%plotting graph in MATLAB

x = 0:5:100;

y = x;

plot(x, y);

% subplot Many graph on same window

clc;

clear;

x = 0:5:100;

y1 = x;

y2=3\*x;

subplot(2,2,1);

plot(x, y1);

xlabel('x');

ylabel('y1');

legend('y1');

subplot(2,2,2);

plot(x,y2);

xlabel('x');

ylabel('y2');

legend('y2');

x = 0:0.01:10;

y = sin(x);

plot(x, y);

xlabel('x');

ylabel('Sin');

title('Sin(x) Graph');

grid on;

%Drawing Multiple Functions on the Same Graph

x = [0 : 0.01: 10];

y = sin(x);

g = cos(x);

plot(x, y, '\*', x, g, '.-');

legend('Sin(x)', 'Cos(x)');

% plot of x square

x = 1:1:10;

y = x.^2;

plot(x, y);

%f(x) = 3x4 + 2x3+ 7x2 + 2x + 9 and

% g(x) = 5x3 + 9x + 2 on same graph

x = -10 : 1: 10;

y = 3\*x.^4 + 2 \* x.^3 + 7 \* x.^2 + 2 \* x + 9;

g = 5 \* x.^3 + 9 \* x + 2;

plot(x, y, x, g);

%Create Bar Graph

y = [50, 85, 110, 135, 155, 180, 210, 230, 255, 280, 300];

bar(y);

x = 1900:10:2000;

y = [75 91 105 123.5 131 150 179 203 226 249 281.5];

bar(x,y)

% discreate plot

t=-10:1:10;

y= t;

stem(t,y);

% input from user

clear;

t=0:.001:.1;

f=input('enter frequency');

x = sin(2\*pi\*f\*t);

plot(t,x);

a = 10;

% check the condition using if statement

if a < 20

% if condition is true then print the following

fprintf('a is less than 20\n' );

end

fprintf('value of a is : %d\n', a);

a = 100;

% check the boolean condition

if a < 20

% if condition is true then print the following

fprintf('a is less than 20\n' );

else

% if condition is false then print the following

fprintf('a is not less than 20\n' );

end

fprintf('value of a is : %d\n', a);

Tutorial 1

>> 1+2\*3

ans = 7

>> x = 1+2\*3

x =7

Table 1.1: Basic arithmetic operators Symbol Operation Example

+ Addition 2 + 3

− Subtraction 2 − 3

∗ Multiplication 2 ∗ 3

/ Division 2/3

**Creating MATLAB variables**

variable name = a value (or an expression)

>> x = 1+2\*3

Overwriting variable: - Once a variable has been created, it can be reassigned. In addition, if you do not wish to see the intermediate results, you can suppress the numerical output by putting a semicolon (;) at the end of the line. Then the sequence of commands looks like this:

>> t = 5;

>> t = t+1

t = 6

Making corrections To make corrections, we can, of course retype the expressions. But if the expression is lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key ↑. When the command is displayed at the command prompt, it can be modified if needed and executed.

**Controlling the hierarchy of operations or precedence**

>> (1+2)\*3

ans = 9

The contents of the workspace persist between the executions of separate commands. Therefore, it is possible for the results of one problem to have an effect on the next one. To avoid this possibility, it is a good idea to issue a clear command at the start of each new independent calculation.

>> clear

The command clear or clear all removes all variables from the workspace. This frees up system memory. In order to display a list of the variables currently in the memory, type

>> who

while, whos will give more details which include size, space allocation, and class of the

Keeping track of your work session It is possible to keep track of everything done during a MATLAB session with the diary command.

>> diary

or give a name to a created file,

>> diary FileName where FileName could be any arbitrary name you choose.

Entering multiple statements per line It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to enter more than one statement at once. Commas (,) allow multiple statements per line without suppressing output.

