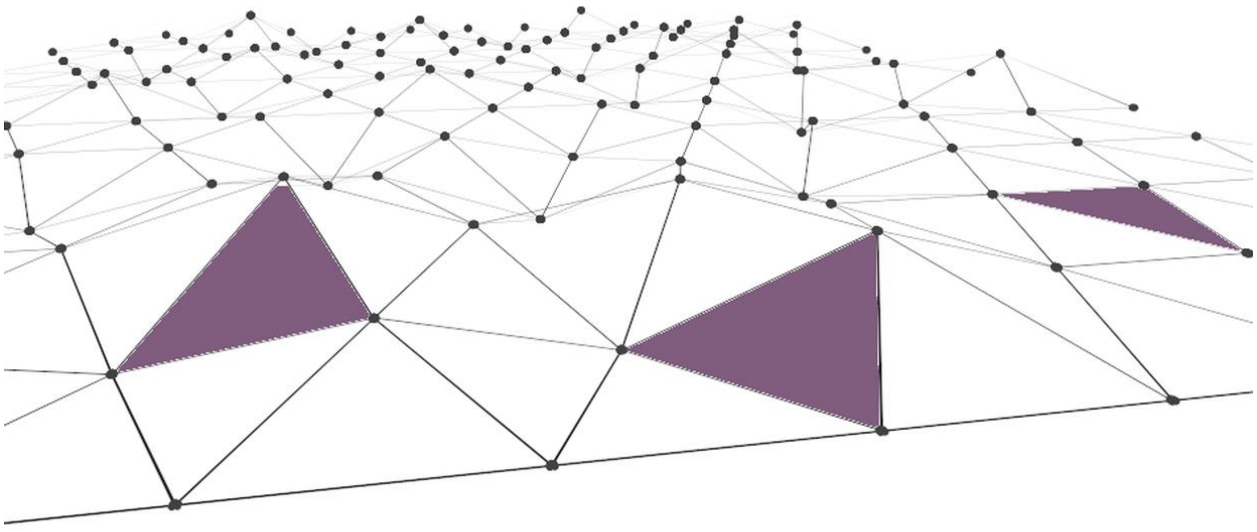
Lecture Notes in Computational Mathematics II (MATLAB) for Second Class of Mathematics



Imad A. Aziz

Erbil

2022-2023

Chapter 1
MATLAB Graphics

Graphical representation of data is an attractive method of showcasing numerical data that help in analyzing and representing quantitative data visually. A graph is a kind of a chart where data are plotted as variables across the coordinate. It became easy to analyze the extent of change of one variable based on the change of other variables. Graphical representation of data is done through different mediums such as lines, plots, diagrams, etc. Let us learn more about this interesting concept of graphical representation of data, the different types, and solve a few examples.
MATLAB has many commands that can be used to create basic 2-D plots, overlay plots, specialized 2-D plots, 3-D plots, mesh, and surface plots.

1.1 Basic 2-D Plots

1.1.1 Graphing with plot

The plot command works on vectors of numerical data. The syntax is

```
plot(x values, y values,‘style option’)
```

where **x** values and **y** values are vectors containing the x- and y-coordinates of points on the graph. style option is an optional argument that specifies the color, line-style, and the point marker style. The style option in the plot command is a character string that consists of 1, 2, or 3 characters that specify the color and/or the line style. The different color, line-style and marker style options are summarized in Tables

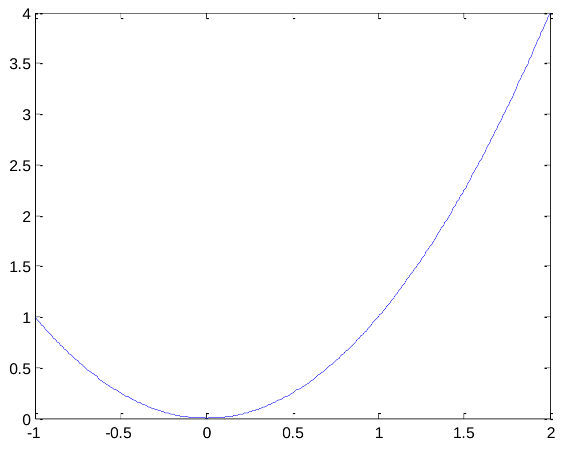
Line Style	**Description**	**Resulting Line**
`"-"`	Solid line	________
`"--"`	Dashed line	- — — — —
`":"`	Dotted line	………………
`"-."`	Dash-dotted line	_._._._.-

Marker	Description	Resulting Marker
"o"	Circle	○
"+"	Plus sign	+
"*"	Asterisk	✳
"."	Point	•
"x"	Cross	×
"_"	Horizontal line	—
"|"	Vertical line	|
"square"	Square	□
"diamond"	Diamond	◇
"^"	Upward-pointing triangle	△
"v"	Downward-pointing triangle	▽
">"	Right-pointing triangle	▷
"<"	Left-pointing triangle	◁
"pentagram"	Pentagram	☆
"hexagram"	Hexagram	✡

Color Name	Short Name	RGB Triplet
"red"	"r"	[1 0 0]
"green"	"g"	[0 1 0]
"blue"	"b"	[0 0 1]
"cyan"	"c"	[0 1 1]
"magenta"	"m"	[1 0 1]
"yellow"	"y"	[1 1 0]
"black"	"k"	[0 0 0]
"white"	"w"	[1 1 1]

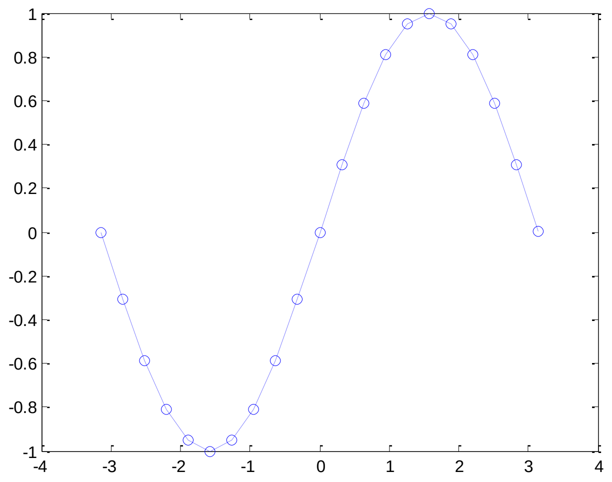
Example 1.1: - To plot $y=x^2$ on the interval from -1 to 2

```
 X = -1:0.01:2;
plot(X, X.^2)
```



Example 1.2: - To plot $y=sin\ x$ on the -interval from $-\pi$ to π

```
x=-pi:pi/10:pi;
y=sin(x);
plot(x,y,':bo')
```



1.1.2 Adding various labels or making adjustments to plots

1- axis:- Set axis ranges in a figure window

`axis([xmin xmax ymin ymax])`

sets the limits for the *x*- and *y*-axis of the current axes or can use

`axis('auto')`: sets MATLAB default behavior to compute the current axes limits automatically, based on the minimum and maximum values of *x*, *y*.

`axis('square')`: makes the current axes region square.

2- title:- Add title to current axes.

```
 title('string') or title(fname)
```

3- xlabel:- Add x-axis labels to plot and ylabel:- Add y-axis labels to plot.

```
xlabel('somestring')
ylabel('somestring')
```

4- grid on:- Add grid lines to plot and grid off to turn off.

5- legend:- Add figure legend to top-left corner of plot.

```
legend('first', 'second', 'Location')
```

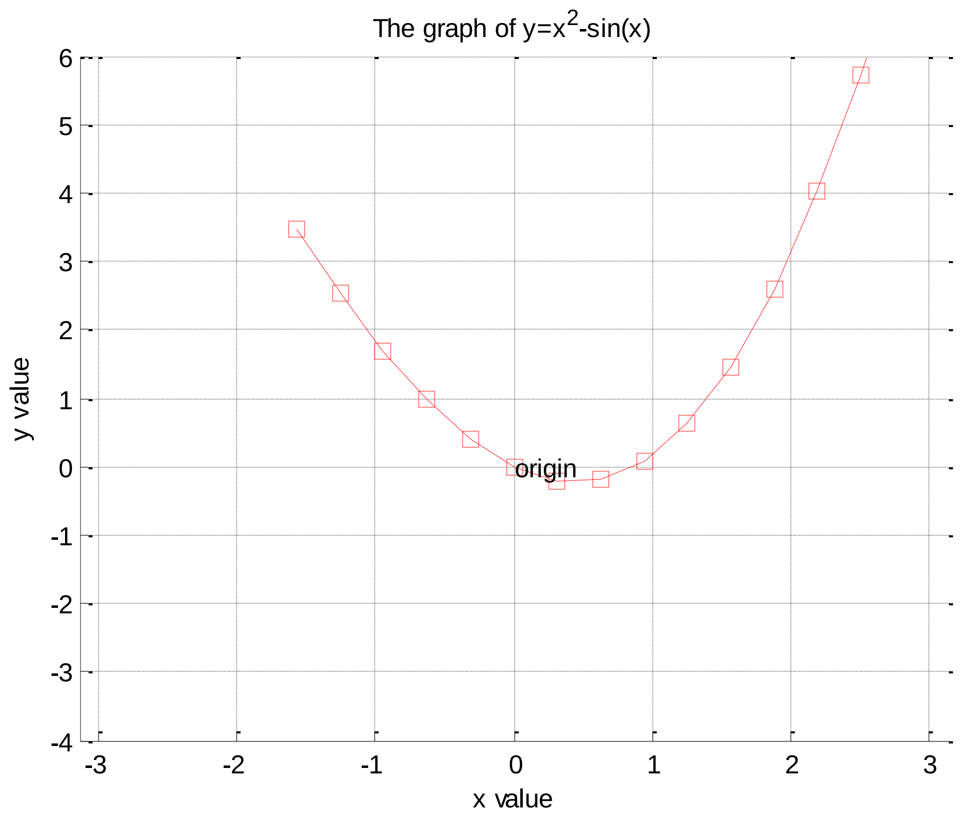
6- text:-Create text object in current axes.

```
text(x,y,'string')
```

7- hold on:- Adding more things to a figure hold on means everything plotted from now on in that figure window is added to what's already there. Hold off turns it off.

