{-1}** for which the matrix product AA^{-1} and A^{-1}A are both equal to the identity matrix. The inverse of an ill-conditioned or singular matrix does not exist. The inverse of the matrix A is performed with the function **inv(A)**

>> C = inv(A)

ans =

0.1791	**0.0030**	**0.0388**
0.0448	**0.1493**	**0.0597**
0.0597	**0.0657**	**0.1463**

4.5 Basic Operations in Matrices

Matrices of the same dimension can be added or subtracted. Two matrices A and B can be multiplied together to form the product AB if they are conformable (the number of columns of A is equal to the number of rows of B).

Two symbols are used for non-singular matrix division. A\B is equivalent to $A^{-1}B$, and A/B is equivalent to $B^{-1}A$. For example,

```
>> A = [6   1    2;–1  8    3;2   4    9];
>> B = [3   5    0;5   4    1;0   –2   2];
```

To add two matrices A and B, simply type

```
>> A + B

ans =

     9     6     2
     4    12     4
     2     2    11
```

Similarly, to multiply two matrices A and B, simply type

```
>> A * B
ans =

    23    30     5
    37    21    14
    26     8    22
```

Dividing by matrices is also straightforward once you understood how MATLAB interprets the divide symbols / and \. This can be illustrated into the next topic.

4.6 Solving a System of Equations

Consider the following system of three equations with three unknowns

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3
\end{aligned}
$$

where x_i are the unknowns and a_{ij} and b_i are known coefficients. The previous system can be written in matrix form as

$$
\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}
$$

which can be written in compact form as

$$
[A]\{X\} = [B]
$$

where

$$[A]=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \qquad \{X\}=\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}, \qquad [b]=\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

A system of equations is nonsingular if the matrix [A] containing the coefficients of the equations is nonsingular. Solving the previous equation for the unknown $\{X\}$ yields

$$\{X\}=[A]^{-1}[B]$$

In MATLAB, a system of simultaneous equations can be solved using matrix division. The solution to the matrix equation $[A]\{X\}=[B]$ can be computed using matrix left division, as in A\B. The solution to the matrix equation $\{X\}[A]=[B]$ can be computed using matrix right division, as in B/A. (MATLAB uses a Gauss elimination technique to perform both left and right matrix division).

■ **Example** Solve the following system of equations

$$\begin{aligned} 3x_1 + 2x_2 - x_3 &= 10 \\ -x_1 + 3x_2 + 2x_3 &= 5 \\ x_1 - x_2 - x_3 &= -1 \end{aligned}$$

where

$$[A]=\begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix}, \qquad \{X\}=\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}, \qquad [b]=\begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

This is shown in MATLAB as

```
>> A=[3   2   -1;-1   3   2;1   -1   -1];
>> B=[10   5   -1]';
>> x=A\B

X=
        -2.0000
         5.0000
        -6.0000
```

The vector x then contains the following values: –2, 5, –6. We can also define and solve the same system of equations using the matrix equation $\{X\}[A]=[B]$ as shown in this statement.

```
>> x= B/A
```

produces the same result. The vector x then contains the following values: -2, 5, -6. If a set of equations is singular, an error message is displayed; the solution vector may contain values of NaN (Not a Number) or ∞.

4.7 Eigenvalues and Eigenvectors

If [A] is $n \times n$ matrix, the n numbers λ that satisfy

$$[A]\{x\} = \lambda\{x\}$$

are the eigenvalues of [A]. They are found using eig(A), which returns the eigenvalues in a column vector.

Eigenvalues and associated eigenvectors can be obtained with a double assignment statement [X, D] = eig (A). The diagonal elements of D are the eigenvalues and the columns of X are the corresponding eigenvectors such that AX = XD.

■ **Example:** Find the eigenvalues and the associated eigenvectors of the matrix [A] given by

$$A = \begin{bmatrix} 0 & 1 & -1 \\ -6 & -11 & 6 \\ -6 & -11 & 5 \end{bmatrix}$$

```
>> A = [0   1   – 1; – 6   – 11   6; – 6   – 11   5];
>> [X,D] = eig(A)

X =

  – 0.7071   0.2182   – 0.0921
    0.0000   0.4364   – 0.5523
  – 0.7071   0.8729   – 0.8285

D =

     –1.0000
     –2.0000
     –3.0000
```

4.8 Utility Matrices

We often need to enter an identity matrix I in MATLAB programs. A statement eye(n) gives an $n \times n$ identity matrix. That is,

```
>> eye(4)

ans =

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

4.9 Diagonal Matrix

If x is a vector, a statement diag(x) produces a diagonal matrix with x on the diagonal line. For example, for a vector **x=ones(1,n)**, the command **diag(ones(1,5))** gives an $n \times n$ identity matrix as follows

```
>> diag(ones(1,5))

ans =

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

If A is a square matrix, then a diag(A) is a vector consisting of the diagonal of A, and diag(diag(A)) is a diagonal matrix with elements of diag(A) appearing on the diagonal line. See the following MATLAB output

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

ans =

     1
     5
     9

>> diag(diag(A))

ans =

     1     0     0
     0     5     0
     0     0     9
```

4.10 The Magic Function

MATLAB actually has a built-in function that creates magic squares of almost any size. Not surprisingly, this function is named magic.

```
>> B = magic(3)

B =

     8     1     6
     3     5     7
     4     9     2

>> B = magic(4)

B =
```

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

PRACTICE PROBLEMS

Exercise 1: Give the contents of the following matrices. Then check your answers by entering the MATLAB commands. Use the following matrix

$$[G] = \begin{bmatrix} 0.6 & 1.5 & 2.3 & -0.5 \\ 8.2 & 0.5 & -0.1 & -2.0 \\ 5.7 & 8.2 & 9.0 & 1.5 \\ 0.5 & 0.5 & 2.4 & 0.5 \\ 1.2 & -2.3 & -4.5 & 0.5 \end{bmatrix}$$

a) A=G(:,2)
b) C=10:15
c) D=[4:9; 1:6]
d) F=0.0:0.1:1.0
e) T1=G(4:5,1:3)
f) T2=G(1:2:5,:)

Exercise 2: Assume that the array $[C]$ is defined as shown, and determine the contents of the following subarrays:

$$[C] = \begin{bmatrix} 1.1 & -3.2 & 3.4 & 0.6 \\ 0.6 & 1.1 & -0.6 & 3.1 \\ 1.3 & 0.6 & 5.5 & 0.0 \end{bmatrix}$$

a) C(2,:)
b) C(:,end)
c) C(1:2,2:end)
d) C(6)
e) C(4:end)
f) C(1:2,2:4)
g) C([1 4],2)
h) C([2 2],[3 3])

5. POLYNOMIALS

A polynomial of a single variable can be expressed in the following general form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$$

where the variable is x and the polynomials' coefficients are represented by the values of $a_n, a_{n-1}, a_{n-2}, \ldots, a_1, a_0$. The degree of a polynomial is equal to the largest value used as an exponent.

5.1 Polynomials Representation in MATLAB

In MATLAB, polynomials are represented by a row vector in which the elements are the coefficients $a_n, a_{n-1}, a_{n-2}, \ldots, a_1, a_0$. The first element is the coefficient of the x with the highest power. The vector has to include all the coefficients, including the ones that are equal to 0. For example,

Table 10 Polynomials Representation in MATLAB

Polynomial	MATLAB Representation
$L(x) = 8x + 5$	$L = [8 \quad 5]$
$P(x) = 2x^2 - 4x + 10$	$P = [2 \quad -4 \quad 10]$
$Q(x) = 6x^3 - 150$	$Q = [6 \quad 0 \quad 0 \quad -150]$
$R(x) = 5x^4 + 6x^2 - 7x$	$R = [5 \quad 0 \quad 6 \quad -7 \quad 0]$

5.2 Polynomial Functions in MATLAB

MATLAB provides suitable tools for handling polynomials. The summary of polynomials functions is provided in Table 2.

Table 11 Polynomial Functions

Command	Description
poly(P)	Converts a collections of roots into a polynomial equation
roots(P)	Finds the roots of a polynomial equation
Polyval(P,m)	Evaluate the polynomial P for a given value m
polyder(P,m)	Differentiate the polynomial P
conv(P,Q)	Multiply two polynomials P and Q
deconv(P,Q)	Decompose a polynomial P into a dividend and a residual
polyfit(P,n)	Curve fitting of a given data to polynomial P of degree n

5.3 Roots of a Polynomial

A polynomial equation can be given by

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0 = 0$$

The roots of the above polynomials are found using roots command

$$\text{roots}\left(\left[a_n \quad a_{n-1} \quad a_{n-2} \quad a_1 \quad a_0\right]\right)$$

Example Find the roots of $x^4 - 12x^3 + 25x + 116 = 0$

The above polynomial is entered in MATLAB as

```
>> P = [1    –12    0      25     16]
>> r = roots(P)
```

The last expression yields the answer as

