**CHAPTER 1**

**Introduction**

MATLAB is a mathematical software package for high performance numerical computation and visualization. The basic building block of MATLAB is matrix. The name MATLAB stands for Matrix Laboratory.

MATLAB has notable applications in Mathematics and many engineering fields (for example Aerodynamics, Medical Science, CFD, signal processing, Chemical Industry, Image Processing etc.).

**Basics in MATLAB**

Simple computation may be carried out in the **Command Window** by entering an instruction at the prompt. Most of the arithmetic operations and common functions are available in MATLAB with usual notation and name with few exceptions. Commands and names are case sensitive.

**Used fixed constants:**

pi (π), i (imaginary)

**Built-in Functions**

Mathematical functions are available with commonly used name.

Note the following change:

log(#) natural logarithm(ln), log10(#) log base 10

Numerical calculations are similar to calculator.

Ex.

>>1+sin(pi/3) % by default results are shown using

ans = % ans= if not indicated.

1.8660

The default format shows approximately five significant decimal figures. Entering the command format long will display approximately 16 significant decimal figures.

Ex.

>> format long

>> result=1+sin(pi/3)

result =

1.866025403784439

**Note:** percent (%) sign is used to write comments.

semicolon (;) at the end of command suppress the output.

**Matrices**

All variables in MATLAB are treated as matrices or arrays.

**A row vector may be entered as**

>> x=[1 2 3] or x=[1,2,3]

x =

1 2 3

The elements, enclosed between the left bracket and the right bracket, are separated by either commas or spaces.

**A column vector may be entered as**

>> y = [4; 5; 6] or y = [4 5 6]’ or y = [4

5

6]

y =

4

5

6

Semicolons are used to separate the rows of a matrix.

**An example of a 3-by-4 matrix is**

B= [1 2 3 4

5 6 7 8

9 10 11 12]

or B = [1 2 3 4; 5 6 7 8; 9 10 11 12]

The elements of a matrix may be accessed as in standard matrix notation;

e.g. B(2,3) = 7. An entire row or column of a matrix may be accessed by using a colon instead of specific element.

>>B(2, :)

ans =

5 6 7 8

>> B(: , 3)

ans =

3

7

11

Linearly spaced vectors can be generated in the following ways:

**(a) Using colon (;)**

**a:b** % generates a vector with interval of 1 from

**a:h:b** % generates a vector with interval of 1 from

For examples,

>> x=1:5

x =

1 2 3 4 5

>> x1=1:0.5:3

x1 =

1 1.5 2 2.5 3

**(b) Use linspace**

**linspace (a, b, n)** % generate *n* values in [a, b] with equal length

**linspace (a, b)** % generate 100 values in [a, b} with equal length

**>> x2=linspace(1,2.5,4)**

**x2 =**

**1 1.5 2 2.5**

**Printing in MATLAB (Output)**

**Controlling Number of Digits**

The fixed-point numbers:

format short displays 5 digits

format long displays 16 digits

The floating-point representation:

format short e gives 5 digits plus the exponent

format long e gives 16 digits plus the exponent

The combined format (fixed point or floating depending on the magnitude of the number)

format short g

format long g

For results that are ratios of small integers, use

format rat

**Printing:** There are three common ways we can display the outputs;

1. By typing the name of a variable(displays the output indicating variable name).

>>clear

>> A=[1, 2.25 4.56];

>> B='AIUB';

>> A

A =

1.0000 2.2500 4.5600

>> B

B =

AIUB

1. By using “disp” builtin function. This displays output without variable name.

>>clear

>> A=[1, 2.25 4.56];

>> B='AIUB';

>>disp(A)

1.0000 2.2500 4.5600

>>disp(B)

AIUB

1. By using “**fprintf** “ function’

**Syntax:** fprintf(formatSpec, A1, A2, . . . , A3)

In formatSpec

\n is used for new line,

%12.8 is used for a fixed point field of width 12 with 8 digits after the decimal point and

%6s is used for string with width 6.