Example 1.3: - draw the graph of $y = x^2 - sinx$ from $-\pi/2\ to\ \pi$.

```
hold on
title('The graph of y=x^2-sin(x)')
axis([-pi pi -4 6])
x=-pi/2:pi/10:pi;
y=x.^2-sin(x);
grid on
xlabel('x value')
ylabel('y value')
plot(x,y,'-rs')
text(0,0,'origin')
```

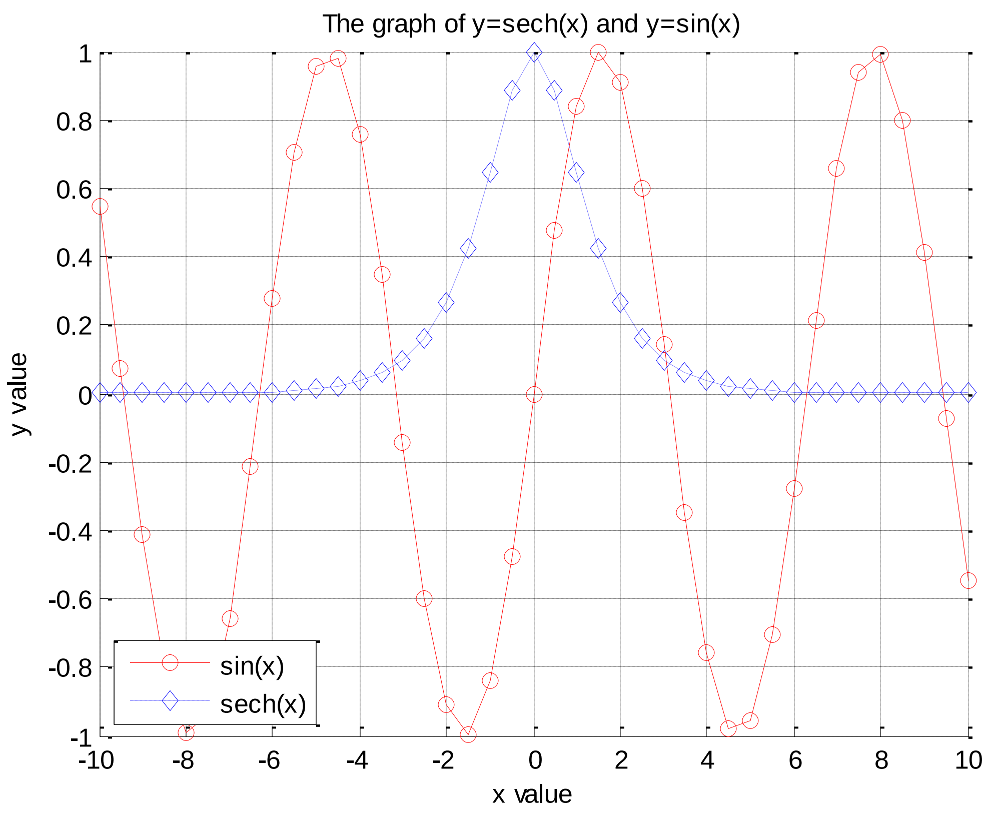


1.1.3 Plotting Multiple Curves

Each time you execute a plotting command, MATLAB erases the old plot and draws a new one. If you want to overlay two or more plots, type hold on. This command instructs MATLAB to retain the old graphics and draw any new graphics on top of the old. It remains in effect until you type hold off.

Example 1.4: - draw the graphs of $y = \operatorname{sech}(x)$ and $y = \sin(x)$ from -10 to 10.

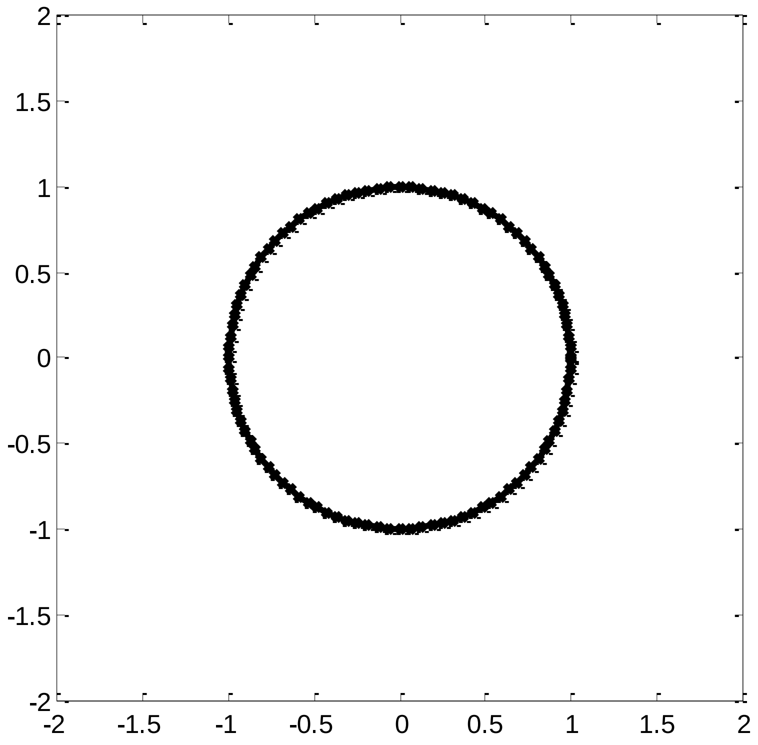
```
hold on
title('The graph of y=sech(x) and y=sin(x)')
axis('normal')
x=-10:0.5:10;
y1=sin(x);
y2=sech(x);
grid on
xlabel('x value')
ylabel('y value')
plot(x,y1,'-ro')
plot(x,y2,':bd')
legend('sin(x)','sech(x)',3)
```



1.1.4 Parametric Plots

Sometimes *x* and *y* are both given as functions of some parameter. For example, the circle of radius 1 centered at (0,0) can be expressed in *parametric* form as $x = \cos(2\pi t)$, $y = \sin(2\pi t)$ where *t* runs from 0 to 1. Though *y* is not expressed as a function of *x*, you can easily graph this curve with **plot**, as follows:

```
T = 0:0.01:1;

x=cos(2*pi*T);

y=sin(2*pi*T);

plot(x,y)

axis square

axis([-2 2 -2 2])
```



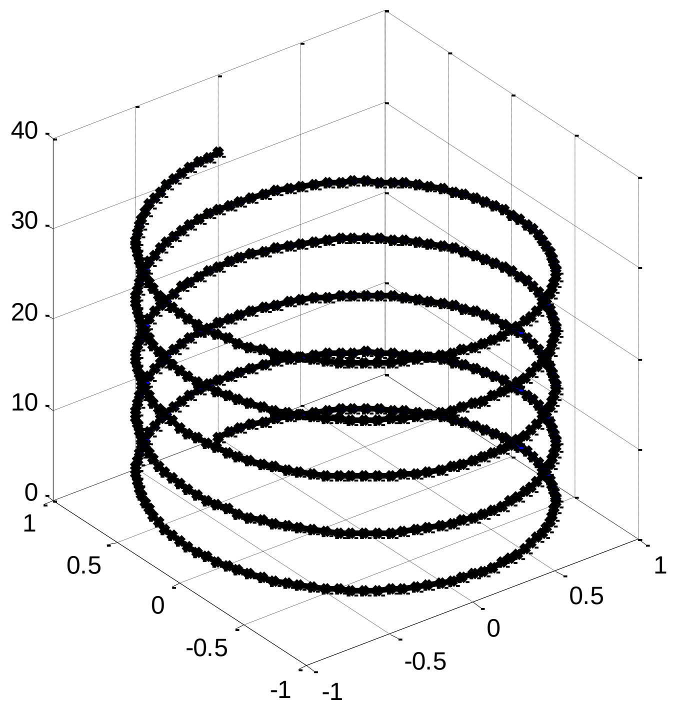
1.3 Three-Dimensional Plots

MATLAB has several routines for producing three-dimensional plots.

1.3.1 Curves in Three-Dimensional Space

For plotting curves in 3-space, the basic command is plot3. It works like plot, except that it takes three vectors instead of two, one for the x-coordinates, one for the y-coordinates, and one for the z-coordinates. For example, we can plot a helix with

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



1.3.2 Surfaces in Three-Dimensional Space

There are two basic commands for plotting surfaces in 3-space: mesh and surf. The former produces a transparent “mesh” surface; the latter produces an opaque shaded one. There are two different ways of using each command, one for plotting surfaces in which the z coordinate is given as a function of x and y, and one for parametric surfaces in which x, y, and z are all given as functions of two other parameters. Let us illustrate the former with mesh and the latter with surf.
To draw the surface of a function $y = f(x, y)$ needs to discretization the region.

meshgrid :-Generate X and Y arrays for 3-D plots

```
[X,Y] = meshgrid(x,y)
```

transforms the domain specified by vectors x and y into arrays X and Y, which can be used to evaluate functions of two variables and three-dimensional mesh/surface plots. The rows of the output array X are copies of the vector x; columns of the output array Y are copies of the vector y.

`[X,Y] = meshgrid(x)` is the same as `[X,Y] = meshgrid(x,x)`.

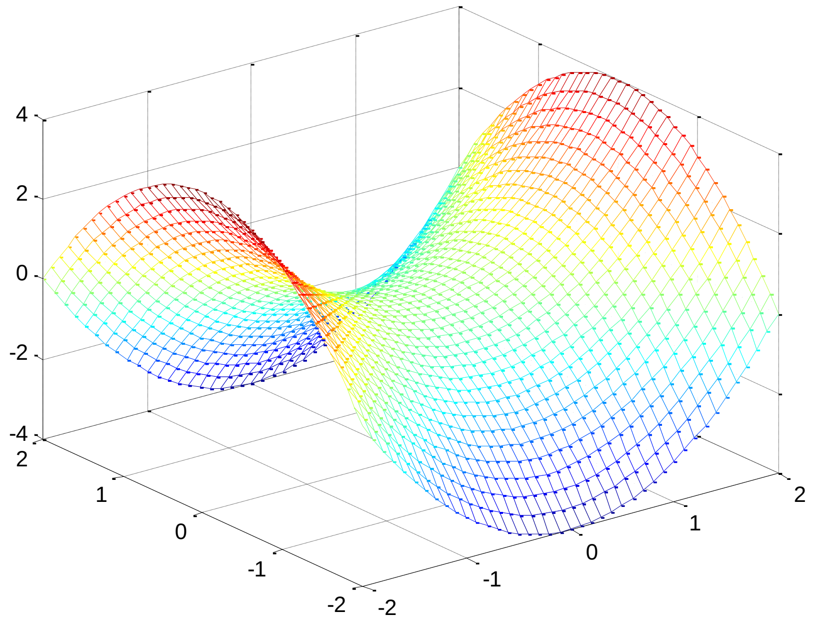
```
>> [x y]=meshgrid(1:4,5:7)

x =
     1     2     3     4
     1     2     3     4
     1     2     3     4

y =
     5     5     5     5
     6     6     6     6
     7     7     7     7
```

`mesh(X,Y,Z)`

If X and Y are rectangular arrays containing the values of the x and y coordinates at each point of a rectangular grid, and if z is the value of a function evaluated at each of these points, **mesh(X, Y, Z)** will produce a three-dimensional perspective graph of the points. The same results can be obtained with **mesh(x, y, z)** can also be used.

Example 1.5:- Draw the "saddle surface" $z = x^2 - y^2$ on region $-2 \le x,y \le 2$

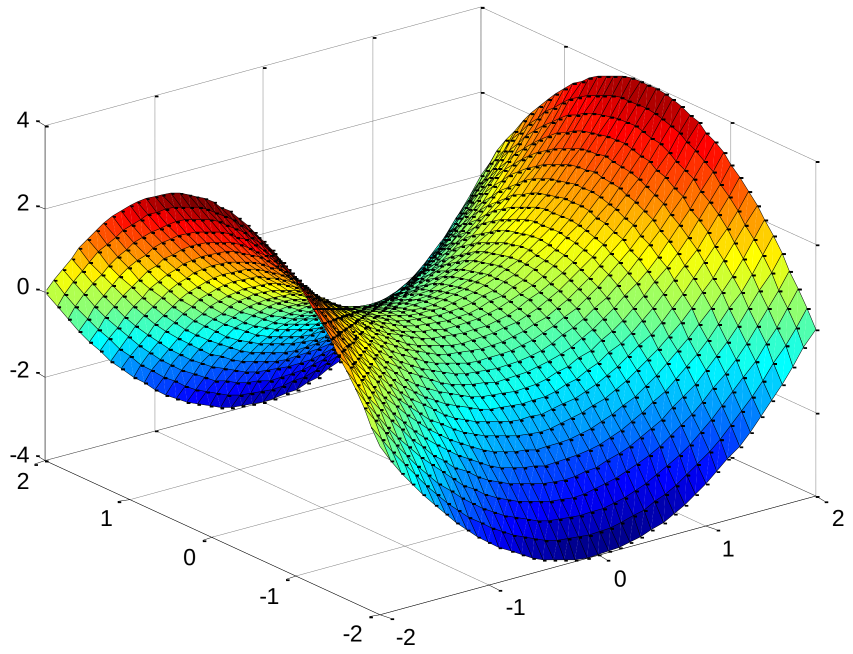
```
 [X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);
 Z = X.^2 - Y.^2;
 mesh(X, Y, Z)
```



`surf(X,Y,Z)`

Produces a three-dimensional perspective drawing. Its use is usually o draw surfaces, as opposed to plotting functions, although the actual tasks are quite similar. The output of surf will be a shaded figure.

```
[X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);

Z = X.^2 - Y.^2;

surf(X, Y, Z)
```



Exercises

1- Write a program to draw the graph of $f(x) = \begin{cases} x^2+1 & if \;\; x \geq 2 \\ 1 \;\; if & 0 \leq x < 2 \\ \;\; |x-3| & x < 0 \end{cases}$.