```
r =
      11.7473
       2.7028
      –1.2251  + 1.4672i
      –1.2251  – 1.4672i
```

Example Find the roots of $x^4 - 5x^2 + 10x = 0$

The above polynomial is entered in MATLAB as

```
>> P = [1    0      -5     10     0]
>> r = roots(P)
```

The last expression yields the answer as

```
r =
      0.0000
      2.9055
      1.4527 + 1.1538i
      1.4527 – 1.1538i
```

5.4 Generation of a Polynomial Equation Using Roots

Since both a polynomial and its roots are vectors in MATLAB, MATLAB adopts the convention that polynomials are row vectors and roots are column vectors. Given the roots of a polynomial, it is also possible to construct the associated polynomial. In MATLAB, the command poly performs this task

The poly commend takes the roots and converts them into a polynomial equation. For instance, if we know $\left(\left[r_1, \quad r_2, \quad r_3, \ldots, \quad r_n\right]\right)$ in

$$(x-r_1)(x-r_2)\ldots(x-r_2) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

then

>> **poly** $\left(\left[r_1, r_2, r_3, \ldots, r_n\right]\right)$ provides the coefficients $\left[a_n, a_{n-1}, a_{n-2}, \ldots, a_0\right]$ of the polynomial equation.

Example Given the following roots of a polynomial $r = 1, r = -5, r = 3$, find the polynomial

The above roots are entered as

```
>> r = [1    –5    3]
>> p = poly(r)
```

The last expression yields the answer as

```
P =
          1     1   –17    15
```

The above expression means that the polynomial p is simply

$$P(r) = r^3 + r^2 - 17r + 15$$

5.5 Polynomial Multiplication

The conv command is used to multiply two polynomials.

Example Given $a(s) = s^3 + 2\,s^2 + 3\,s + 4$ and $b(s) = s^3 + 2\,s^2 + 9\,s + 16$

The multiplication $c(s) = a(s) \cdot b(s)$ is shown below

```
>> a = [1    2     3     4]; b=[1    2     9     16];
>> c = conv(a,b)
```

gives

```
c =
        1   4    16    44    67    84    64
```

The above expression means that the polynomial **c** is simply

$$c(s) = s^6 + 4s^5 + 16s^4 + 44s^3 + 67s^2 + 84s + 64$$

Multiplication of more than two polynomials requires repeated use of **conv**.

Example Given:

$$a(s) = s^2 + 3s - 1,\ b(s) = s^3 - 2s^2 + 6s - 7 \text{ and } c(s) = 2s^2 - 6s + 10$$

The multiplication $P(s) = a(s) \cdot b(s) \cdot c(s)$ is shown below

```
>> a = [1    3      -1];
>> b=[1     -2     6      -7];
>> c=[2     -6     10];
>> P = conv(a,conv(b,c))        or      P = conv(conv(a,b),c)
```

gives

```
P =
        2     -4     2     42    -142    306    -312    70
```

The above expression means that the polynomial p is simply

$$P(s) = 2s^7 - 4s^6 + 2s^5 + 42s^4 - 142s^3 + 306s^2 - 312s + 70$$

5.6 Polynomial Addition

MATLAB does not provide a direct function for adding polynomials. Standard array addition works if both polynomial vectors are the same size. Add the polynomial a(x) and b(x) given above.

Example Given: $a(s) = s^2 + 3s - 1$, and $b(s) = 2s^2 - 6s + 10$

The addition $c(s) = a(s) + b(s)$ is shown below

```
>> a = [1    3    -1];
>> b = [2    -6    10];
>> c = a + b
```

gives

```
c =
        3    -3    9
```

The above expression means that the polynomial d is simply

$$c(s) = 3s^2 - 3s + 9$$

When two polynomials are of different orders, the one having lower order must be padded with leading zeros to make it have the same effective order as the higher-order polynomial.

Example Given $a(s) = s^2 + 3s - 1$, and $b(s) = s^3 - 2s^2 + 6s - 7$

The addition $c(s)=a(s)+b(s)$ is shown below

```
>> a = [1    3    –1];

>> b =[1    –2     6      –7];
>> c = a + b
```

gives

```
??? Error using ==> plus
Matrix dimensions must agree
```

The above addition can performed by writing the polynomial $a(s)$ in the form

```
>> a = [0    1    3    -1];
>> b = [1    –2    6    –7];
>> c = a+b
```

gives

```
      c =
            1    -1    9    -8
```

The above expression means that the polynomial **d** is simply

$$c(s) = s^3 - s^2 + 9s - 8$$

5.7 Polynomial Division

In MATLAB, the division of polynomials is accomplished with the function **deconv**. **[q,r] = deconv(v,u)** deconvolves vector u out of vector v, using long division. The quotient is returned in vector q and the remainder in vector r such that v = conv(u,q)+r.

Example Using the polynomials c(s) and a(s) of the previous example, find the polynomial division $q(s)=c(s)/a(s)$.

The polynomial division $q(s)=c(s)/a(s)$ is given by

```
>> a = [1…….. 3 ……–1];
>> c = [1    –1     9      –8];
>> [q, r] = deconv(c,a)
```

gives

```
      q =
            1    –4
```

```
r =
     0     0    22   -12
```

The above expression means that the polynomial p is simply

$$q(s) = \frac{s^3 - s^2 + 9s - 8}{s^2 + 3\,s - 1} = s - 4 + \frac{22s - 12}{s^2 + 3\,s - 1}$$

Example
Given **u(s) = [1 2 3 4]** and **c(s) = [10 40 100 160 170 120]**, find $q(s) = c(s)/u(s)$

The polynomial division $q(s) = c(s)/u(s)$ is given by

```
>> u = [1    2    3    4];
>> c = [10   40   100   160   170   120];
>> [q, r] = deconv(c,u)
```

gives

```
q =
    10    20    30

r =
     0     0     0     0     0
```

The above expression means that the polynomial p is simply

$$q(s) = \frac{10s^5 + 40s^4 + 100s^3 + 160s^2 + 170s + 120}{s^3 + 2s^2 + 3\,s + 4} = \underbrace{10s^2 + 20s + 30}_{q(s)}$$

5.8 Polynomial Derivatives

Because differentiation of a polynomial is simple to express, MATLAB offers the function **polyder** for polynomial differentiation

Example Use the function **polyder** to find the derivative of the following polynomial $e(x) = x^6 + 6\,x^5 + 20\,x^4 + 52\,x^3 + 81\,x^2 + 96\,x + 84$

```
>> e=[1   6   20   52   81   96   84]
>> h= polyder(e)
```

gives

```
h =
     6    30    80   156   162    96
```

which is

$$h(x) = 6\,x^5 + 30\,x^4 + 80\,x^3 + 156\,x^2 + 162\,x + 96.$$

5.9 Polynomial Evaluation

In MATLAB, the evaluation of polynomials is accomplished with the function **polyval**.

Example Use the function **polyval** to evaluate the polynomial $e(x)$ given above $e(x) = x^6 + 6\,x^5 + 20\,x^4 + 52\,x^3 + 81\,x^2 + 96\,x + 84$, at the points x = -5 and x = 12

```
>> e= [1   6   20   52   81   96   84]
>> v= polyval(e,–5)

v =
    4504

>> W= polyval(e,12)

v =
4996452
```

6. PLOTTING

MATLAB supports some easy plotting tools. The summary of plotting commands is listed in Table 10.

Table 12 Plotting Commands

Command	Description
plot	Basic plot command
title('txt')	Writes the **txt** as a title centered in the top of the graphics window
xlabel('txt')	Writes the string **txt** as a label centered below the x-axis.
ylabel('txt')	Writes the string **txt** as a label centered below the y-axis.
zlabel('txt')	Writes the string **txt** as a label centered below the z-axis.
text(x,y,txt)	Writes the string **txt** in the graphics window at position (x,y). The coordinates x and y are given in the same scale in which the plot is drawn.
text(x,y,txt,'sc')	Writes the string **txt** at position (x,y) in the graphic window. The coordinates are given with (0.0, 0.0) as the lower left-corner, and (1.0, 1.0) as the top right-hand corner.
gtext	Displays a text string in the current figure window after you select a location with the mouse
axis	Freezes the current axis scaling for subsequent plots. A second execution of the command returns the system to

	automatic scaling.
axis(v)	Specifies the axis scaling using the scaling values in the vector v, which should contain [xmin, xmax, ymin, ymax].
grid	Grid lines for two- and three-dimensional plots
semilogx(x,y)	Generates a plot of the values x and y using a logarithmic scale for x and a linear scale for y.
semilogy(x,y)	Generates a plot of the values x and y using a linear scale for x and a logarithmic scale for y.
loglog(x,y)	Generates a plot of the values x and y using a logarithmic scale for both x and y.

6.1 Plotting Data Points

Example Create a linear x-y plot for the following variables

t	0	1	2	3	4	5	6	7	8	9
y	0	0.5	1	2	4	7	11	14	15.5	16

For a small amount of data, you can type in data explicitly using square brackets.