(a)

>>clear

>> A=[1.3 2 3.12; 1.23 2.45 3.84]

A =

1.3000 2.0000 3.1200

1.2300 2.4500 3.8400

>>fprintf('%5.0f %12.3f \n',A) % 5.0f converts floating point to integer

1 1.230

2 2.450

3 3.840

(b)

>>clear

>> x=0:0.5:2;

>> y=sin(x);

>>fprintf('%6s %12s\n','x','sin(x)');...

fprintf('%6.2f %12.6f\n',[x;y])

x sin(x)

0.00 0.000000

0.50 0.479426

1.00 0.841471

1.50 0.997495

2.00 0.909297

(c)

>>clear

>> x=0:0.5:2;

>>fprintf('x=%6.2f sin(x)=%10.7f\n',[x;sin(x)])

x= 0.00 sin(x)= 0.0000000

x= 0.50 sin(x)= 0.4794255

x= 1.00 sin(x)= 0.8414710

x= 1.50 sin(x)= 0.9974950

x= 2.00 sin(x)= 0.9092974

**Plotting in MATLAB**

In MATLAB curves are drawn by joining two consecutive points by line segment. Curves can be made smooth by closer points.

**(a) 2-D Plot**

plot(y) x = 1 : n (if not supplied)

plot(x,y) x, y are vectors

plot(x1, y1, . . .. , xn,yn)

title(‘plot title’) grid on grid off

xlabel(‘label for x-axis’) grid (toggles)

ylabel(‘label for y-axis’) hold on hold off

text(0.75, 0.65, “sin(x)’) box on

**Ex 1**. Plot the function  in .

>> x=-4:0.2:4;

>> y=7\*cos(x)+2\*x-1;

>> plot(x,y);grid

The script for 2-phase plot is as follows:

**Ex 2.** Plot the functions  and in .

**>> x=linspace(-3,5,40);**

**>> y1=7\*cos(x); y2=1-2\*x;**

**>> plot(x,y1,x,y2); grid**

**(b) 3-D plot**

 Command plot3(x,y,z)

Ex 3. Plot the function



**>> x=linspace(-5,5,50);**

**>> y=x;**

**>> z=4-x.^2-y.^2;**

**>> plot3(x,y,z); grid**

**Construction of Functions in the Command Window**

**1. Inline function**

f1 =jnline(expr)

f2=inline(expr, arg1, arg2,. . . )

**Example**

>> f=inline('x^2+2\*x\*y')

f =

Inline function:

f(x,y) = x^2+2\*x\*y

>> v1=f(1, 2.2)

v1 =

5.4000

**2. Function\_handle (@) )**

handle=@(arglist) anonymous\_function

**Example**

>> ff=@(x,y) x^2+x/y

ff =

@(x,y)x^2+x/y

>> ff(1.1,2)

ans =

1.7600

>> ff2=@(x,y) x.^2+x./y

ff2 =

@(x,y)x.^2+x./y

>> a=[1,2]

a =

1 2

>> c=[3,4]

c =

3 4

>> ff2(a,c)

ans =

1.3333 4.5000

To compute elementwise in array use

POWER with (.^) , DIVISION with (./) and PTRODUCT with (.\*)

**3. Function using Symbols**

>> syms x y

>> ff(x,y)=x^2+x/y

ff(x, y) =

x^2 + x/y

>> ff(1.2, 2) % returns result in fracttion

ans =

51/25

>> ffval=eval(ff(1.2,2)) % eval( ) is used to convert fraction to decimal form

ffval =

2.04

**m-Files**

There are two types of programs (m-files) in MATLAB: Functions and Scripts.

**User Defined Functions**

To be created in function window.

To open: New 🡪 Function

To save: Save 🡪 enter file name and save

To run: Type function name in command window

To edit: Open 🡪 select the file from the list and edit. save again

**function t=FF(a)**

**t=7\*cos(a)+2\*a-1;**

**end**

The function which is created as an m-file and saved as FF.m.

To use the above function type in command window.

>> t=FF(2)

t =

0.0870

**Scripts**

Scripts provide a set of MATLAB commands, comments, values, plotting commands, and so on.

A scripts that has been created and saved is executed by typing the file name at the MATLAB prompt in the command window or using save and run from the Debug menu.

To open: New 🡪 Script

To save: Save 🡪 enter file name and save

To run: Type script name in command window

To edit: Open 🡪 select the file from the list and edit.Save again

% Script TestProgm

%Values of f(x) for different values of x

x= x0;

disp(' n xn f(xn) ')

for i=1:nmax

fx=f(x);

n=i-1;

disp([n,x,fx])

x=x+h;

end

Save the script as TestProg.m

To execute the script from Command Window, type following commands:

>> clear

>> x0=1;

>> h=0.5;

>> f=inline('7\*cos(x)+2\*x-1')

f =

Inline function:

f(x) = 7\*cos(x)+2\*x-1

>> nmax=5

nmax =

5

>> TestProg % Type the Script name

**Output**

n xn f(xn)

0 1.0000 4.7821

1.0000 1.5000 2.4952

2.0000 2.0000 0.0870

3.0000 2.5000 -1.6080

4.0000 3.0000 -1.9299

**Programming in MATLAB**

Normally in a programming language, it is necessary to specify type of variable used in the program (integer, real, complex, and so on). MATLAB treats all variables as matrices (whatever dimension is needed) and perform the necessary calculations.