2- Write a program to draw the graph of $f(x) = \ln(x^2-4)$ on interval [-10, 10].

3- Write a program to draw the graph of $f(x) = \sqrt{x-2} + \sqrt{1-x^2}$ on interval [-10, 10].

4- Write a program to input a polynomial then draw its graph and plot its max. and min. points if exist on his graph.

5- Write a program to draw the surface of $f(x,y) = \ln(\mathrm{x}^2 - \mathrm{y}^2) + \sqrt{\mathrm{xy}-1}$ on region $C = \{(x,y) \in \mathbb{R}^2;\, -2 \leq x \leq 2 \; and -1 \leq y \leq 1\}$

6- Write a program to draw the surface of $f(x,y) = \mathrm{y}\mathrm{e}^{\mathrm{x}^2+\mathrm{y}^2}$ on region $C = \{(x,y) \in \mathbb{R}^2;\ x^2 + y^2 \leq 4\}$

7- Write a program to draw the surface of $f(x,y) = \mathrm{x}\sin(\mathrm{y}) + \mathrm{y}\cos(\mathrm{x})$ on region $C = \{(x,y) \in \mathbb{R}^2; 1 \leq x^2 + y^2 \leq 4\}$

8- Write a program to draw the surface of $f(x,y) = \mathrm{x}^2 + \mathrm{y}^2 - 6$ on region $C = \{(x,y) \in \mathbb{R}^2; x \in [-4,-2\,] \cup [\,2\,,4\,]\ and\ y \in [-4,-2] \cup [\,2\,,4\,]\,\}$

9- Write a program to draw the surface of $f(x,y) = \mathrm{x}^2 + \mathrm{y}^2 - 6$ on region $C = \{(x,y) \in \mathbb{R}^2; (x,y) \in \Delta ABC : A = (1\,,1), B = (6\,,2\,), C = (3\,,\ 5)\,\}$

10-Write a program to draw the surface of $f(x,y) = \mathrm{x}^2 - \mathrm{y}^2$ on region $C = \{(x,y) \in \mathbb{R}^2;\ -2 \leq x \leq 2\ and - 1 \leq y \leq 1\ and\ x^2 + y^2 - xy \geq 1\}$

Chapter 2
Polynomials

A *polynomial* is a function of a single variable that can be expressed in the following form:

$$f(x) = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + \ldots + a_{n-1}x_1 + a_n$$

where the variable is *x* and the coefficients of the polynomial are represented by the values a_0, a_1, … and so on. The *degree* of a polynomial is equal to the largest value used as an exponent.

2.1 Input Polynomial

A vector represents a polynomial in MATLAB. When entering the data in MATLAB, simply enter each coefficient of the polynomial into the vector in descending order.

Example 2.1:- consider the polynomial $p(s) = 5s^5 + 7s^4 + 2s^2 - 6s + 10$

To enter this into MATLAB, we enter this as a vector as

```
>> p = [5 7 0 2 - 6 10]
p =
     5 7 0 2 - 6 10
```

It is necessary to enter the coefficients of all the terms.

2.2 Polynomial multiplication and division

MATLAB contains functions that perform polynomial multiplication and division, which are listed below:

conv(*p*, *q*)

Let $p(x)$ and $q(x)$ be two polynomial of degree n and m respectively, to find $h(x) = p(x) * q(x)$ using **h=conv(*p*, *q*)** computes a coefficient vector that contains the coefficients of the product of polynomials represented by the coefficients in *p* and *q*. The vectors *p* and *q* do not have to be the same size.

Example 2.2:- consider the polynomials

$p(x) = x^4 - 4x^2 + 2x + 1$ and $q(x) = x + 2$
then to find $h(x) = p(x) * q(x)$ write Matlab code as follows

```
p=[1 0 -4 2 1];
q=[1 2];
h=conv(p,q)
```

result is

```
h =
     1     2    -4    -6     5     2
```

or
$h(x) = x^5 + 2x^4 - 4x^3 - 6x^2 + 5x + 1$

deconv(*p*, *q*)

Let $p(x)$ and $q(x)$ be two polynomial of degree n and m respectively, to find
$h(x) = p(x)/q(x)$ using **[*h*, *r*]=deconv(*p*, *q*)**
Returns two vectors. The first vector contains the coefficients of the quotient and the second vector contains the coefficients of the remainder polynomial.

Example 2.3: - consider the polynomials
$p(x) = x^4 - 4x^2 + 2x + 1$ and $q(x) = x^2 + 2$
then to find $h(x) = p(x)/q(x)$ write Matlab code as follows

```
p=[1 0 -4 2 1];
q=[1 0 2];
[h,r]=deconv(p,q)
```

result is

```
h =
     1     0    -6
r =
     0     0     0     2    13
```

or
$h(x) = x^2 - 6$ and $r(x) = 2x + 13$

2.3 Polynomial Algebraic

roots(p)

Let $p(x)$ is a polynomial the MATLAB function for determining the roots of a polynomial is the **roots** function. **r=roots(p)** determines the roots of the polynomial represented by the coefficient vector *p*. The roots function returns a column vector containing the roots of the polynomial, the number of roots is equal to the degree of the polynomial.

Example 2.4: - consider the polynomial $p(x) = x^4 - 4x^2 + 2x + 1$ then to find the roots of *p* write Matlab code as follows

```
p=[1 0 -4 2 1];
r=roots(p)
```

result is

```
r =
   -2.1701
    1.4812
    1.0000
   -0.3111
```

are four real roots of polynomial *p*.

poly(r)

When the roots of a polynomial are known, the coefficients of the polynomial are determined when all the linear terms are multiplied, we can use the poly function *p*=**poly(*r*)** Determines the coefficients of the polynomial whose roots are contained in the vector *r*. The output of the function is a row vector containing the polynomial coefficients.

Example 2.5: - consider the polynomial $r = [1\ 2\ \ 0\ -4]$,then to find the $p(x)$ with roots *r*, write Matlab code as follows

```
r=[1 2 0 -4];
p=poly(r)
```

result is

```
p =
     1     1   -10     8     0
```
or
$p(x) = x^4 + x^3 - 10x^2 + 8x$

polyval (*p*, *x*)

The value of a polynomial can be computed using the *polyval* function
y=**polyval(***p*, *x***)** it evaluates a polynomial with coefficients *p* for the values in *x*. The result is a matrix the same size of *x*.

Example 2.6: - consider the polynomial $p(x) = x^4 - 4x^2 + 2x + 1$ and *x*=2, then to find the $p(2)$ write Matlab code as follows

```
p=[1 0 -4 2 1];
x=2;
y=polyval(p,x)
```

result is
```
y =
     5
```

2.4 Polynomial Calculus

polyder(p)

Let $p(x)$ is a polynomial the MATLAB function for determining the derivative of $p(x)$ is *q*=**polyder(***p***)** returns the derivative of the polynomial represented by the coefficients in *p*,

$$q(x) = \frac{d}{dx}p(x).$$

h = **polyder(***p*,*q***)** returns the derivative of the product of the polynomials *p* and *q*,

$$h(x) = \frac{d}{dx}[p(x) * q(x)].$$

[*h*,*r***] = polyder(***p*,*q***)** returns the derivative of the quotient of the polynomials *p* and *q*,

$$\frac{h(x)}{r(x)} = \frac{d}{dx}\frac{p(x)}{q(x)}$$

Example 2.7: - consider the polynomial $p(x) = x^4 - 4x^2 + 2x + 1$, then to find the derivative of *p* write Matlab code as follows

```
p=[1 0 -4 2 1];
q=polyder(p)
```

result is

```
q =
     4     0    -8     2
```

or

$q(x) = 4x^3 - 8x + 2$

polyint

Let $p(x)$ is a polynomial the MATLAB function for determining the integration of $p(x)$ is *q* = **polyint**(*p*,*k*) returns the integral of the polynomial represented by the coefficients in *p* using a constant of integration *k*. *q* = **polyint**(*p*) assumes a constant of integration *k* = 0.

Example 2.8: - consider the polynomial $p(x) = 4x^3 - 3x^2 + 8x + 1$, then to find the integration of *p* write Matlab code as follows

```
p=[4 -3 8 1];
q=polyint(p)
```

result is

```
q =
     1    -1     4     1     0
```

or

$q(x) = x^4 - x^3 - 4x^2 + x$

Exercises

1- Write a program to find the tangent line of polynomial $p(x)$ at the point x_0 then draw the graph of polynomial ant its tangent line.
2- Write a program to find all local maximum and minimum pints of polynomial $p(x)$ then draw polynomial and (max., min.) points.
3- Write a program to find the area under polynomial $p(x)$ if its bounded.

4- Write a program to find the area under polynomial $p(x)$ on interval $[a, b]$.
5- Write a function to add two polynomial $p(x)$ and $q(x)$.
6- Write a function to subtract two polynomial $p(x)$ and $q(x)$.
7- Write a program to find the area between two polynomials $p(x)$ and $q(x)$ if its bounded.

Chapter 3
Symbolic Processing

We have focused on the use of Matlab to perform numerical operations, involving numerical data represented by double precision floating point numbers. We also given some consideration to the manipulation of text, represented by character strings. In this section, we introduce the use of Matlab to perform *symbolic processing* to manipulate mathematical expressions, in much the way that we do with pencil and paper.
The objective of symbolic processing is to obtain what are known as *closed form* solutions, expressions that don't need to be iterated or updated in order to obtain answers. An understanding of these solutions often provides better physical and mathematical insight into the problem under investigation.

2.1 Declaring Symbolic Variables and Constants

To enable symbolic processing, the variables and constants involved must first be declared as *symbolic objects*.

`Syms` *`var1,var2,`*……

For example, to create the symbolic variables with names x and y:

```
>> syms x y
```

If x and y are to be assumed to be real variables, they are created with the command:

```
>> syms x y real
```

`Sym(var)`

To declare symbolic constants, the sym function is used. Its argument is a string containing the name of a special variable, a numeric expression, or a function evaluation. It is used in an assignment statement which serves as a declaration of a symbolic variable for the assigned variable. Examples include:

```
>> pi = sym(’pi’);

>> delta = sym(’1/10’);

>> sqroot2 = sym(’sqrt(2)’);
```

If the symbolic constant pi is created this way, it replaces the special variable **pi** in the workspace. The advantage of using symbolic constants is that they maintain full accuracy until a numeric evaluation is required.

Example 2.1: -

```
 >> x=sym(1/2)          >> x*y             >> x/y
 x = 1/2                ans = 1/3          ans = 3/4
 >> y=sym(2/3)          >> x+y             >> x^2
 y = 2/3                ans = 7/6          ans = 1/4
```

Symbolic variables and constants are represented by the data type symbolic object.

```
>> whos
Name        Size       Bytes        Class
delta       1x1         132       sym object
pi          1x1         128       sym object
sqroot2     1x1         138       sym object
x           1x1         126       sym object
```

2.2 Symbolic Expressions

Symbolic variables can be used in expressions and as arguments of functions in much the same way as numeric variables have been used. The operators `+ - * / ^` and the built-in functions can also be used in the same way as they have been used in numeric calculations.