>> a=7; b=cos(a), c=cosh(a)

b = 0.6570

c = 548.3170

• To clear the Command Window, type clc

* >> help Command
* On the other hand, the command lookfor inverse will produce detailed information, which includes the function of interest, inv.
* >> lookfor inverse

Tutorial 2

Mathematical functions:

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions. Typing help elfun and help specfun calls up full lists of elementary and special functions respectively. There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as sin(x), cos(x), tan(x), e x , ln(x), are evaluated by the functions sin, cos, tan, exp, and log respectively in

: Elementary functions

cos(x) Cosine

abs(x) Absolute value

sin(x) Sine

sign(x) Signum function

tan(x) Tangent

max(x) Maximum value acos(x)

min(x) Minimum value

sqrt(x) Square root

rem(x) Remainder after division

log(x) Natural logarithm

angle(x) Phase angle

log10(x) Common logarithm

conj(x) Complex conjugate

In addition to the elementary functions, MATLAB includes a number of predefined constant values.

pi The π number, π = 3.14159 . . . i,j The imaginary unit i, √ −1

Inf The infinity, ∞

NaN Not a number

Examples

The value of the expression y = e −a sin(x) + 10√y, for a = 5, x = 2, and y = 8 is \_\_\_\_\_\_

>> a = 5; x = 2; y = 8;

>> y = exp(-a)\*sin(x)+10\*sqrt(y)

y =

28.2904

>> log(142)

ans =

4.9558

>> log10(142)

ans =

2.1523

>> sin(pi/4)

ans =

0.7071

Basic plotting

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning mathematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of xcoordinates, x = (x1, . . . , xN ), and a vector of y-coordinates, y = (y1, . . . , yN ), locate the points (xi , yi), with i = 1, 2, . . . , n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length.

The MATLAB command to plot a graph is plot(x,y). The vectors x = (1, 2, 3, 4, 5, 6)and y = (3, −1, 2, 4, 5, 1)

>> x = [1 2 3 4 5 6];

>> y = [3 -1 2 4 5 1];

>> plot(x,y)

plot(x,y) produces a graph of y versus x.

For example, to plot the function sin (x) on the interval [0, 2π], we first create a vector of x values ranging from 0 to 2π, then compute the sine of these values, and finally plot the result.

>> x = 0:pi/100:2\*pi;

>> y = sin(x);

>> plot(x,y)

Notes: • 0:pi/100:2\*pi yields a vector that – starts at 0, – takes steps (or increments) of π/100, – stops when 2π is reached.

• If you omit the increment, MATLAB automatically increments by 1.

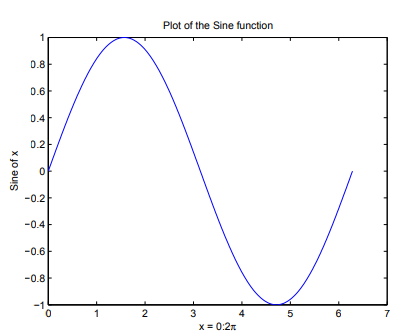
**Adding titles, axis labels, and annotations**

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add x- and y-axis labels. The character \pi creates the symbol *π*.

>> xlabel(’x = 0:2\pi’)

>> ylabel(’Sine of x’)

>> title(’Plot of the Sine function’)

****

The color of a single curve is, by default, blue, but other colors are possible. The desired color is indicated by a third argument. For example, red is selected by plot(x,y,’r’). Note the single quotes, ’ ’, around r.

Multiple (x, y) pairs arguments create multiple graphs with a single call to plot. For example, these statements plot three related functions of x: y1 = 2 cos(x), y2 = cos(x), and y3 = 0.5 ∗ cos(x), in the interval 0 ≤ x ≤ 2π.

Multiple data sets in one plot

>> x = 0:pi/100:2\*pi;

>> y1 = 2\*cos(x);

>> y2 = cos(x);

>> y3 = 0.5\*cos(x);

>> plot(x,y1,'--',x,y2,'-',x,y3,':')

>> xlabel('0 \leq x \leq 2\pi')

>> ylabel('Cosine functions')

>> legend('2\*cos(x)','cos(x)','0.5\*cos(x)')

>> title('Typical example of multiple plots')

>> axis([0 2\*pi -3 3])

**MATRIX**

Matrices are the basic elements of the MATLAB environment. A matrix is a two-dimensional array consisting of *m* rows and *n* columns.