To repeat similar calculations several times for and while loops are used. Syntax are

***For Loops***

***for*** i = 1 : k

commands . . .

***end***

***While Loops***

***while*** statement == true

commands . . .

***end***

Conditional execution can be indicated by ***if*** statement.

***if*** expression

commands . . .

***end***

If there are more alternatives, the form is

***if*** expression1

commands . . .

***elseif*** expression2

commands . . .

***elseif*** …

. . .

***end***

***Break*** and ***Error***

The ***break*** command causes MATLAB to jump outside the loop in which it occurs.

The ***error*** command abort function execution , displays a character string in the command Window, and returns control to the keyboard.

**Example 1**: Write MATLAB codes for calculating n! using for loop.

>> clear

>> n=input('please input number = ');

fact=1;

i=1;

for i=1:n

fact=fact\*i;

i=i+1;

end

disp(fact)

please input number = 5

120

indow

**Example 2** Write MATLAB codes in command for calculating the sum of the following series using for loop:

>> clear

>> sum1=0;

for j=2:2:100

sum1=sum1+j;

j=j+2;

end

disp(['sum is =',num2str(sum1)])

sum is =2550

**Exercise A1**

1. Calculate the Celsius temperature by using the following relation between and

where the range of Fahrenheit temperatures from 10 to 200 with interval 5.

Then print these values with the proper headings by using the fprintf command in MATLAB.

2. Write MATLAB codes for calculating the sum of the following series

1+3+5+………+n

using for loop with different *n* values.

3. Define the function in MATLAB. Generate a linearly spaced vector of length n from [0:pi/2:3pi] using builtin MATLAB command “linspace”. Then plot the function with red .

**Computer Algebra System (CAS) with MATLAB**

MATLAB contributes to solve in numerous common mathematical areas such as calculus, linear algebra, algebraic equations differential equations and so on. MATLAB also provides symbolic calculations and manipulations symbolic mathematical expression~~.~~ For this purpose we need to define the symbols and it can be done by using the MatLab default command ‘syms ’. For example if we want to define the expression in MATLAB using symbol we have to define it in the following manner

syms x

x^2-5\*x+6

Use of symbolic expressions are explained through the following examples

**Example 1**: Solve the equation , whether *a* is a constant.

MATLAB codes are

>>syms x a

>> Solution=solve(x^2-5\*x+a==0,x) % solve(eqn, var) is to solve an equation

Solution=

5/2 - (25 - 4\*a)^(1/2)/2

(25 - 4\*a)^(1/2)/2 + 5/2

**Example 2:** Consider the solution of the nonlinear system of equation

MATLAB commands:

>>syms x y

>>[x2, y2] = solve(y + x^2 == 1, x - y == 10)

x2 =

- (3\*5^(1/2))/2 - 1/2

(3\*5^(1/2))/2 - 1/2

y2 =

- (3\*5^(1/2))/2 - 21/2

(3\*5^(1/2))/2 - 21/2

**Example 3**: Find the first and second derivative of .

**MATLAB inputs**

>> clear

>> syms x

>> f(x)=x^2\*sin(2\*x);

>> d1=diff(f) % diff(X,n) is used for derivatve of X w.r.t *x n-*times.

% if n not supplied it takes default value 1

d1(x) =

2\*x\*sin(2\*x) + 2\*x^2\*cos(2\*x)

>> d2=diff(f,2)

d2(x) =

2\*sin(2\*x) + 8\*x\*cos(2\*x) - 4\*x^2\*sin(2\*x)

Example 4:Evaluate .

>> clear

>> syms x

>> f(x)=(1+x^2)/sqrt(1+x)

f(x) =

(x^2 + 1)/(x + 1)^(1/2)

>> I=int(f) % int(f) integrate *f*(*x*) w.r.t. *x*

I(x) =

(2\*(x + 1)^(1/2)\*(3\*(x + 1)^2 - 10\*x + 20))/15

>> IS=simplify(I)

IS(x) =

(2\*(x + 1)^(1/2)\*(3\*x^2 - 4\*x + 23))/15

**Example 5:** Find the general solution of differential equation

>> clear

>> syms y(x)

>> Dy=diff(y);

>> D2y=diff(y,2);

>> GS=dsolve(D2y+3\*Dy+2\*y==exp(-x)) % dsolve(eqn) solves D.E.