```
>> t = [0  1  2  3  4  5  6  7  8  9];
>> y = [0  0.5  1  2  4  7  11  14  15.5  16];
>> plot(t,y)
>> title('Plot of t versus y')
>> xlabel('Variable t')
>> ylabel('Function y')
>> grid
```

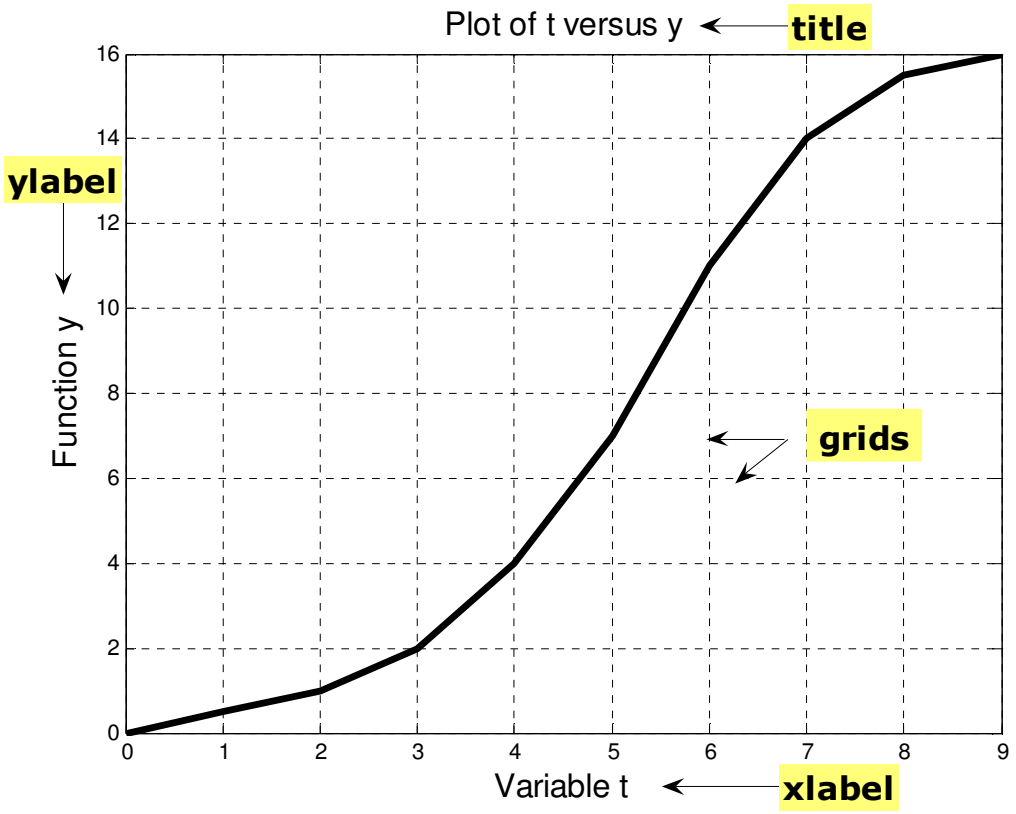


Figure 5 Example of x-y plot

6.2 Plotting Functions

6.2.1 Single Plot

Example Plot the function $y = sin(t)/t$, for $-4\pi < t < 4\pi$.

```
>> t = – 4*pi:0.05:4*pi;  %     starting point:increment:end point
>> y = sin(t)./t;         %     the dot is placed after sin(t) to show that this is a
                          %     scalar division.
>> plot(t,y)
>> title('The Sinc Function')
>> xlabel('Radians')
>> ylabel('Displacement')
>> grid
```

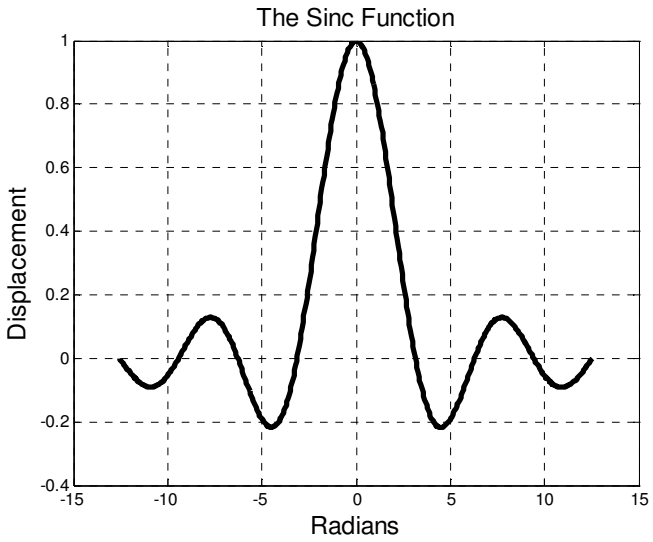


Figure 6 Example of Sinc function plot

6.2.2 Multiple Plots

Example

For $0 < t < 4\pi$, Plot the functions $x = 0.5\,sin(0.5t)$; and $y = 0.5\,cos(0.5t)$.

```
>> clf                       %clear the previous figure windows
>> t = 0:0.05:4*pi;          %starting point:increment:end point
>> x =0.5*sin(0.5*t);        %first function
>> y =.5*cos(0.5*t);         %second function
>> plot(t,x,':',t,y,'-')     %multiple plots, the first function is plotted with
                             %dotted lines while the second one is plotted as solid
                             %lines.
>> title('Plot of Two Function')
>> xlabel('Radians')
>> ylabel('Displacement')
>> grid
>> legend('x','y')           % a legend is plotted to show the different plots
```

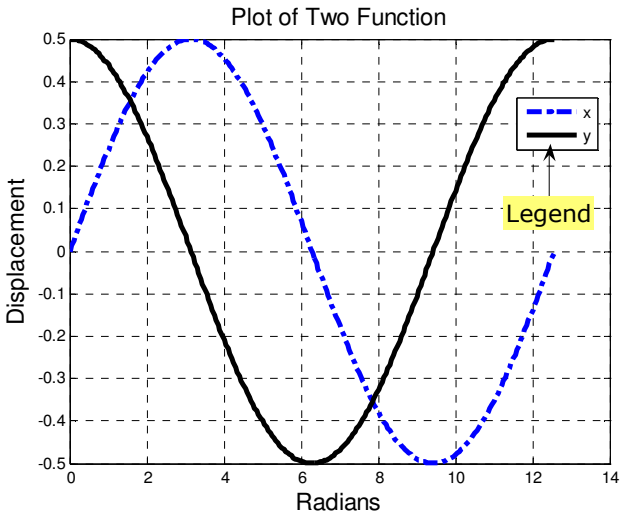


Figure 7 Example of two function plot

6.2.3 SubPlots

The subplot command allows you to split the graph window into subwindows. The arguments to the subplot command are three integers :m, n, p. The digits m and n specify that the graph windows is to be split into an m by n grid of smaller windows, and the digit p specifies the pth windows for the current plot. The windows are numbered from left to right, top to bottom.

Example Use the subplot command to plot the following functions:

```
x = 2 sin(t)+3 sin(2t)+4 sin(3t)+5 sin(4t);
y = (t)*cos(t).*sin(t);
z = cos(2t)+2 cos(3t)+2 cos(3t)*sin(3t);
w=x + y – z;
```