Example 2.2:-

```
>> syms x y
f = x^2*y + 5*x*sqrt(y)
f =
 x^2*y + 5*x*y^(1/2)

syms s t A
g = s^2 + 4*s + A
g =
 s^2 + 4*s + A

 >> h=f*g
 h =
 (x^2*y + 5*x*y^(1/2))*(s^2 + 4*s + A)
```

The variable *x* is the default independent variable, but as can be seen with the expressions above, other variables can be specified to be the independent variable. It is important to know which variable is the independent variable in an expression. The command to find the independent variable is:

symvar (S)

Finds the symbolic variables in a symbolic expression or matrix S by returning a string containing all of the symbolic variables appearing in S. The variables are returned in alphabetical order and are separated by commas. If no symbolic variables are found, **symvar** returns the empty string.
Use example 2.2

```
>> symvar(f)
ans =
     [x,y]
>> symvar(z)
ans =
     [A,s,t]
```

2.3 Manipulating Polynomial Expressions

In the examples above, symbolic variables were declared and were used in symbolic expressions to create polynomials. We now wish to manipulate these polynomial expressions algebraically.
The Matlab commands for this purpose include:

expand(S)

Expands each element of a symbolic expression S as a product of its factors. **expand** is most often used on polynomials, but also expands trigonometric, exponential and logarithmic functions.

Example 2.3:-

```
syms x;
expand((x-2)*(x-4))
ans = x^2 - 6*x + 8

syms x y;
expand(cos(x+y))
ans = cos(x)*cos(y) - sin(x)*sin(y)

syms a b;
expand(exp((a+b)^2))
ans = exp(2*a*b)*exp(a^2)*exp(b^2)

syms t;
expand([sin(2*t), cos(2*t)])
ans =[ 2*cos(t)*sin(t), cos(t)^2 - sin(t)^2]
```

factor(S)

Factors each element of the symbolic matrix S.

Example 2.4:-

```
syms x y;
factor(x^3-y^3)
ans = (x - y)*(x^2 + x*y + y^2)
```

```
syms a b;
factor([a^2 - b^2, a^3 + b^3])
ans = [ (a - b)*(a + b), (a + b)*(a^2 - a*b + b^2)]
```

```
x=sym(5678)
x = 5678
factor(x)
ans = 2*17*167
```

simplify(S)

Simplifies each element of the symbolic matrix S.

Example 2.5:-

```
syms s
H = (s^3 +2*s^2 +5*s +10)/(s^2 + 5);
H = simplify(H)
H = s+2
```

```
syms x;
simplify(sin(x)^2 + cos(x)^2)
ans =1
```

```
syms a b c;
simplify(exp(c*log(sqrt(a+b))))
ans =(a + b)^(c/2)
```

```
S = [(x^2 + 5*x + 6)/(x + 2), sqrt(16)];
R = simplify(S)
R =[ x + 3, 4]
```

[n , d] = numden(S)

Returns two symbolic expressions that represent the numerator expression num and the denominator expression den for the rational representation of the symbolic expression S.

Example 2.6:-

```
[n, d] = numden(sym(4/5))
ans
n =4
d =5
```

```
syms x y;
[n,d] = numden(x/y + y/x)
ans
n = x^2 + y^2
d = x*y
```

```
 syms s
G = s+4 + 2/(s+4) + 3/(s+2);
[N, D] = numden(G)
N = s^3+10*s^2+37*s+48
D =(s+4)*(s+2)
```

collect(f)

views f as a polynomial in its symbolic variable, say x, and collects all the coefficients with the same power of x. A second argument can specify the variable in which to collect terms if there is more than one candidate.

Example 2.7:-

```
f=(x-1)*(x-2)*(x-3)
collect(f)
ans= x^3-6*x^2+11*x-6
```

```
f=x*(x*(x-6)+11)-6
collect(f)
ans= x^3-6*x^2+11*x-6
```

```
f=(1+x)*t + x*t
collect(f)
ans=2*x*t+t
```

subs(S,old,new)

Symbolic substitution, replacing symbolic variable old with symbolic variable new in the symbolic expression S.

Example 2.8:-

```
f = 2*x^2 - 3*x + 1
subs(f,2)
ans =3
```

```
syms x y
f = x^2*y + 5*x*sqrt(y)
subs(f, x, 3)
ans = 9*y+15*y^(1/2)
```

```
syms s
H = (s+3)/(s^2 +6*s + 8);
G = subs(H,s,s+2)
G = (s+5)/((s+2)^2+6*s+20)
```

```
E = s^3 -14*s^2 +65*s -100;
F = subs(E,s,7.1)
F =13671/1000
```

2.4 Some necessary subjects

2.4.1 Function handles

A function handle (@) is a reference to a function that can then be treated as a variable. It can be copied, placed in cell array, and evaluated just like a regular function.
Function handles can refer to built-in MATLAB functions, to your own function in an M-file, or to anonymous functions. An anonymous function is defined with a one-line expression, rather than by an M-file. Try:

```
g = @(x) x^2-5*x+6-sin(9*x)
g(1)=1.5879
```

Some MATLAB functions that operate on function handles need to evaluate the function on a vector, so it is often better to define an anonymous function (or M-file)so that it can operate entry-wise on scalars, vectors, or
matrices. Try this instead:

```
g = @(x) x.^2-5*x+6-sin(9*x)
g([-1 0 2 3])
ans =
   12.4121    6.0000    0.7510   -0.9564
```

The general syntax for an anonymous function is

```
fname = @(var1, var2, ...) expression
```

Here is an example with two input arguments:

```
norm2 = @(x,y) sqrt(x^2 + y^2)
norm2(4, 5)
```

Example 2.9: - Write a program to draw the surface of input function on square region around origin point.

```
f=input('input handle f(x,y)=');
[x1,y1]=meshgrid(-3:0.1:3);
z1=f(x1,y1);
surf(x1,y1,z1)
```

should input the function as `@(x,y)x.^2+y.^2`

2.4.2 Cell arrays

Cell arrays are arrays which contain elements of arbitrary types. They are identified by curly braces instead of square ones:

```
>> c = {[3,4],18.2,[1,2;2,1],'string'};
```

defines a cell array c. We can access elements of c in the following way:

```
>> c{3}
ans =
     1  2
     2  1
```

Example 2.10: - You can substitute multiple symbolic expressions, numeric expressions, or any combination, using cell arrays of symbolic or numeric values. Try:

```
syms x y
S = x^y
subs(S, x, 3)
subs(S, {x y}, {3 2})
subs(S, {x y}, {3 x+1})
```

Example 2.11: - Let

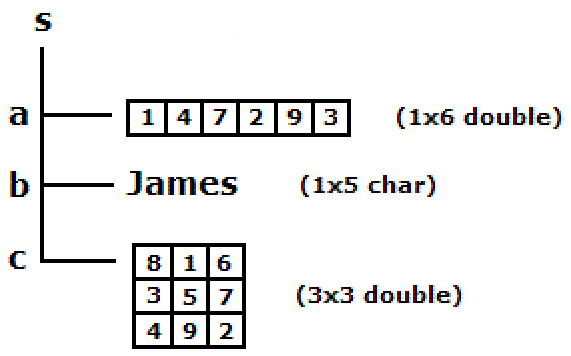
```
D = {‘red’;‘blue’;‘green’;‘yellow’}
```

```
 D =                                   sort(D)
 ‘red’                                 ans =
 ‘blue’                                    'blue'
 ‘green’                                   'green'
 ‘yellow’                                  'red'
                                           'yellow'
```

2.4.1 Structure

A structure is a MATLAB data type that provides the means to store hierarchical data together in a single entity. A structure consists mainly of data containers, called fields, and each of these fields stores an array of some MATLAB data type. You assign a name to each field as you create the structure. The figure below shows a structure s that has three fields: a, b, and c.

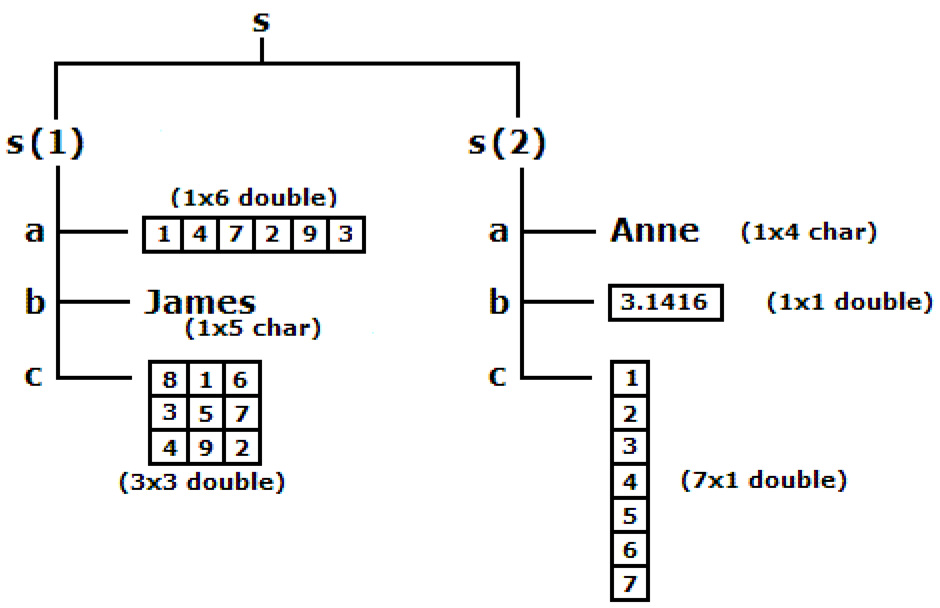
Example 2.12: -



```
s.a=[1 4 7 2 9 3];
s.b='James'
s.c=[8 1 6;3 5 7;4 9 2];
```

Like all MATLAB data types, the structure is an array. The class of a structure is called struct, so an array of structures is often referred to as a struct array. Like other MATLAB arrays, a struct array can have any dimensions. The struct array shown below has the dimensions 1-by-2 and is composed of two elements: s(1) and s(2). Each of these elements is a structure with fields a, b, and c of its own.

Example 2.13: -



```
s(1).a=[1 4 7 2 9 3];
s(1).b='James'
s(1).c=[8 1 6;3 5 7;4 9 2];
s(2).a='Anne';
s(2).b=pi;
s(2).c=[1;2;3;4;5;6;7];
```

2.5 Polynomial and Solving Equations

Polynomials divisor function

[q, r] = quorem(p, h) Divided p by h and return quotient q and remainder r.

Example 2.14: -

```
syms x
p=x^3-4*x^2+3*x+1;
h=x^2+1;
[q, r] = quorem(p, h)
q = x - 4
r =2*x + 5
```

Coefficients of polynomial
C = **coeffs(p)** returns the coefficients of the polynomial p with respect to all the in determinates of p.
C = **coeffs(p, x)** returns the coefficients of the polynomial p with respect to x.

Example 2.15: -

```
syms x
 f=2*x^3-6*x+2;
coeffs(f)
ans =[ 2, -6, 2]
```

Example 2.16: -

```
syms x y
z = 3*x^2*y^2 + 5*x*y^3;
coeffs(z)
coeffs(z,x)
ans =[ 5, 3]
ans =[ 5*y^3, 3*y^2]
```

Conversions

sym2poly(P):- Converts from a symbolic polynomial P to a row vector containing the polynomial coefficients.

poly2sym(p) :-Converts from a polynomial coefficient vector p to a symbolic polynomial in the variable x. poly2sym(p,v) uses the symbolic the variable v.

Example 2.17: - consider the polynomial $A(s) = s^3 + 4s^2 - 7s - 10$
In Matlab:

```
a = [1 4 -7 -10];
A = poly2sym(a,s)
A = s^3+4*s^2-7*s-10
```

Example 2.18: - For the polynomial $B(s) = 4s^3 - 2s^2 + 5s - 16$

```
syms s
B = 4*s^3 -2*s^2 +5*s -16;
b = sym2poly(B)
b = 4 -2 5 -16
```

double

The function double(c) converts the symbolic object c (constants, scalar, or matrix) into a double precision floating point variable.