GS =

x\*exp(-x) - exp(-x) + C3\*exp(-x) + C4\*exp(-2\*x)

**Exercise 1**

Use MATLAB to find the exact solution of the following Problems.

1. Solve the following equations;

2. Solve the following system of equations.

(a)

(b)

(c)

3. Find the first and second derivatives of the following functions:

4. Evaluate the following integrals:

5. Find the general solution of the following differential equations.

Representation of Numbers

**Decimal places (d.p.)**: The number of digits counted after the decimal marker.

**Significant figures (s.f.):** All digits including zero are counted from the first non-zero digit.

**Rounding:** The last retained digit is corrected up if the succeeding digit is greater than or equal to 5, otherwise chopped off.

**For example**

34607 = 30000, correct to 1 s.f. 0.004037 = 0.004, correct to 1 s.f./3 d.p.

= 35000, correct to 2 s.f. = 0.0040, correct to 2 s.f./4 d.p.

= 34600, correct to 3 s.f. = 0.00404, correct to 3 s.f./5 d.p.

= 34610, correct to 4 s.f.

2.30498 = 2.3, correct to 1 d.p./2 s.f

= 2.30, correct to 2 d.p./3 s.f.

= 2.305, correct to 3 d.p./4 s.f.

= 2.3050, correct to 4 d.p./5 s.f.

**Error Measurement**

Numerical calculations can be in error due to the use of approximate values in the calculation. The following definitions are used in measuring the errors.

Absolute error =⏐True value − Approximate value⏐

Relative error = 

Percentage of error = Relative error × 100 %

**Note that** in absence of true value an approximate relative error,  can be estimated by using the relation



**Rounding Error**

If 2.326 is a number rounded to 3 d.p., the true value α is

or

Thus the maximum absolute error is 

**If a number is rounded to *n* decimal places, the maximum absolute error is .**

Consider two numbers 235.3 and 0.003267 which are rounded to 4 s.f.. The errors can be estimated as follows

For 235.3, the relative error is 

For 0.003267. the relative error is 

In general, **if a number is rounded to *n* significant figures the maximum relative error is .**

The table below shows various errors corresponding to the given true and approximate values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True value | Approximate value | Absolute error | Relative error | Percentage error |
| 3.141592 | 3.142 | 0.00041 | 0.00013 | 0.013 |
| 100000 | 99950 | 50 | 0.0005 | 0.05 |
| 0.00025 | 0.0003 | 0.00005 | 0.2 | 20 |

**Example 1.1**

1. The solution of the quadratic equation is.

Write a MATLAB script to solve the quadratic equation with variable coefficients which gives variable significant digits.

1. Using your script solve the equation  correct to ten significant figures.

**Solution:**

(a)

% Script QuadEq for solution

clear all

disp('Solution of Quadratic Equation')

% Equation : ax^2+bx+c=0

% Roots: x1= -b+sqrt(b^2-4ac)/(2a)

% x2= -b-sqrt(b^2-4ac)/(2a)

abc=input('Supply a,b,c as [a,b,c]=');

a=abc(1); b=abc(2); c=abc(3);

x1=(-b+sqrt(b^2-4\*a\*c))/(2\*a);

x2=(-b-sqrt(b^2-4\*a\*c))/(2\*a);

Roots=[x1,x2]

disp('Roots to n significant digits')

n=input('Value of n = ');

Roots\_n=vpa([x1; x2],n)

**(b) Output**

>> clear

Solution of Quadratic Equation

Supply a,b,c as [a,b,c]=[1 -9876 1]

Roots =

1.0e+03 \*

9.8760 0.0000

Roots to n significant digits

Value of n = 10

Roots\_n =

9875.999899

0.0001012555704

**Exercise 2**

1 State the maximum errors in the following numbers which are correct to the significant figures given.

(a) 42.73, (b) 0.00346, (c) 7.00245, (d) 90, (e) .

2. Students collected the following set of data to find the constant . Use the relation where *A*  is the area in metre-square and *r* is radius in meter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Radius (r) | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| Area (A) | 0.03 | 0.13 | 0.28 | 0.5 | 0.79 |

Find out the approximate value of at each observations, also estimate the corresponding percentage errors with respect to true value of

Use MATLAB for the calculation.

3. Find the roots of the following quadratic equations using MATLAB scripts giving your results correct to 10 significant digits.