```
>> clf
>> t = linspace(–10,10,1000);
%
>> x = 2*sin(t)+3*sin(2*t)+4*sin(3*t)+5*sin(4*t);
>> y = (t).*cos(t).*sin(t);
>> z = cos(2*t)+2*cos(3*t)+2*cos(3*t).*sin(3*t);
>> w=x+y–z;
%
>> hold on
>> subplot(221);plot(t,x); title('The Function x');
>> subplot(222);plot(t,y); title('The Function y');
>> subplot(223);plot(t,z); title('The Function z');
>> subplot(224);plot(t,w); title('The Function w');
>> hold off
```

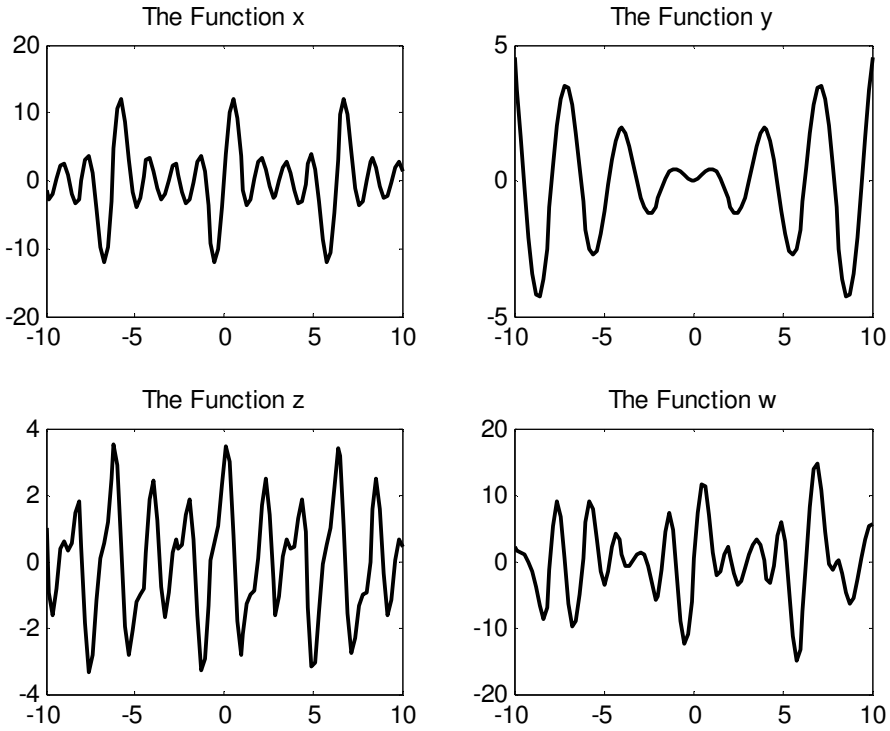


Figure 8. Example of subplot.

6.3 Polynomial Curve Fitting

In general, a polynomial fit to data in vector x and y is a function p of the form

$$p(x) = c_1 x^d + c_2 x^{d-1} + \ldots + c_n$$

The degree of the polynomial is d and the number of coefficients is $n = d + 1$. Given a set of points in vector x and y, **polyfit(x,y,d)** returns the coefficients of **d**th order polynomial in descending powers of x.

Example Find a polynomial of **degree 3** to fit the following data. Find y(5) and y(11).

x	0	1	2	4	6	10
y	1	7	23	109	307	1231

```
>> x = [0   1   2   4   6   10];
>> y = [1   7   23   109   307   1231];
>> p = polyfit(x,y,3)
```

The coefficients of a third degree polynomial are found as follows

```
p =
      1.0000    2.0000    3.0000    1.0000
```

or

$$y = x^3 + 2x^2 + 3x + 1$$

Example Find a polynomial of that would best fit the following data:

x	0	1	2	3	4	5
y	0	20	60	68	77	110

```
>> close all
>> format short
>> x=0:5;
>> y=[0     20     60     68     77     110];
>> coeff=polyfit(x,y,1)
>> ybest=coeff(1)*x+coeff(2)
>> sum_sq=sum((y-ybest).^2)
>> plot(x,ybest,x,y,'o')
>> title('Linear Regression')
>> xlabel('Time, s')
>> ylabel('Temperature,  ({}^\circ  C)')
>> grid
>> axis([-1 6 -20 120])
>> pause(2)
%
>> newx=0:0.05:5;
>> for k=1:5
       f(:,k)=polyval(polyfit(x,y,k),newx)';
       figure(k);plot(newx,f(:,k),x,y,'o');
       title(sprintf('Highest Degree Polynomial is k=%3.0f',k ))
       string = ['Coef. are:' num2str(polyfit(x,y,k))];
       disp(string)
       gtext(sprintf(string))
       xlabel('Time, s')
       ylabel('Temperature,  (C^\circ C)')
       grid
       pause(1)
>> end
```

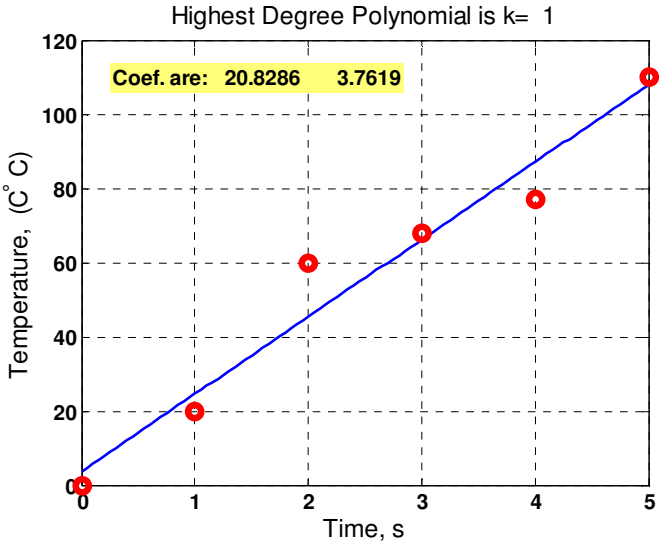


Figure 9

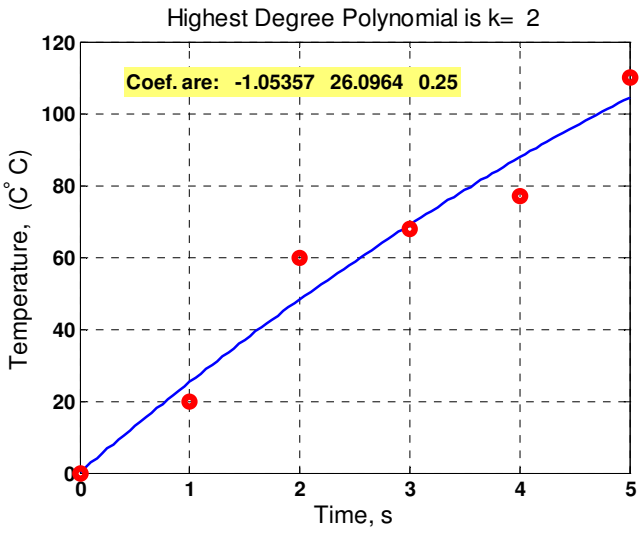


Figure 10

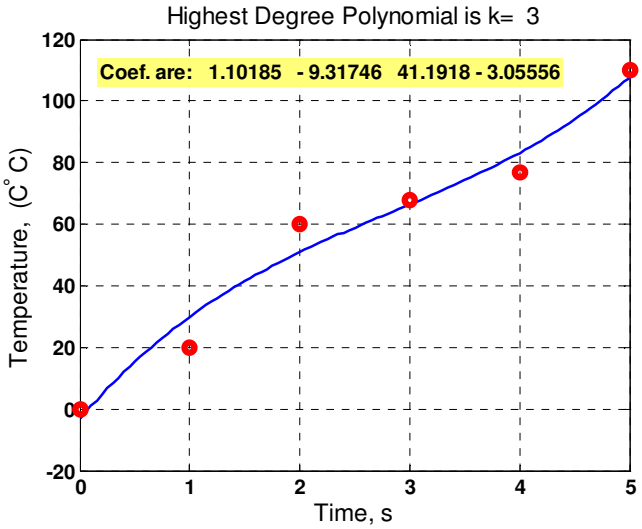


Figure 11

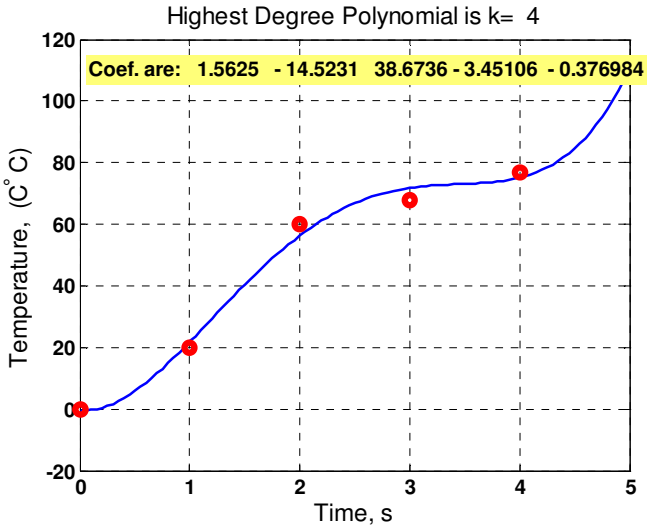


Figure 12

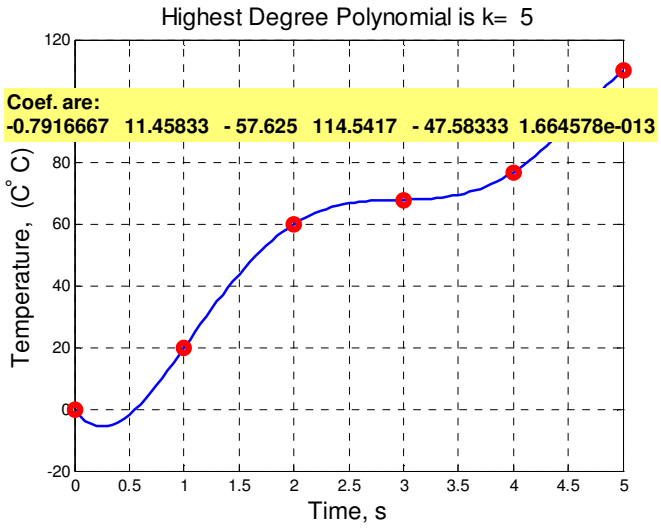


Figure 13

6.4 Three Dimensional Mesh Surface Plot

The statement **mesh(z)** creates a three-dimensional plot of the elements in matrix Z. A mesh surface is defined by the Z coordinates of points above a rectangular grid in the x-y plane. The plot is formed by joining adjacent points with straight lines. meshgrid transforms the domain specified by vector x and y into arrays X and Y.

Use of meshgrid or surface plots

Example Obtain the Cartesian plot of the Bessel function $J_o\sqrt{x^2+y^2}$ over the range $-12 \le x \le 12, \quad -12 \le y \le 12$.