Example 2.19: -

```
 a=sym(2/3)
 b=sym(1/5);
 a+b=13/15
 double(a+b)= 0.8667
```

Solving Equations

You can solve equations involving variables with **solve**.

Example 2.20: - to find the solutions of the quadratic equation $x^2 - 2x - 4 = 0$, type

```
solve('xˆ2 - 2*x - 4 = 0')
ans =
[ 5^(1/2)+1]
[ 1-5^(1/2)]
```

Or

```
syms x
f=x^2-2*x-4;
solve(f)
```

```
ans =
 1 - 5^(1/2)
 5^(1/2) + 1
```

Note that the equation to be solved is specified as a string; that is, it is surrounded by single quotes. The answer consists of the exact (symbolic) solutions $1 \pm\sqrt{5}$. To get numerical solutions, type **double(ans)**, or **vpa(ans)** to display more digits.

```
double(solve(f))
ans =
   -1.2361
    3.2361
```

```
Or
```

```
vpa(solve(f))
ans =
 -1.2360679774997896964091736687313
  3.2360679774997896964091736687313
```

The command **solve** can solve higher-degree polynomial equations, as well as many other types of equations. It can also solve equations involving more than one variable. If there are fewer equations than variables, you should specify (as strings) which variable(s) to solve for.

Example 2.21: - to solve $2x - \log y = 1$ for y in terms of x.

```
syms x y
solve(2*x - log(y) ==1, x)
ans =
log(y)/2 + 1/2
```

You can specify more than one equation as well.

Example 2.22: -

```
syms x y
[xsol, ysol] = solve(x^2 - y == 2, y - 2*x ==5,[x,y])
```

Exercises

Q1) Write a program to draw the graph of input function.

Q2) Write a program to draw the surface of input function.

Q3) Write a program to check if the input function is odd or even.

Q4) Write a program to find all asymptotic line of input function if exist.

Q5) Write a program to find the area between three line if they are not co-linear and then draw the triangle of intersection.

Q6) Write a program to find out if the input function is onto one or not then find it inverse if exist the draw the graphs.

Q7) Write a program to find all singular point of input equation.

Q8) Write a program to find the equation of line from two points.

Q9) Mixing MATLAB built-in functions to do the following by one-line statement in command window

a) Find the number of imaginary roots of polynomial.
b) Find the sum of real roots of polynomial.
c) Find the maximum coefficient in polynomial.
d) Find the degree of polynomial.

Chapter 4
Calculus Applications

The Symbolic Math Toolbox provides functions to do the basic operations of calculus; differentiation, limits, integration, summation, and Taylor series expansion.

4.1 Limits

limit:-Compute limit of symbolic expression

Syntax

```
limit(expr, x, a)
limit(expr, a)
limit(expr)
limit(expr, x, a, 'left')
limit(expr, x, a, 'right')
```

Description

`limit(expr,x,a)`:- computes bidirectional limit of the symbolic expression `expr` when `x` approaches *a*.

`limit(expr,a)`:- computes bidirectional limit of the symbolic expression `expr` when the default variable approaches *a*.

`limit(expr)`:- computes bidirectional limit of the symbolic expression `expr` when the default variable approaches `0`.

`limit(expr,x,a,'left')`:-computes the limit of the symbolic expression `expr` when `x` approaches *a* from the left.

`limit(expr,x,a,'right')`:- computes the limit of the symbolic expression `expr` when `x` approaches *a* from the right.

Example 4.1:- Find the following limits

$$1)\ \lim_{x\to 2}\frac{x^2-4}{x-2} \qquad 2)\ \lim_{y\to 0}(xy+x) \qquad 3)\ \lim_{x\to 0^+}\frac{1}{x} \qquad 4)\ \lim_{t\to\infty}\frac{1-t^2}{3t^2+t} \qquad 5)\ \lim_{x\to 0^-}\frac{|x|}{x}$$

1)
```
syms x y
limit((x^2-4)/(x-2),2)
ans =4
```

2)
```
limit(x*y+x,y,0)
  ans = x

if we write
```

```
or
limit((x^2-4)/(x-2),x,2)
 ans =4
```

```
limit(x*y+x,0)
ans = 0
```

```
3) limit(1/x,x,0,'right')
   ans = Inf
```

```
4) syms t
 limit((1-
t^2)/(3*t^2+t),t,inf)
 ans =-1/3
```

```
5) limit(abs(x)/x,x,0,'left')
   ans = -1
```

4.2 Differentiation

diff:- Differentiate symbolic expression

Syntax

```
diff(expr)
diff(expr, v)
diff(expr, n)
diff(expr, v, n)
```

Description

`diff(expr)` :- differentiates a symbolic expression `expr` with respect to its free variable as determined by `symvar`.

`diff(expr, v)` :- differentiate `expr` with respect to `v`.

`diff(expr, n)` :- differentiates `expr` *n* times. *n* is a positive integer.

`diff(expr, v, n)` :- differentiate `expr` with respect to `v` `n` times.

Example 4.2: - Find the following derivative

$$1)\ \frac{d}{dx}(e^x \sin ax) \quad 2)\ \frac{d^3}{dt^3}(t^3 + \tan\ t) \quad 3)\frac{\partial}{\partial y}(x^2 + y^2 - 3xy)$$

```
 syms x y t a
1) diff(exp(x)*sin(a*x))
   ans = exp(x)*sin(a*x) + a*exp(x)*cos(a*x)

  diff(exp(x)*sin(a*x),x)
  ans = exp(x)*sin(a*x) + a*exp(x)*cos(a*x)

2) diff(t^3+tan(t),t,3)
```

```
   ans =2*(tan(t)^2 + 1)^2 + 4*tan(t)^2*(tan(t)^2 + 1) + 6

3) diff(x^2+y^2-3*x*y,y)
   ans =2*y - 3*x
```

jacobian

Compute Jacobian matrix

Syntax
`jacobian(f, v)`

Description
`jacobian(f, v):-` computes the Jacobian of the scalar or vector `f` with respect to `v`. The `(i, j)`-th entry of the result is $\partial f(i)/\partial v(j)$. If `f` is scalar, the Jacobian of `f` is the gradient of `f`. If `v` is a scalar, the result equals to `diff(f, v)`.

Example 4.3: - Let $x = rcos\theta$ and $y = rsin\theta$ then find $J = \frac{\partial(x,y)}{\partial(r,\theta)}$.

```
 syms x y r th
  J=jacobian([x; y], [r th])
  x=r*cos(th);
  y=r*sin(th);
  J=jacobian([x; y], [r th])
  J = [ cos(th), -r*sin(th)]
      [ sin(th),  r*cos(th)]
  det(J)
  ans = r*cos(th)^2 + r*sin(th)^2
  simplify(ans)
  ans = r
```

4.3 Integration

int:-Integrate symbolic expression

Syntax
```
int(expr)
int(expr, v)
int(expr, a, b)
int(expr, v, a, b)
```

Description

`int(expr)` :- returns the indefinite integral of `expr` with respect to its symbolic variable as defined by `symvar`.

`int(expr,v)` :- returns the indefinite integral of `expr` with respect to the symbolic scalar variable `v`.

`int(expr,a,b)` :- returns the definite integral from *a* to *b* of `expr` with respect to the default symbolic variable. *a* and *b* are symbolic or double scalars.

`int(expr,v,a,b)` :- returns the definite integral of `expr` with respect to `v` from *a* to *b*.

Example 4.4: - Find the following integral

$$1)\ \int \frac{1}{2+x^2}dx \qquad 2)\ \int xe^x dx \qquad 3)\ \int_0^1 \sqrt{1-x^2}dx$$

```
1) syms x
   int(1/(2+x^2),x)
   ans =(2^(1/2)*atan((2^(1/2)*x)/2))/2

2) int(x*exp(x))
   ans = exp(x)*(x - 1)

3) int(sqrt(1-x^2),x,0,1)
   ans = pi/4
```

4.4 Symbolic Summation

symsum:- Evaluate symbolic sum of series

Syntax

```
r = symsum(expr)
r = symsum(expr, v)
r = symsum(expr, a, b)
r = symsum(expr, v, a, b)
```

Description

`r = symsum(expr)` :- evaluates the sum of the symbolic expression `expr` with respect to the default symbolic variable `defaultVar` determined by `symvar`. The value of the default variable changes from `0` to `defaultVar - 1`.
`r = symsum(expr,v)` :- evaluates the sum of the symbolic expression `expr` with respect to the symbolic variable `v`. The value of the variable `v` changes from `0` to `v - 1`.
`r = symsum(expr,a,b)` :- evaluates the sum of the symbolic expression `expr` with respect to the default symbolic variable `defaultVar` determined by `symvar`. The value of the default variable changes from `a` to `b`.
`r = symsum(expr,v,a,b)` :- evaluates the sum of the symbolic expression `expr` with respect to the symbolic variable `v`. The value of the default variable changes from `a` to `b`.

Example 4.5:- Find the sum of the following series

$$1)\ \sum_{i=0}^{n} i \qquad 2)\ \sum_{n=1}^{\infty} \frac{1}{n^2} \qquad 3)\ \sum_{k=0}^{n} x^k \qquad 4)\ \sum_{k=1}^{n} \frac{1}{k} - \frac{1}{k+1}$$

```
1) syms x n k
    symsum(k,1,n)
    ans = (n*(n + 1))/2
 or
   symsum(n+1)
   ans = n^2/2 + n/2

2) s1=symsum(1/k^2,1,inf)
   s1 = pi^2/6

3) s2=symsum(x^k,k,0,inf)
   s2 = piecewise([1 <= x, Inf], [abs(x) < 1, -1/(x - 1)])

4) symsum(1/k-1/(k+1),k,1,n)
  ans = psi(n + 1) - psi(n + 2) + 1
  simplify(ans)
  ans = 1 - 1/(n + 1)
```

4.5 Taylor Series

taylor:- Taylor series expansion

Syntax

```
taylor(f)
taylor(f, n)
taylor(f, a)
taylor(f, n, v)
taylor(f, n, v, a)
```

Description

`taylor(f)` :- returns the fifth order Maclaurin polynomial approximation to `f`.
`taylor(f,n)` : − returns the (`n−1`)-order Maclaurin polynomial approximation to `f`. Here `n` is a positive integer.
`taylor(f,a)` :- returns the fifth order Taylor series approximation to `f` about point `a`. Here `a` is a real number. If `a` is a positive integer or if you want to change the expansion order, use `taylor(f,n,a)` to specify the base point and the expansion order.

`taylor(f,n,v)` : − returns the (`n−1`)-order Maclaurin polynomial approximation to `f`, where `f` is a symbolic expression representing a function and `v` specifies the independent variable in the expression. `v` can be a string or symbolic variable.
`taylor(f,n,v,a)` : − returns the Taylor series approximation to `f` about `a`. The argument `a` can be a numeric value, a symbol, or a string representing a numeric value or an unknown. If `a` is a symbol or a string, do not omit `v`.

Example 4.6:- Find Maclaurin series of e^x and Taylor series of it about $x_0 = 1$.

```
syms x
taylor(exp(x))
ans = x^5/120 + x^4/24 + x^3/6 + x^2/2 + x + 1

taylor(exp(x),10)
ans = x^9/362880 + x^8/40320 + x^7/5040 + x^6/720 +
x^5/120 + x^4/24 + x^3/6 + x^2/2 + x + 1

taylor(exp(x),3,1)
ans = exp(1) + exp(1)*(x - 1) + (exp(1)*(x - 1)^2)/2
```

4.6 Functional inverse

inverse:- Functional inverse

Syntax

```
g = finverse(f)
g = finverse(f,v)
```

Description

`g = finverse(f)` returns the functional inverse of `f`. `f` is a scalar `sym` representing a function of one symbolic variable, say `x`. Then `g` is a scalar `sym` that satisfies `g(f(x)) = x`. That is, `finverse(f)` returns f^{-1}, provided f^{-1} exists.