```
>> clf
>> [x,y] = meshgrid(–12:0.6:12,  –12:0.6:12);
>> r = sqrt(x.^2+y.^2);
>> z = bessel(0,r);
>> m = [-45  60];
>> mesh(z,m)
```

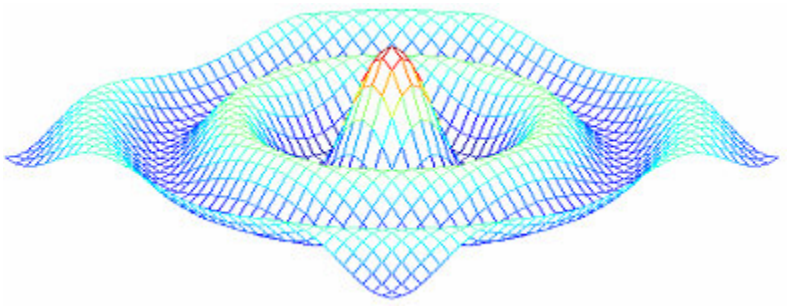


Figure 14 Three dimensional plot.

Example Obtain the 3-D plot of $z = \sqrt{x^2 + y^2}$ over the range $-10 \le x \le 10, \quad -10 \le y \le 10$

%

```
>> [x,y]=meshgrid(-10:0.1:10, -10:0.1:10);
>>z=(x.^2+y.^2);
>>figure(1)
>>mesh(x,y,z)
```

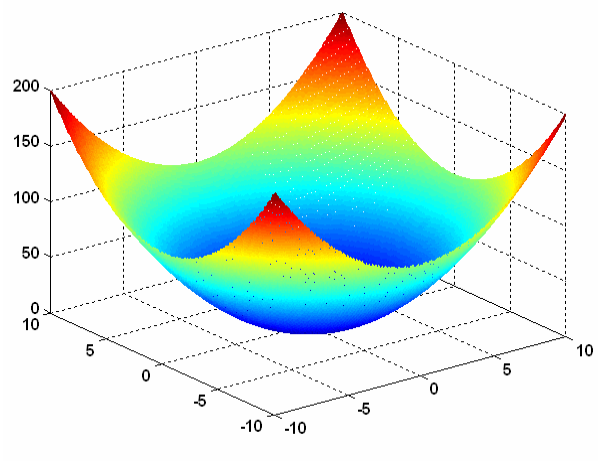


Figure 15

```
>>figure(2)
>>surf(x,y,z)
```

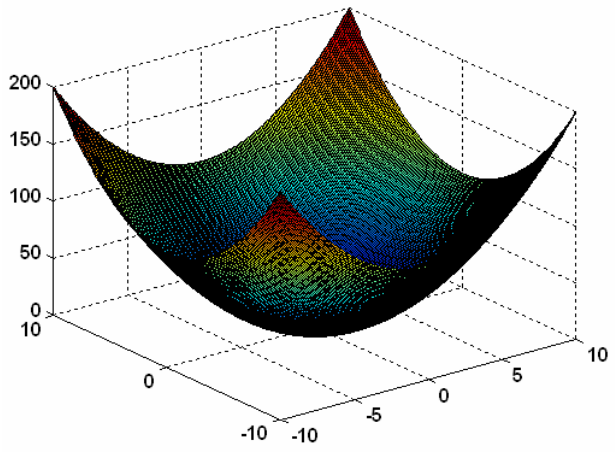


Figure 16

6.5 Line Plots of 3-D Data

The 3-D analog of the plot function is plot3. If x, y, and z are three vectors of the same length, plot3(x,y,z) generates a line in 3-D through the points whose coordinates are the elements of x, y, and z and then produces a 2-D projection of that line on the screen.

Example The following statements produce a helix.

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t)
>> axis square; grid on
```

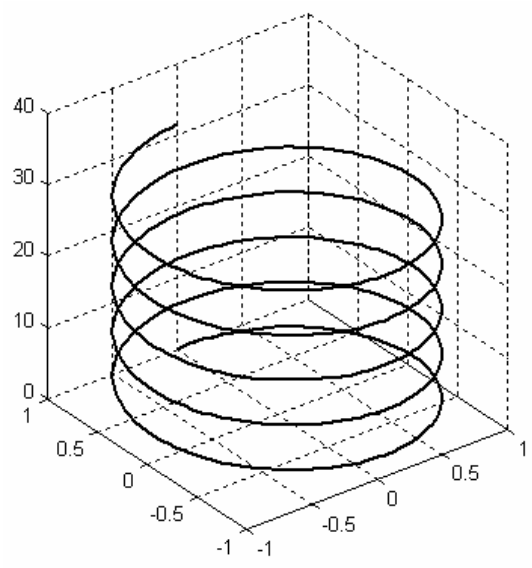


Figure 17 Helix plot

6.6 Plotting Matrix Data

If the arguments to plot3 are matrices of the same size, MATLAB plots lines obtained from the columns of X, Y, and Z. For example,

Example The following lines produce plot obtained from the columns of X, Y, and Z.

```
>> [X,Y] = meshgrid([-2:0.1:2]);
>> Z = X.*exp(-X.^2-Y.^2);
>> plot3(X,Y,Z)
>> grid on
```

Notice how MATLAB cycles through line colors.

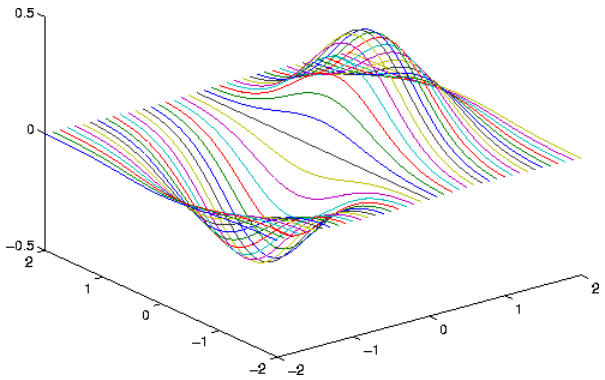


Figure 18 Three dimensional plot

6.7 Line Style, Line Width and Line Style

The command plot(x,y) generates a line plot that connects the points represented by the vectors x and y with line segments. You can also select other line types-dashed, dotted and dash-dot. You can also select a point, plus sign, stars circles or x-mark plots instead of a line plot. Table 13 contains these different options for lines and marks.

Table 13 Line Specifications

Line Type		Point Type		Colors	
-	solid	.	point	y	yellow
--	dashed	+	plus	w	white
:	dotted	*	star	r	red
-.	Dash-dot	o	circle	g	green
		x	x-mark	m	magenta
				I	invisible
				c	cyan
				b	blue
				k	black

7. ADVANCED PLOTTING IN MATLAB

7.1 Plotting of Functions in 2-D

Example plot the real and imaginary part of the following function:

$$y = e^{-0.2t}\left(cos(t) + i\,sin(t)\right),\ i = \sqrt{-1}$$

```
%
%       Both parts of the complex function are plotted
%
>> t=0.0:pi/20:6.3662*pi;
>> y=exp(-0.2*t).*(cos(t)+i*sin(t));
>> figure(2)
>> plot(t,real(y),'b-');
>> hold on
>> plot(t,imag(y),'r--');
>> title('\bfPlot of Complex Function vs Time');
>> xlabel('t');
>> ylabel('y(t)')
>> legend('real', 'imaginary');
>> hold off
%
```

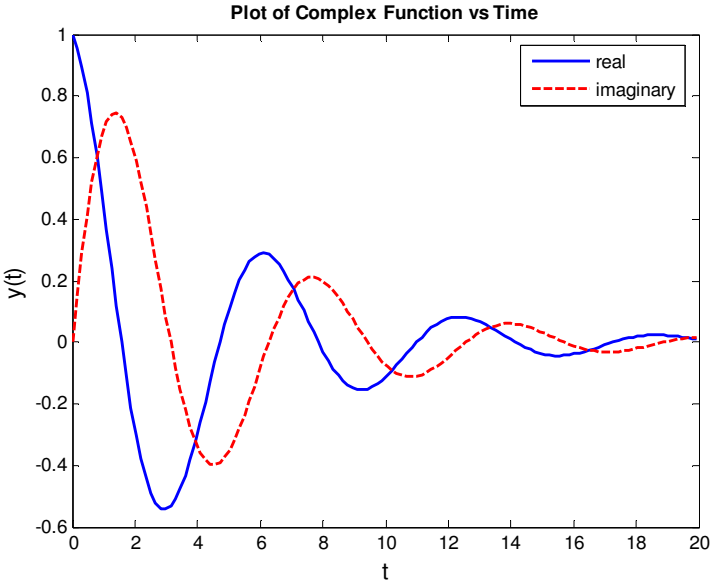


Figure 19

Use of ezplot function

Example Use the ezplot function to plot the hyperbola $x^2 - y^2 = 1$.

```
>> ezplot('x^2 - y^2 - 1')
>> grid on
```

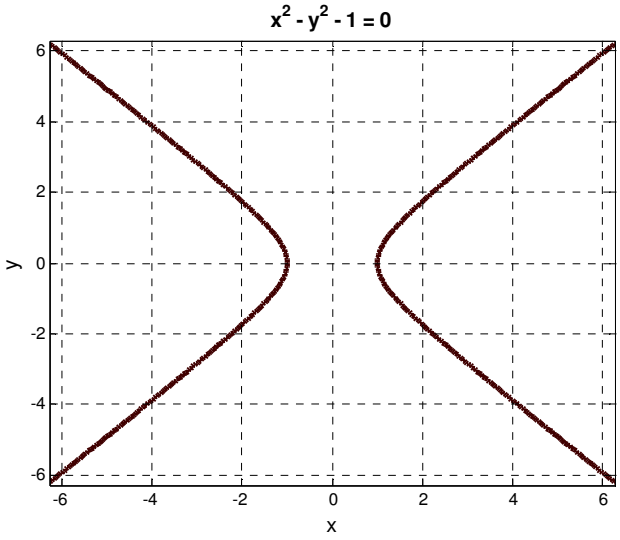


Figure 20

Example Use the ezplot function to plot the circle $x^2 + y^2 = 1$.

```
ezplot('x^2 + y^2 - 1',[-1.25,1.25]); axis equal
grid on
```

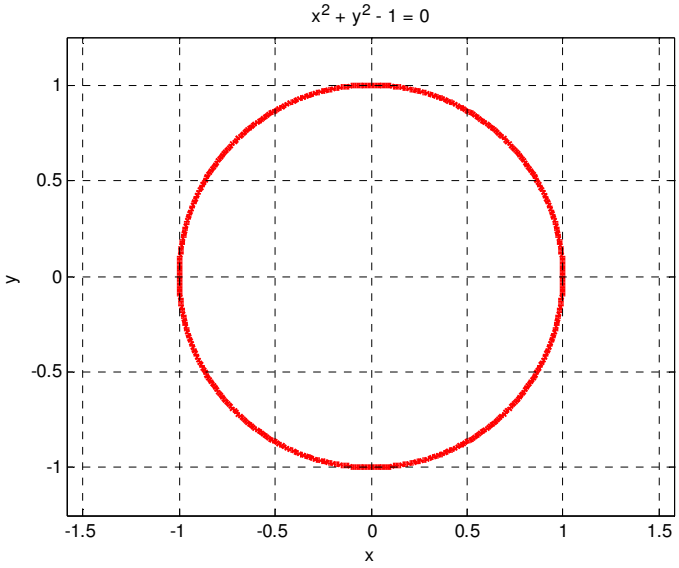


Figure 21

7.2 Plotting of Functions in 3-D

Use of cylinder function

Example Use the cylinder function to plot the hyperbola $= 1.5 + sin(z)$.

```
>> [x,y,z]=cylinder(1.5+sin(0.0:0.25:2*pi),32);
>> surface(x,y,z);
>> view(-35,-34)
>> axis off
```

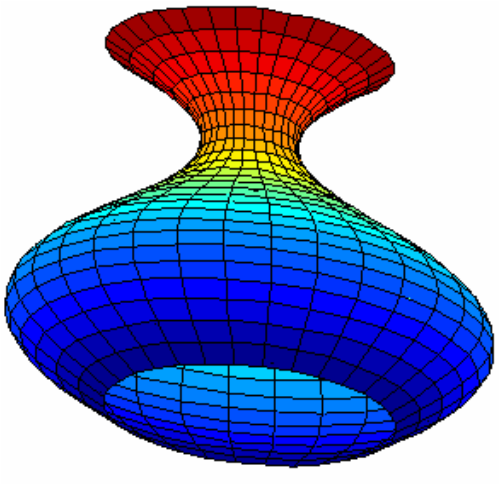


Figure 22

Multiple surface plots

Example Consider the surface $z(r,\theta) = r^3 cos(3\theta)$ where $0 \le r \le 1$ and $0 \le \theta \le 2\pi$. It is assumed that this surface intersects two parallel discs

of radius 1 located at = +-0.5.

```
>> nr=12;
>> nth=50;
>> r=linspace(0,1,nr);
>> theta=linspace(0,2*pi,nth);
>> [R,T]=meshgrid(r,theta);
>> x=cos(theta')*r;
>> y=sin(theta')*r;
>> z=R.^3.*cos(3*T);
>> surface(x,y,z);
>> hold on
>> z0=repmat(0.5,size(x));
>> surface(x,y,z0)
>> surface(x,y,-z0)
>> view(-42.5,20)
```

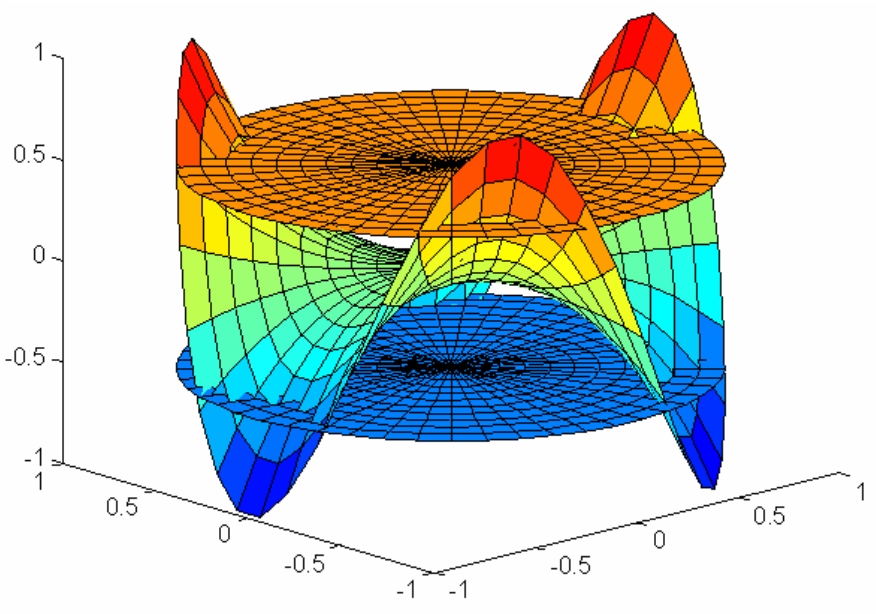
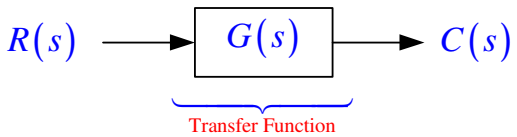


Figure 23

7.3 Response Analysis of a Second Order System to some typical signals

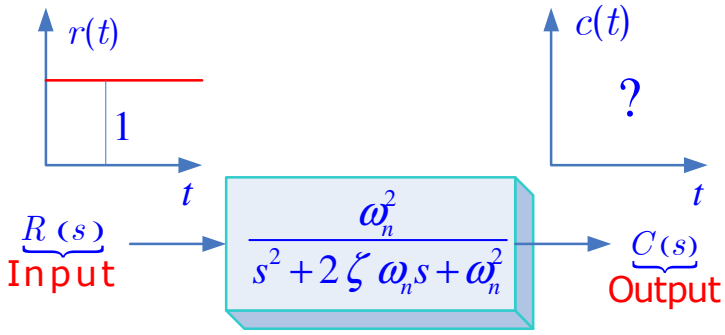
For a given system given by its transfer function $G(s)$ and a given input $R(s)$ what is the output $C(s)$



For a second order system, this can be written as

7.3.1 Step Input

The step function is defined in details in class. It is built in MATLAB. In this case.