`g = finverse(f,v)` uses the symbolic variable `v`, where `v` is a `sym`, as the independent variable. Then `g` is a scalar `sym` that satisfies `g(f(v)) = v`. Use this form when `f` contains more than one symbolic variable.

Example 4.7:- Find inverse of following functions

1) $f(x) = 3x - 1$ 2) $g(x) = 2 + e^x$ 3) $h(x) = \sin(2x + 1)$

```
syms x
1) f=3*x-1;
   finverse(f)

   ans = x/3 + 1/3

2) g=2+exp(x);
   finverse(g)
   ans = log(x - 2)

3) h=sin(2*x+1);
   finverse(h)
   Warning: finverse(sin(2*x + 1)) is not unique.
   ans = asin(x)/2 - ½
```

4.8 Functional composition

compose:-Functional composition

Syntax

```
compose(f,g)
compose(f,g,z)
compose(f,g,x,z)
compose(f,g,x,y,z)
```

Description

`compose(f,g)` :- returns `f(g(y))` where `f = f(x)` and `g = g(y)`. Here `x` is the symbolic variable of `f` as defined by `symvar` and `y` is the symbolic variable of `g` as defined by `symvar`.

`compose(f,g,z)` :- returns `f(g(z))` where `f = f(x)`, `g = g(y)`, and `x` and `y` are the symbolic variables of `f` and `g` as defined by `symvar`.

`compose(f,g,x,z)` :- returns `f(g(z))` and makes `x` the independent variable for `f`. That is, if `f = cos(x/t)`, then `compose(f,g,x,z)` returns `cos(g(z)/t)` whereas `compose(f,g,t,z)` returns `cos(x/g(z))`.

`compose(f,g,x,y,z)` :- returns `f(g(z))` and makes `x` the independent variable for `f` and `y` the independent variable for `g`. For `f = cos(x/t)` and `g = sin(y/u)`, `compose(f,g,x,y,z)` returns `cos(sin(z/u)/t)` whereas `compose(f,g,x,u,z)` returns `cos(sin(y/z)/t)`.

Examples 4.9: -

Suppose

```
syms x y z t u;
f = 1/(1 + x^2); g = sin(y); h = x^t; p = exp(-y/u);
```

Then

```
a = compose(f,g)
b = compose(f,g,t)
c = compose(h,g,x,z)
d = compose(h,g,t,z)
e = compose(h,p,x,y,z)
f = compose(h,p,t,u,z)
```

returns:

```
a = 1/(sin(y)^2 + 1)
b = 1/(sin(t)^2 + 1)
c = sin(z)^t
d = x^sin(z)
e =(1/exp(z/u))^t
f = x^(1/exp(y/z))
```

Exercises

Q1) Write a program to find derivative of input function by definition.

Q2) Write a program to find the area under function $f(x)$ on interval $[a, b]$ by Riemann integral.

Q3) Write a program to find tangent line of function $f(x)$ at x_0 and draw the graphs.

Q4) Write a program to find local maximum, Local minimum and Inflection point of function $f(x)$ then draw its graph.

Q5) Write a program to find the area between two input functions.

Q6) Write a program to find the area under function $f(x)$ on interval $[a, b]$.

Q7) Write a program to find the area under function $f(x)$ if exist.

Q8) Write a program to find local maximum, local minimum and saddle point of function $f(x, y)$ then draw its graph.

Q9) Write a program to find the parametric equation of line in R^3 that pass through the two points and plot of its graph.

Q10) Write a program to find the equation of plane pass through the three points and plot of its graph.

Q11) Write a program to find the eq. of tangent plane of the surface $f(x, y, z)$ at the point (x_0, y_0, z_0).

Chapter 5
Linear Algebra

5.1 Solving Linear Systems

Using mldivide or x=A\b

Suppose that A is a non-singular n × n matrix and b is a column vector of length n. Then typing `x = A\b` numerically computes the unique solution to `A*x = b`. Type help mldivide for more information.

Example 5.1:- use MATLAB to solve the following linear system

$$\begin{aligned} x - 2y + z &= -1 \\ x + y - z &= 0 \\ 2x + y - 3z &= 2 \end{aligned}$$

solve:-

```
>>A=[1 -2 1;1 1 -1;2 1 -3]
>> b=[-1;0;2]
>> X=A\b

X =
   -0.8000
   -0.6000
   -1.4000
```

Using inv(A)

Suppose that A is a non-singular n × n matrix and b is a column vector of length n. Then can solved linear system $Ax = b$ by $x = inv(A) * b$

Example 5.1 can be solving by

```
X=inv(A)*b
```

Using linsolve

X = linsolve(A,B) solves the linear system AX = B using LU factorization with partial pivoting when A is square and QR factorization with column pivoting otherwise. The number of rows of A must equal the number of rows of B.

5.2 Calculating Eigenvalues and Eigenvectors

The eigenvalues of a square matrix A are calculated with `eig(A)`. The command `[U, R] = eig(A)` calculates both the eigenvalues and eigenvectors. The eigenvalues are the diagonal elements of the diagonal matrix R, and the columns of U are the eigenvectors.
Here is an example illustrating the use of eig.

Example 5.2:-

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

>> eig(A)

ans =

      1

     -1

      2

>> [U, R] = eig(A)

U =

        0  -0.4082  -0.8165

        0  -0.8165  -0.4082

   1.0000   0.4082  -0.4082

R =

   1   0   0

   0  -1   0

   0   0   2
```

The eigenvector in the first column of U corresponds to the eigenvalue in the first column of R, and so on.

5.3 MATLAB Linear Algebra Functions

Euclidean Norm

The length of a vector is called the norm of the vector. From Euclidean geometry, the distance between two points is the square root of the sum of the squares of the distances in each dimension. Thus, the notation and definition of the Euclidean norm is

$$\|X\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Note that the norm can be defined in terms of the inner product:

$$\|X\| = \sqrt{(X, X)}$$

Vector norm

`n = norm(X)` returns the Euclidean Norm of vector X.

`n = norm(X,p)` returns a different kind of norm, depending on the value of p.

Example 5.3: - Let `a=(2 3 -4)` and `b=(7 -4 5)` then find the norm and unit vector for each of a and b then find the angle between a and b.

```
>> a=[2 3 -4];

>> b=[7 -4 5];

>> na=norm(a)

na =   5.3852

>> nb=norm(b)

nb =   9.4868

>> ua=a/norm(a)

ua =   0.3714    0.5571   -0.7428

>> ub=b/norm(b)

ub =   0.7379   -0.4216    0.5270

>> theta = acos(dot(a,b)/(norm(a)*norm(b)))

theta =  1.9309
```

Rank of Matrix

rank(A) :- Returns the rank of the matrix A, which is the number of independent rows of A.

Example 5.4:- Let `M = [0 2 2 3 -4; -2 4 2 -1 -6; 3 -4 -1 2 8];` then

```
        rank(M)=3
```

Reduced row echelon form

R = rref(A) produces the reduced row echelon form of A using Gauss Jordan elimination with partial pivoting.

Example 5.5:- Use *rref* to solve the following linear system

$$\begin{aligned} x - 2y + z &= -1 \\ x + y - z &= 0 \\ 2x + y - 3z &= 2 \end{aligned}$$

sol:-
```
>> A=[1 -2 1;1 1 -1;2 1 -3];
>> b=[-1;0;2];
>> R=rref([A b]);
>> Sol=R(:,4)'
Sol = -0.8000   -0.6000   -1.4000
```

Other functions

There are useful function listed in below table

1	C = dot(A,B)	returns the scalar dot product of A and B.
2	C = cross(A,B)	returns the cross product of A and B.
3	L = tril(A)	returns the lower triangular part of A.
4	U = triu(A)	returns the Upper triangular part of A.

5	b = trace(A)	is the sum of the diagonal elements of the matrix A.
6	Q = orth(A)	returns an orthonormal basis for the range of A. The columns of Q are vectors, which span the range of A. The number of columns in Q is equal to the rank of A.

Exercises

Q1) Mixing MATLAB built-in functions to do the following by one-line statement in command window

a) Check whether the matrix $\mathbf{A}_{n\times m}$ is zero matrix or not.
b) Check whether the matrix $\mathbf{A}_{n\times n}$ is identity matrix or not.
c) Check whether the matrix $\mathbf{A}_{n\times n}$ is diagonal matrix or not.
d) Check whether the matrix $\mathbf{A}_{n\times n}$ is Symmetric matrices or not.
e) Check whether the matrix $\mathbf{A}_{n\times n}$ is Singular matrix or not.
f) Check whether the matrix $\mathbf{A}_{n\times n}$ is Hermitian matrix or not.
g) Check whether the matrix $\mathbf{A}_{n\times n}$ is Orthogonal matrix or not.
h) Check whether the matrix $\mathbf{A}_{n\times n}$ is Idempotent matrix or not.
i) Check whether the matrix $\mathbf{A}_{n\times n}$ is Involuntary matrix or not.
j) Check whether the matrix $\mathbf{A}_{n\times n}$ is Nilpotent matrix of power *p* or not.
k) Check whether the all eigenvalue of matrix $\mathbf{A}_{n\times n}$ is real or not.
l) Check whether two vectors in R^3 are parallel or not.
m) Find the angle between two vectors in R^3.

Q2) Write a program to input three vectors in R^3 then find out are independent or not.

Q3) Write a program to solve linear system by using Cramer's rule.

Q4) Write a program input three vectors V_1, V_2 and V_3in R^3 then find the area of triangle $V_1V_2V_3$.

Chapter 6

Solving Differential Equation

6.1 Single Differential Equation

The function dsolve computes symbolic solutions to ordinary differential equations. The equations are specified by symbolic expressions containing the letter D to denote differentiation. The symbols D2, D3, ... DN, correspond to the second, third, ..., Nth derivative, respectively. Thus, D2y is the Symbolic Math Toolbox equivalent of d^2y/dt^2. The dependent variables are those preceded by D and the default independent variable is t. Note that names of symbolic variables should not contain D. The independent variable can be changed from t to some other symbolic variable by including that variable as the last input argument.

Syntax

```
dsolve('eq','cond1','cond2',...,'v')

or
dsolve(eq,cond1,'cond2)
```

Example 6.1:- Find the general solution of the first order differential equation

$$\frac{dy}{dt} + y = te^t$$

sol:-

```
>> dsolve('Dy + y = t*exp(t)')

ans =
     1/2*t*exp(t)-1/4*exp(t)+exp(-t)*C1

Or

>> syms y(t)
>> dsolve(diff(y)+y==t*exp(t))
ans =
(exp(t)*(2*t - 1))/4 + C1*exp(-t)
```

Example 6.2:- Find the partical solution of the first order differential equation

$$\frac{dy}{dt} = 1 + y^2, \quad y(0) = 1$$

```
dsolve('Dy=1+y^2')
```

uses *y* as the dependent variable and *t* as the default independent variable.

The output of this command is

```
ans = tan(t+C1)
```

To specify an initial condition, use

```
y = dsolve('Dy=1+y^2','y(0)=1')
```

This produces

```
y =tan(t+1/4*pi)


or
>> dsolve(diff(y)==1+y^2,y(0)==1)
ans =
tan(t + pi/4)
```

Example 6.3: - Nonlinear equations may have multiple solutions, even when initial conditions are given:

```
x = dsolve('(Dx)^2+x^2=1','x(0)=0')
```

results in

```
x =
   [ sin(t)]
   [ -sin(t)]
Or
>> syms x(t)
>> x = dsolve(diff(x)^2+x^2==1,x(0)==0)
x =
 cosh(t*1i + (pi*1i)/2)
 cosh(t*1i - (pi*1i)/2)
```

Example 6.4:- Here is a second order differential equation with two initial conditions. The commands

```
y = dsolve('D2y=cos(2*x)-y','y(0)=1','Dy(0)=0', 'x');
simplify(y)
```

produce

```
ans = 4/3*cos(x)-2/3*cos(x)^2+1/3
or
>> y = dsolve(diff(y,2)==cos(2*x)-
y,y(0)==1',diff(y(0))==0)
y =
(5*cos(x))/3 + C20*sin(x) + sin(x)*(sin(3*x)/6 + sin(x)/2)
- (2*cos(x)*(6*tan(x/2)^2 - 3*tan(x/2)^4 +
1))/(3*(tan(x/2)^2 + 1)^3)
>> simplify(y)
ans =
(4*cos(x))/3 - (2*cos(x)^2)/3 + C20*sin(x) + 1/3
```

6.2 Several Differential Equations

The function dsolve can also handle several ordinary differential equations in several variables, with or without initial conditions. For example, here is a pair of linear, first-order equations.