MATLAB PROGRAM:

```
>> wn=1;
>> zeta=[0.2   0.5   0.7   1   2   5];
>> for k=1:6
        num=[0   0   wn^2]; den=[1   2*zeta(k)   wn^2];
        sys=tf(num,den); step(sys);
        hold on
>> end
```

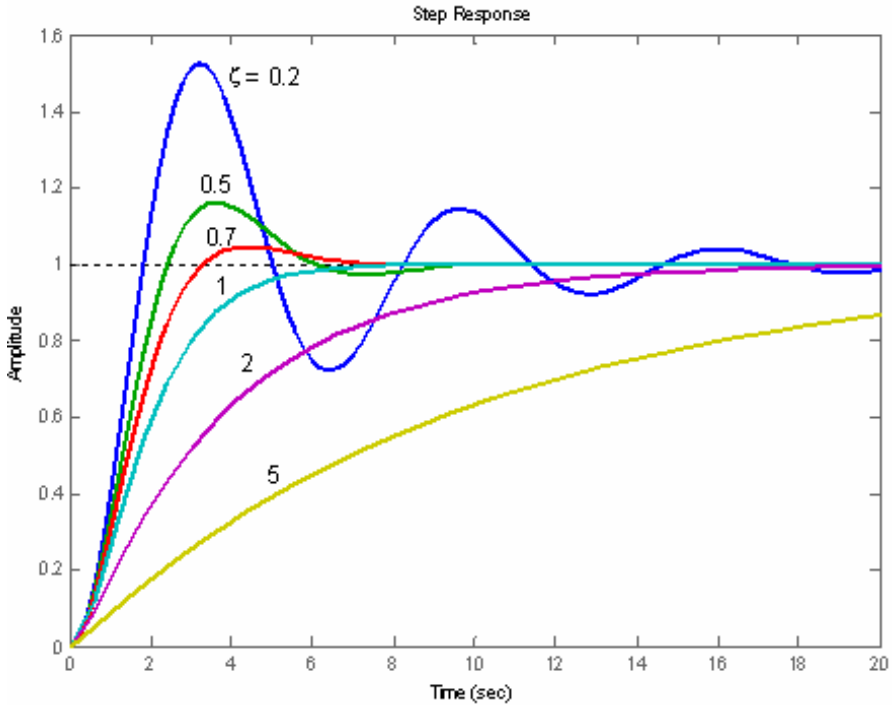
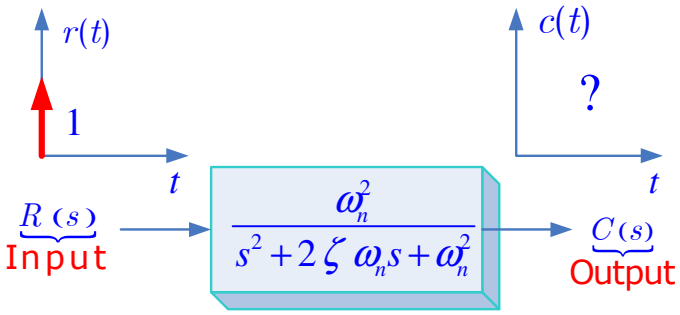


Figure 24

7.3.2 Impulse Input



MATLAB PROGRAM:

```
>> wn=1;
>> zeta=[0.2   0.5   0.7   1   2   5];
>> for k=1:6
       num=[0   0   wn^2];
       den=[1   2*zeta(k)   wn^2];
       sys=tf(num,den)
       impulse(sys);
       hold on
>> end
```

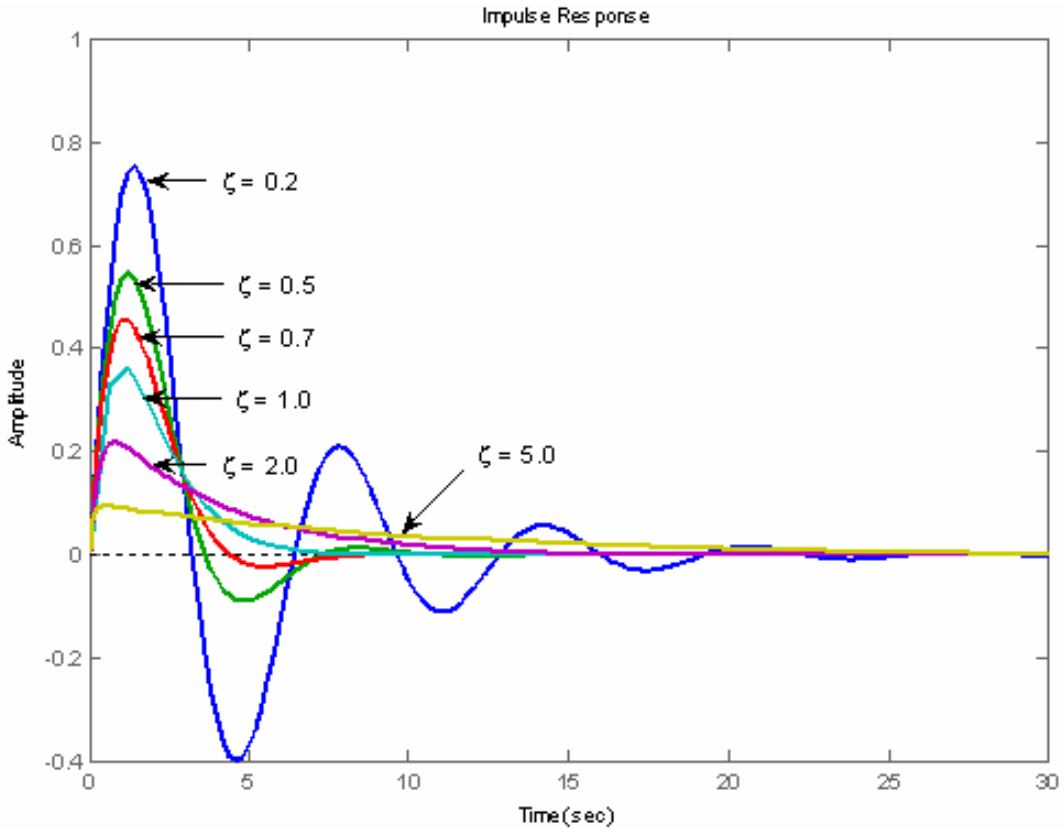
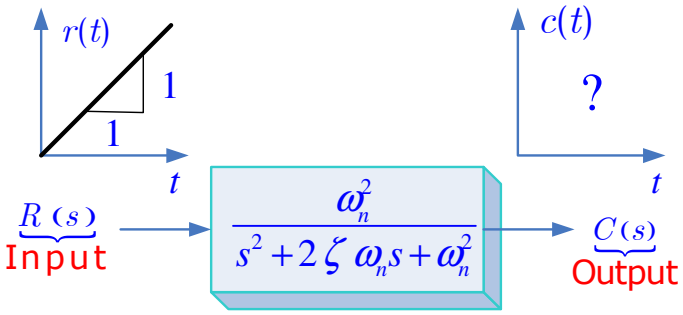


Figure 25

7.3.3 Ramp Input

in this case $U(s) = 1/s^2$



The above equation can be modified to use the step input as follows

$$G(s) = \frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

$$C(s) = \frac{\omega_n^2}{s^2\left(s^2 + 2\xi\omega_n s + \omega_n^2\right)} = \frac{\omega_n^2}{s\left(s^3 + 2\xi\omega_n s^2 + \omega_n^2 s\right)}$$

This is as if we had a transfer function of the form

$$G_1(s) = \frac{C(s)}{R(s)} = \frac{\omega_n^2}{\left(s^3 + 2\xi\omega_n s^2 + \omega_n^2 s\right)}$$

with the input as a step function $R(s) = 1/s$. The following MATLAB script shows a response for a ramp input with a slope of unity and a damping factor of 0.7.

MATLAB PROGRAM:

```
>> wn=1;
>> zeta=0.7;
>> num=[0   0   wn^2];
>> den=[1  . 2*zeta*wn    wn^2    0];
>> sys=tf(num,den)
>> t=0.0:0.0001:10;
>> c=step(num,den,t);
>> plot(t,c,'r-.',t,t)
```