```
dsolve('eq1','eq2','eq3', ...,'cond1','cond2',...,'v')

or
dsolve(eq1,eq2,eq3,...,cond1,'cond2,...,)
```

Example 6.5: - solve linear system of differential equations

$$x' = 3x + 4y$$
$$y' = -4x + 3y$$

```
S = dsolve('Dx = 3*x+4*y', 'Dy = -4*x+3*y')
```

The computed solutions are returned in the structure S. You can determine the values of f and g by typing

```
x = S.x
x = exp(3*t)*(C1*sin(4*t)+C2*cos(4*t))
y = S.y
y = exp(3*t)*(C1*cos(4*t)-C2*sin(4*t))
```

If you prefer to recover f and g directly as well as include initial conditions, type

```
[x,y] = dsolve('Dx=3*x+4*y,Dg =-4*x+3*y', 'x(0) = 0,y(0) = 
1')
f = exp(3*t)*sin(4*t)
g = exp(3*t)*cos(4*t)


or
>> syms x(t) y(t)
>> S = dsolve(diff(x) == 3*x+4*y, diff(y) == -4*x+3*y);
>> S.x
ans =
C22*cos(4*t)*exp(3*t) + C21*sin(4*t)*exp(3*t)
>> S.y
ans =
C21*cos(4*t)*exp(3*t) - C22*sin(4*t)*exp(3*t)
```

Chapter 7
Introduction to Numerical analysis

7.1 Numerical analysis

Is the study of algorithms that use numerical approximation (as opposed to general symbolic manipulations) for the problems of mathematical analysis.

7.1.1 Numerical Errors

1. True Error

True Error is defined as the difference between the true value in a calculation and the approximate value found using a numerical method etc.

True Error = True Value – Approximate Value

2. Approximate Error

Approximate error is defined as the difference between the present approximation and the previous approximation.

Approximate Error (E_a) = Present Approximation – Previous Approximation

3. Sources of Error

a. Human Error
It causes when we use inaccurate measurement of data or inaccurate representation of mathematical constants.

b. Truncation Error
It causes when we are forced to use mathematical techniques which give approximate, rather than exact answer.

c. Round-off Error
These types of errors are associated with the limited number of digits numbers in the computers.

7.2 Solutions of Equations in one Variable

This section deals with finding solutions of algebraic and transcendental equations of the forms

$$f(x) = 0, \qquad (1)$$

where we want to solve for the unknown *x*.

7.2.1 Locating the position of roots (Programming Method)

To locate the position of roots of the function (equation) $f(x) = 0$ by using ***programming method,*** let $f(x)$ be continuous function on the interval $[a, b]$. We divide the interval $[a, b]$ into n subintervals $a = x_0 < x_1 < x_2 < \dots < x_n = b$ where $x_i = a + i\,h$, $i = 0,1,\dots,n$; $h=\dfrac{b-a}{n}$. If $f(x_i)\times f(x_{i+1}) < 0$ for any $0 \le i \le n$, then there exits c, $a < c < b$ for which $f(c)=0$.

Example 7.1:

Find the approximate location of roots of the function

1. $f(x) = x^4 - 7x^3 + 3x^2 + 26x - 10 = 0$ on the interval [-8,8] with *n*=4 and *n*=8.
2. $f(x) = x^3 + 4x^2 - 10 = 0$ on the interval [1,2] with *n*=5.
3. $f(x) = x^3 - \dfrac{132}{32}x^2 + \dfrac{28}{32}x + \dfrac{147}{32}$ on the interval [-1,4] with *n*=5.

Solution: (1): Let *n*=4, $h=\dfrac{b-a}{n} = \dfrac{8-(-8)}{4} = 4$:

X	-8	-4	0	4	8
f(*x*)	+	$\underbrace{+\quad}$	$\underbrace{\quad -}$	$\underbrace{-\quad}$	$\underbrace{\quad +}$

There is a root between (-4,0) and (4,8).

If n=8, h=2:

x	-8	-6	-4	-2	0	2	4	6	8
$f(x)$	+	+	+	+	-	+	-	+	+

There is a root between (-2,0), (0,2), (2,4) and (4,6).

Solution: (2): Let n=5, h=0.2

x	1	1.2	1.4	1.6	1.8	2
$f(x)$	-	-	+	+	+	+

There is a root between (1.2,1.4).

Solution: (3): Let n=5, h=1

x	-1	0	1	2	3	4
$f(x)$	-	+	+	-	-	+

There is a root between (-1, 0), (1, 2) and (3, 4), see Figure 2.1.

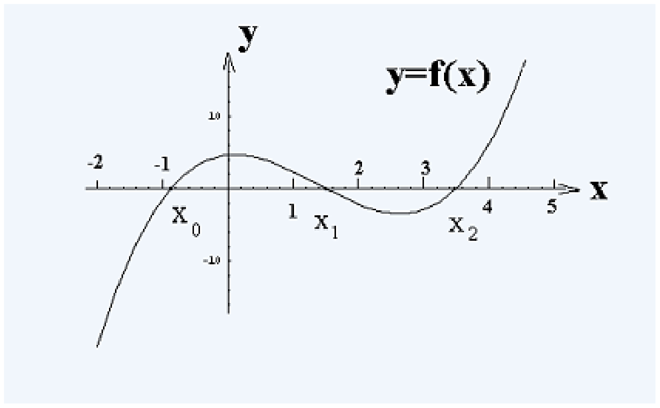


Figure 1. Graph of $f(x)=x^3-\frac{132}{32}x^2+\frac{28}{32}x+\frac{147}{32}$

7.2.2 Numerical methods:

We shall discuss some numerical methods for solving algebraic and transcendental equations.

1. Bisection Method

Suppose a continuous function f defined on the interval $[a, b]$ is given with $f(a)$ and $f(b)$ of opposite sign (i.e. $f(a)\times f(b)<0$). Then by intermediate value theorem 1.2.5 there exists a point $c\in(a, b)$ such that $f(c)=0$. If we choose the midpoint $c=\frac{b+a}{2}$, then three possibilities arise:

$$\text{If } f(a)\times f(c)\begin{cases} <0 & \text{there is a root between } a, c \Rightarrow d=\frac{a+c}{2}, \\ >0 & \text{there is a root between } b, c \Rightarrow d=\frac{b+c}{2}, \\ =0 & c \text{ is exact root ((Stop)).} \end{cases}$$

We stop iteration if the interval width is as small as desired i.e. $|x_i - x_{i+1}| < \varepsilon$ for any i.

Example 7.2: Find an approximate root of $f(x)=x^2-2$ in the interval $[1, 2]$ by using Bisection method if it's possible with error $\leq \varepsilon = 10^{-4}$.

Solution: It is possible to use bisection method because f is continuous on [1, 2] and $f(a)=f(1)=-1=-$ve; $f(b)=f(2)=2=+$ve,

$\therefore \quad f(a)\times f(b)=-2<0$ (i.e. a root lies between 1 and 2).

We have the formula

$$x_1=\frac{a+b}{2}=\frac{1+2}{2}=1.5, \qquad |x_1-a| = 0.5>\varepsilon.$$

Find x_2:

$f(x_1) = 0.25 = +\text{ve}$. So the root lies between 1 and 1.5.

$\therefore\ x_2 = \frac{x_1 + a}{2} = 1.25$, $\qquad |x_1 - x_2| = 0.25 > \varepsilon$.

Find x_3:

$f(x_2) = -0.437 = -\text{ve}$. So the root lies between 1.25 and 1.5.

$\therefore\quad x_3 = \frac{x_1 + x_2}{2} = 1.375$, $\qquad |x_2 - x_3| = 0.125 > \varepsilon$.

$\vdots$

Stop iteration if $|x_i - x_{i+1}| < 10^{-4}$ for any $i=1, 2, \ldots$.

Example 7.3: Find an approximate root of $f(x) = x\log(x) - 1$ in the interval [1, 2] by using Bisection method with error $\leq \varepsilon = 10^{-3}$.

Solution: $f(x_0) = f(1) = -1 = -\text{ve}$; $f(x_1) = f(2) = 0.3863 = +\text{ve}$.

$\therefore\quad f(x_0) \times f(x_1) = -0.3863 < 0$ (i.e. a root lies between 1 and 2).

We have the formula

$$x_2 = \frac{x_0 + x_1}{2} = \frac{1+2}{2} = 1.5, \qquad |x_1 - x_2| = 0.5 > \varepsilon.$$

Find x_3:

$f(x_2) = -$ ve. So the root lies between 1.5 and 2.

$\therefore\quad x_3 = \frac{x_1 + x_2}{2} = 1.75$, $\qquad |x_2 - x_3| = 0.25 > \varepsilon$.

Find x_4:

$f(x_3) = -$ ve. So the root lies between 1.75 and 2.

$\therefore\quad x_4 = \frac{x_1 + x_3}{2} = 1.875$, $\qquad |x_3 - x_4| = 0.125 > \varepsilon$.

Find x_5:

$f(x_4)$ =+ve. So the root lies between 1.75 and 1.875.

$\therefore \quad x_5 = \frac{x_3 + x_4}{2} = 1.8125,$

Similarly, we get

x_6=1.78125,

x_7=1.765625,

x_8=1.7578125,

x_9=1.76171875,

x_{10}=1.763671875,

x_{11}=1.762953125.

So the root is x_{11}=1.762953125 with error $\leq \varepsilon = 10^{-3}$.

Theorem 7.1: Let $f \in C[a_n, b_n]$ and suppose $f(a) \times f(b) < 0$. The bisection procedure generates a sequence $\{P_n\}$ approximating P with the property $|P_n - P| \leq \frac{b-a}{2^n}; \; n \geq 1$.

Proof: For each $n \geq 1$ we have $b_n - a_n = \frac{b-a}{2^{n-1}}$ and $P \in (a_n, b_n)$. Since

$P_n = \frac{1}{2}(a+b)$ for all $n \geq 1$, it follows that

$$|P_n - P| \leq \frac{1}{2}(b_n - a_n) = \frac{b-a}{2^n}; \text{ for all } n \geq 1.$$

Example 7.4: Determine approximately how many bisection iterations are necessary to solve $f(x)$ with error $\leq \varepsilon$ over $[a, b]$.

Solution: We must find an integer n that will satisfy $|P_n - P| \leq \frac{b-a}{2^n} \leq \varepsilon$.

$\therefore \quad \frac{b-a}{\varepsilon} \leq 2^n$.

By taking the natural logarithm to the both sides of the above equation, we get

$$\ln\left(\frac{b-a}{\varepsilon}\right) \le n\ln(2), \text{ which implies that } n \ge \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln(2)}.$$

For example, if $f(x)=x^3+4x-10=0$, $\varepsilon = 10^{-5}$ and [1,2].