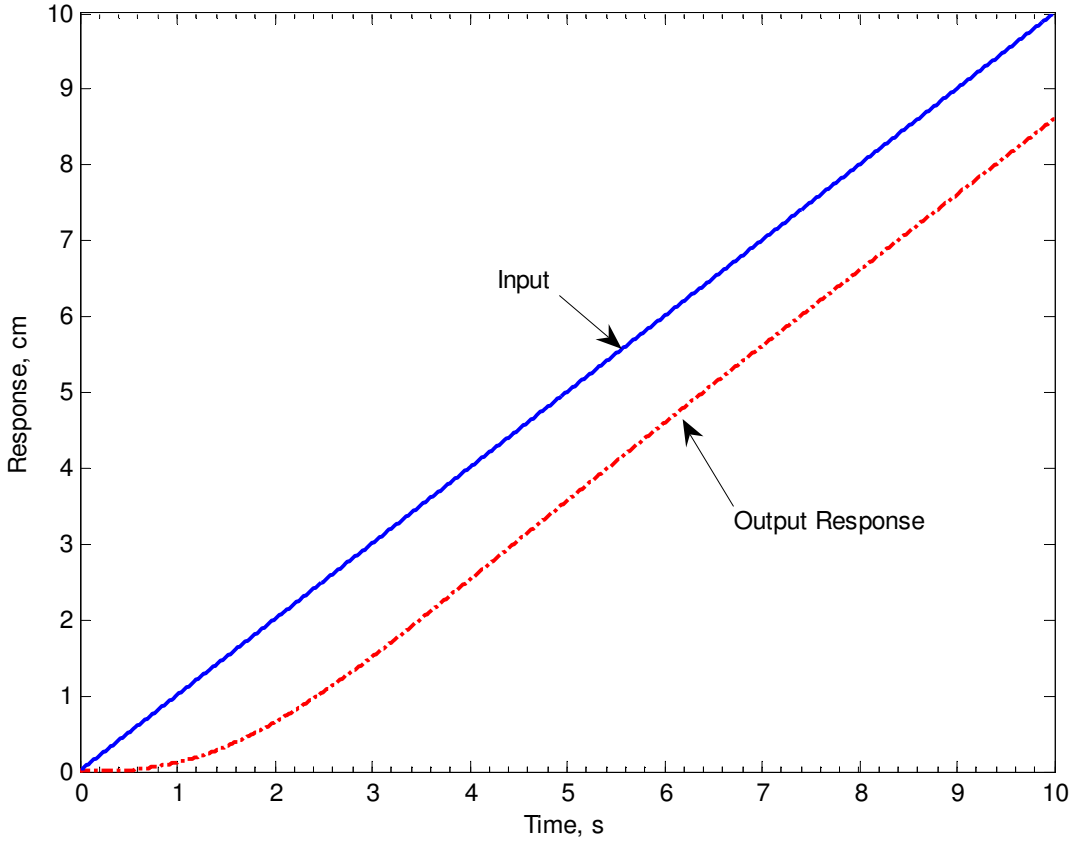


Figure 26

8. PROGRAMMING IN MATLAB

MATLAB contains several commands to control the execution of MATLAB statements, such as conditional statements, loops, and commands supporting user interaction.

Example A script file that produces a sequence of increasingly refined plots of $\sin(2\pi x)$.

```
>> for n = [4   8   12   16   20   50   100   200   500   1000]
            x=linspace(0,pi/2,n);
            y=(sin(2*pi*x));
            plot(x,y)
            title(sprintf('Plot based upon n = %3.0f points.',n))
            pause(1);
>> end
```

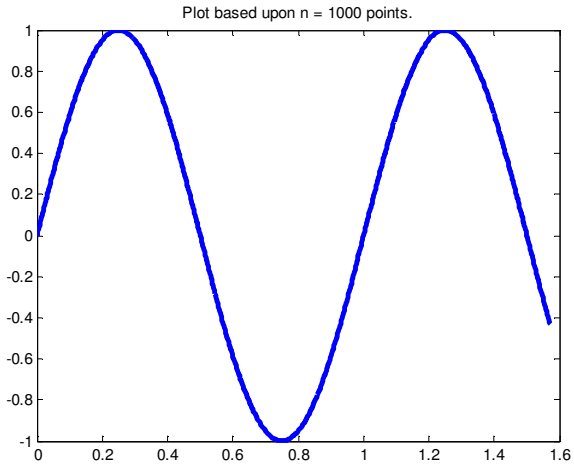


Figure 27 Plot of $\sin(2\pi x)$ through a sequence of increasingly refined points

Example How to create polygons. The following script file produces the graph shown in Figure 28.

```
>> close all
>> clc
>> theta=linspace(0,2*pi,361);
>> c=cos(theta);s=sin(theta);
>> k=0;
>> for sides = [3 4 5 6 8 10 12 18 24]
                stride = 360/sides;
                k=k+1;
                subplot(3,3,k)
                plot(c(1:stride:361),s(1:stride:361))
                xlabel(sprintf(' n = %2.0f',sides));
                axis([-1.2 1.2 -1.2 1.2])
                axis('square')
>> end
```

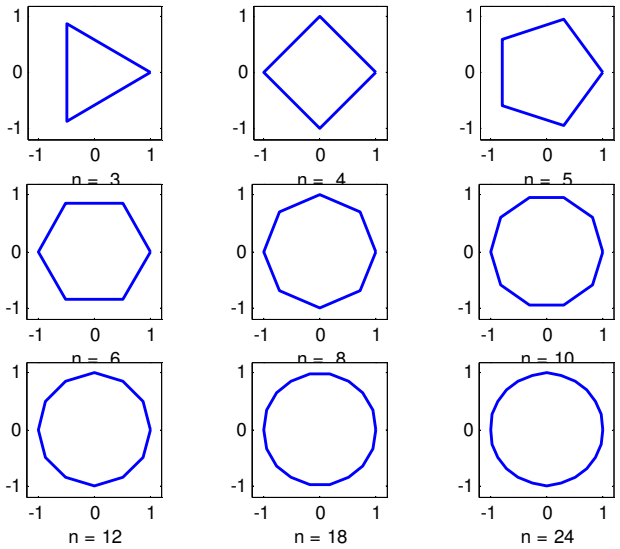


Figure 28. Regular polygons

Example. How to create a STOP sign.

```
>> t = (1/8:2/8:15/8)'*pi;
>> x=sin(t);y=cos(t);
>> fill(x,y,'r')
>> axis('square')
>> text(0,0,'STOP',...
        'Color',[1 1 1],...
        'FontSize',80,...
        'HorizontalAlignment','Center')
```

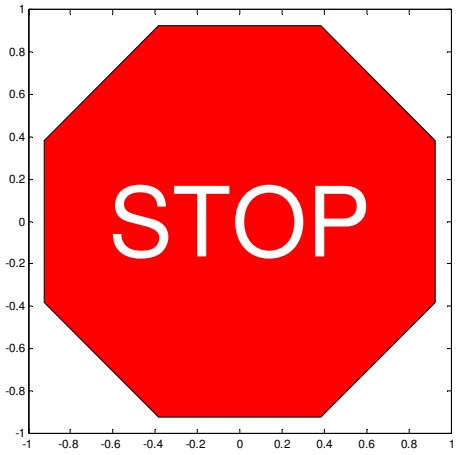


Figure 29 Red stop sign

APPENDIX MATLAB Commands and Matrix Functions

Commands and matrix functions commonly used in solving control engineering problems	Explanations of what commands do, matrix functions mean, or statements mean
abs	Absolute value, complex magnitude
angle	Phase angle
ans	Answer when expression is not assigned
atan	Arctangent
axis	Manual axis scaling
bode	Plot Bode diagram
clear	clear workspace
clg	Clear graph screen
computer	Type of computer
conj	Complex conjugate
cony	Convolution, multiplication
corrcoef	Correlation coefficients
cos	Cosine
cosh	Hyperbolic cosine
cov	Covariance
deconv	Deconvolution, division
det	Determinant
diag	Diagonal matrix
eig	Eigenvalues and eigenvectors
exit	Terminate program
exp	Exponential base e
expm	Matrix exponential
eye	Identity matrix
filter	Direct filter implementation
format long	15-Digit scaled fixed point (Example: 1.33333333333333)
format long e	15-Digit floating point (Example: 1.33333333333333e+000)
format short	5-Digit scaled fixed point (Example: 1.3333)
format short e	5-Digit floating point (Example: 1.3333e+000)
freqs	Laplace transform frequency response
freqz	z-Transform frequency response
grid	Draw grid lines
hold	Hold current graph on the screen
i	$\sqrt{-1}$
imag	Imaginary part
inf	Infinity
inv	Inverse
j	$\sqrt{-1}$
length	Vector length
linspace	Linearly spaced vectors
log	Natural logarithm
loglog	Loglog x-y plot
logm	Matrix logarithm
logspace	Logarithmically spaced vectors
log10	Log base 10
lqe	Linear quadratic estimator design
lqr	Linear quadratic regulator design

max	Maximum value
mean	Mean value
median	Median value
min	Minimum value
NaN	Not-a-number
nyquist	Plot Nyquist frequency response
ones	constant
Pi	Pi (π)
plot	Linear x-y plot
polar	Polar plot
poly	Characteristic polynomial
polyfit	Polynomial curve fitting
polyval	Polynomial evaluation
polyvalm	Matrix polynomial evaluation
prod	Product of elements
quit	Terminate program
rand	Generate random numbers and matrices
rank	Calculate the rank of a matrix
real	Real part
rem	Remainder or modulus
residue	Partial-fraction expansion
rlocus	Plot root loci
roots	Polynomial roots
semilogx	Semilog x-y plot (x-axis logarithmic)
semilogy	Semilog x-y plot (y-axis logarithmic)
sign	Signum function
sin	Sine
sinh	Hyperbolic sine
size	Row and column dimensions
sqrt	Square root
sqrtm	Matrix square root
std	Standard deviation
step	Plot unit-step response
sum	Sum of elements
tan	Tangent
tanh	Hyperbolic tangent
text	Arbitrarily positioned text
title	Plot title
trace	Trace of a matrix
who	Lists all variables currently in memory
xlabel	x-Axis label
ylabel	y-Axis label
zeros	Zero