$$n \ge \frac{\ln\left(\frac{2-1}{10^{-5}}\right)}{\ln(2)} = \frac{\ln(10^5)}{\ln(2)} = \frac{11.512925}{0.693147} \approx 16.6096 \Rightarrow n=17 \text{ (number of iterations).}$$

2. False-Position method (Regula falsi method)

The method of False-Position is often referred to as the method of linear interpolation or the Latin equivalent (Regula falsi method). It is a method that is sometimes used in the attempt to speed up the bisection method. Suppose a continuous function f defined on the interval [a, b] is given with $f(a)$ and $f(b)$ of opposite sign (i.e. $f(a)\times f(b)<0$). To derive a formula for false-position method, approximate the graph of f by a straight line on [a, b] connecting (a, $f(a)$) and (b, $f(b)$) which intersect x-axis at (c,0) where c is more approximate to the exact (actual) root than a and b.

To obtain a formula for c we use the slop equality:

$$\frac{f(b)-f(a)}{b-a} = \frac{f(b)-y}{b-x} \Rightarrow \frac{f(b)-f(a)}{b-a} = \frac{f(b)-0}{b-c} \Rightarrow c = \frac{af(b)-bf(a)}{f(b)-f(a)}.$$

To find another approximation:

$$\text{If } f(a)\times f(c) \begin{cases} <0 & \text{there is a root between } a, c \Rightarrow d=\dfrac{af(c)-cf(a)}{f(c)-f(a)}, \\ >0 & \text{there is a root between } b, c \Rightarrow d=\dfrac{bf(c)-cf(b)}{f(c)-f(b)}, \\ =0 & c \text{ is exact root ((Stop)).} \end{cases}$$

We stop iteration if the interval width is as small as desired i.e.

$|x_i - x_{i+1}| < \varepsilon$ for any i.

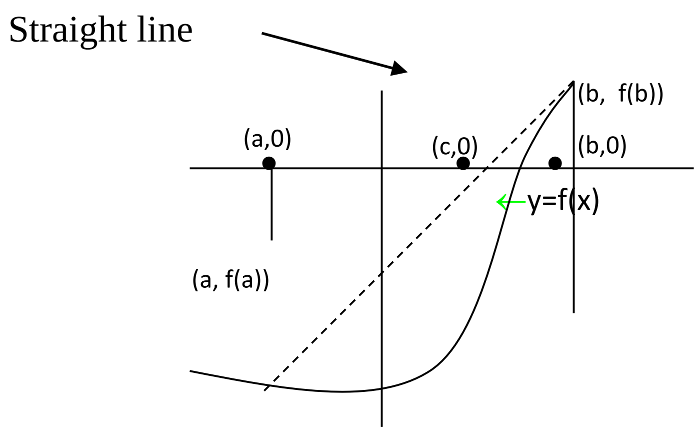


Figure 2 Linear Interpolation

Example 7.5: Find an approximate root of $f(x)=x^2-2x-1=0$, between [-1,0] by using False-Position method with error $\leq \varepsilon = 10^{-5}$.

Solution: Let x_1=-1, x_2=0, $f(x_1)$=2=+ve and $f(x_2)$=−1=−ve.

$\therefore \quad f(x_1)\times f(x_2)=-2<0$ (i.e. a root lies between −1 and 0).

We have the formula

$$x_3=\frac{x_1 f(x_2)-x_2 f(x_1)}{f(x_2)-f(x_1)}.$$

$$\therefore \quad x_3=\frac{-1\times f(0)-0\times f(-1)}{f(0)-f(-1)}=\frac{-1\times(-1)-0\times 2}{-1-2}=0.33. \quad |x_3-x_2|=0.33>\varepsilon.$$

Find x_4:

$f(x_3)$=-0.22=-ve. So the root lies between -1 and 0.33.

$$\therefore \quad x_4=\frac{x_1 f(x_3)-x_3 f(x_1)}{f(x_3)-f(x_1)}=-0.4. \qquad |x_4-x_3|=0.07>\varepsilon.$$

⋮

Stop iteration if $|x_i-x_{i+1}|\leq\varepsilon$ for any i=1, 2, ….

Example 7.6: A real root of the equation $f(x) = x^3 - 5x + 1 = 0$ lies in the interval (0, 1). Perform four iterations of the False-position method to obtain this root.

Solution: We have $x_0 = 0,\ x_1 = 1,\quad f_0 = f(x_0) = 1,\ \ f_1 = f(x_1) = -3$.

$$x_2 = x_1 - \left[\frac{x_1 - x_0}{f_1 - f_0}\right] f_1 = \frac{x_0 f_1 - x_1 f_0}{f_1 - f_0} = 0.25, \qquad f_2 = -0.234375.$$

Since $f_0 \times f_2 < 0$, there is a root *r* between $(x_0,\ x_2)$. Therefore,

$$x_3 = x_2 - \left[\frac{x_2 - x_0}{f_2 - f_0}\right] f_2 = \frac{x_0 f_2 - x_2 f_0}{f_2 - f_0} = 0.202532, \qquad f_3 = -0.004352.$$

Since $f_0 \times f_3 < 0$, there is a root r between $(x_0,\ x_3)$. Therefore,

$$x_4 = x_3 - \left[\frac{x_3 - x_0}{f_3 - f_0}\right] f_3 = \frac{x_0 f_3 - x_3 f_0}{f_3 - f_0} = 0.201654, \qquad f_3 = -0.000070.$$

Since $f_0 \times f_4 < 0$, there is a root r between $(x_0,\ x_4)$. Therefore,

$$x_5 = x_4 - \left[\frac{x_4 - x_0}{f_4 - f_0}\right] f_4 = \frac{x_0 f_4 - x_4 f_0}{f_4 - f_0} = 0.201640$$

Example 7.7: Use the False-position method to determine the root of the equation $\cos(x) - xe^x = 0$. Taking the initial approximations as $x_0 = 0,\ x_1 = 1$.

Solution: We have $x_0 = 0,\ x_1 = 1,\quad f_0 = f(x_0) = 1,\ \ f_1 = f(x_1) = -2.177979523$.

$$x_2 = x_1 - \left[\frac{x_1 - x_0}{f_1 - f_0}\right] f_1 = \frac{x_0 f_1 - x_1 f_0}{f_1 - f_0} = 0.3146653378,$$

$$f_2 = 0.519871175.$$

Since $f_1 \times f_2 < 0$, there is a root r between (x_1, x_2). Therefore,

$$x_3 = x_2 - \left[\frac{x_2 - x_1}{f_2 - f_1}\right] f_2 = \frac{x_1 f_2 - x_2 f_1}{f_2 - f_1} = 0.4467281466,$$

$$f_3 = 0.203544710.$$

Since $f_1 \times f_3 < 0$, there is a root r between (x_1, x_3). Therefore,

$$x_4 = x_3 - \left[\frac{x_3 - x_1}{f_3 - f_1}\right] f_3 = \frac{x_1 f_3 - x_3 f_1}{f_3 - f_1} = 0.4940153366.$$

$$\vdots$$

3. Newton-Raphson method

Suppose that the function f is twice continuously differentiable on the interval [*a*, *b*], that is $f \in C^2$ [*a, b*]. Let *c* be an approximation to the exact root λ such that $f'(c) \neq 0$ and $|c-\lambda|$ is small. Consider the first degree Taylor polynomial of f expanded about *c* i.e.

$$f(x)=f(c)+(x-c)\, f'(c) + \frac{(x-c)^2}{2!} f''(\varphi), \qquad (3)$$

where ϕ lies between *x* and *c*. Since $f(\lambda)=0$, equation (2.3) with $x=\lambda$ gives

$$0= f(c)+(\lambda-c)\, f'(c) + \frac{(\lambda-c)^2}{2!} f''(\varphi)$$

Newton-Raphson is derived by assuming that the term involving $(\lambda - c)^2$ is negligible and that

$$0 \approx f(c)+(\lambda-c)\, f'(c) \Rightarrow \lambda \approx c - \frac{f(c)}{f'(c)}$$

This should be a better approximation to λ than is *c*. This sets the stage for the Newton-Raphson method, which involves generating the sequence $\{x_n\}$ defined by

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})},\ n=1, 2, \ldots.$$

Example 7.8: Perform four iteration of the Newton-Raphson method to find an approximate root of the equation $x^3 - 5x + 1 = 0$ near $x_0 = 0.5$.

Solution: Let $f(x)=x^3-5x+1=0$. Here $f'(x)=3x^2-5$.

Using the Newton-Raphson method

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)},$$

we get

$$x_{k+1}=x_k-\frac{x_k^3-5x_k+1}{3x_k^2-5}=\frac{2x_k^3-1}{3x_k^2-5},\quad \text{for k=0, 1, 2, … .}$$

Starting with $x_0=0.5$, we obtain

$$x_1=0.176471,$$

$$x_2=0.201568,$$

$$x_3=0.201640.$$

The exact value correct to six decimal places is 0.2012640.

Example 7.9: If $f(x)=x^3-x+1$ and $x_0=1$, what are x_1 and x_2 in the Newton iteration?.

Solution: From the basic formula, $x_1=x_0-\dfrac{f(x_0)}{f'(x_{0)}}$.

Now, $f'(x)=3x^2-1$, and so $f'(x_0)=f'(1)=2$. Also $f(x_0)=f(1)=1$.

$$\therefore \qquad x_1=1-\frac{1}{2}=\frac{1}{2}.$$

Similarly

$$x_2=x_1-\frac{f(x_1)}{f'(x_{1)}},\quad f(x_1)=f(\frac{1}{2})=\frac{5}{8}\text{ and } f'(x_1)=f'(\frac{1}{2})=-\frac{1}{4}.$$

$$\therefore \qquad x_2=\frac{1}{2}-\frac{5/8}{-1/4}=3.$$

Example 7.10: Use Newton-Raphson method to find an efficient method for computing:

(1) $\sqrt[n]{a},\ a > 0$; n=2, 3, …

(2) $\frac{1}{a},\quad a \neq 0$ without using division.

Solution (1): Let $x = \sqrt[n]{a}$. Then x^n-a=0 . And let $f(x)$= x^n-a, then $f'(x) = nx^{n-1}$.

$$\therefore \qquad x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^n - a}{nx_i^{n-1}} = \left(1 - \frac{1}{n}\right)x_i + \frac{a}{n}x_i^{1-n} \text{ for } i\text{=0, 1, 2, … .}$$

If, for example, we wish to compute $\sqrt{17}$ and begin with $x_0 = 4$, the successive approximations are as follows: given in rounded form to exhibit only correct figures.

$x_1 = 4.12$,

$x_2 = 4.123106$,

$x_3 = 4.1231056256177$,

$x_4 = 4.123105625617660549821409856$.

The value given by x_4 is correct to 28 figures.

Solution (2): Let $x = \frac{1}{a}$. Then $\frac{1}{x} - a = 0$. And let $f(x) = \frac{1}{x} - a$, then $f'(x) = -\frac{1}{x^2}$.

$$\therefore x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{\frac{1}{x_i} - a}{-\frac{1}{x_i^2}} = x_i + x_i - ax_i^2 = (2 - ax_i)x_i,\ i\text{=0, 1, 2, … .}$$

If, for example, we wish to compute $\frac{1}{21}$ and begin with $x_0 = 0.05$, the successive approximations are as follows:

$x_1 = 0.0475$,
$x_2 = 0.04761875$,
$x_3 = 0.047619047617188$,

$x_4 = 0.047619047619048$,

$\vdots$

The exact value of $\frac{1}{21}$ is 0.047619047619048.