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation.

Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create

vectors and matrices in MATLAB. As discussed earlier, an array of dimension 1*£n* is called

a *row* vector, whereas an array of dimension *m£* 1 is called a *column* vector. The elements

of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by

commas. For example, to enter a row vector, v, type

>> v = [1 4 7 10 13]

v =

1 4 7 10 13

Column vectors are created in a similar way, however, semicolon (;) must separate the

components of a column vector,

>> w = [1;4;7;10;13]

w =

1

4

7

10

13

On the other hand, a *row* vector is converted to a *column* vector using the *transpose* operator.

The *transpose* operation is denoted by an apostrophe or a single quote (').

>> w = v'

w =

1

4

7

10

13

Entering a matrix

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

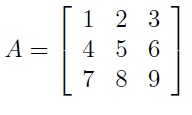
begin with a square bracket, [

separate elements in a row with spaces or commas (,)

use a semicolon (;) to separate rows

end the matrix with another square bracket, ].

Here is a typical example. To enter a matrix A, such as,



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

Note that the use of semicolons (;) here is di®erent from their use mentioned earlier to

suppress output or to write multiple commands in a single line. Once we have entered the matrix, it is automatically stored and remembered in the *Workspace*.

Once we have entered the matrix, it is automatically stored and remembered in the

*Workspace*. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write,

>> A(2,1)

ans =

4

Matrix indexing

A(1,3)=3.

>> A(3,3) = 0

Colon operator

The colon operator will prove very useful and understanding how it works is the key to

e±cient and convenient usage of MATLAB. It occurs in several di®erent forms.

Often we must deal with matrices or vectors that are too large to enter one ele-

ment at a time. For example, suppose we want to enter a vector *x* consisting of points

(0*,0.1,0.2……5)* We can use the command

>> x = 0:0.1:5;

Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: linspace. It

is similar to the colon operator (:), but gives direct control over the number of points. For

example,

>> theta = linspace(0,2\*pi,101)

divides the interval [0*;* 2*π ]*into 100 equal subintervals, then creating a vector of 101 elements.

Colon operator in a matrix

The colon operator can also be used to pick out a certain row or column. For example, the

statement A(m:n,k:l) specifies rows *m* to *n* and column *k* to *l*. Subscript expressions refer

to portions of a matrix. For example,

>> A(2,:)

ans =

4 5 6

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

>> A(:,2:3)

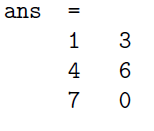
ans =



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

A row or a column of a matrix can be deleted by setting it to a *null* vector, [ ].

>> A(:,2)=[]



Creating a sub-matrix

To extract a *submatrix* B consisting of rows 2 and 3 and columns 1 and 2 of the matrix A, do the following

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

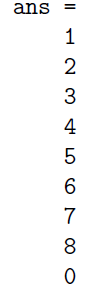
B =

C:\Users\vimal123\Desktop\MATLAB\sub_matrix.png

It is important to note that the *colon operator* (:) stands for *all columns* or *all rows*. To

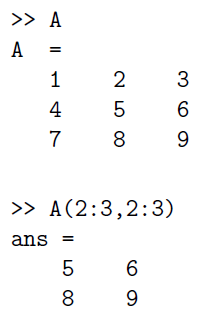
create a vector version of matrix A, do the following

>> A(:)

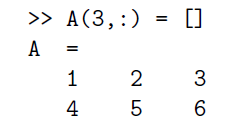


* A(end,:) picks out the last row of A.

Some Examples-



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



Dimension

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

>> size(A)

ans =

3 3

Transposing a matrix

The *transpose* operation is denoted by an apostrophe or a single quote (').

>> A'

ans =

1 4 7

2 5 8

3 6 0

By using linear algebra notation, the transpose of *m £ n* real matrix A is the *n £ m* matrix

that results from interchanging the rows and columns of A. The transpose matrix is denoted

*AT* .

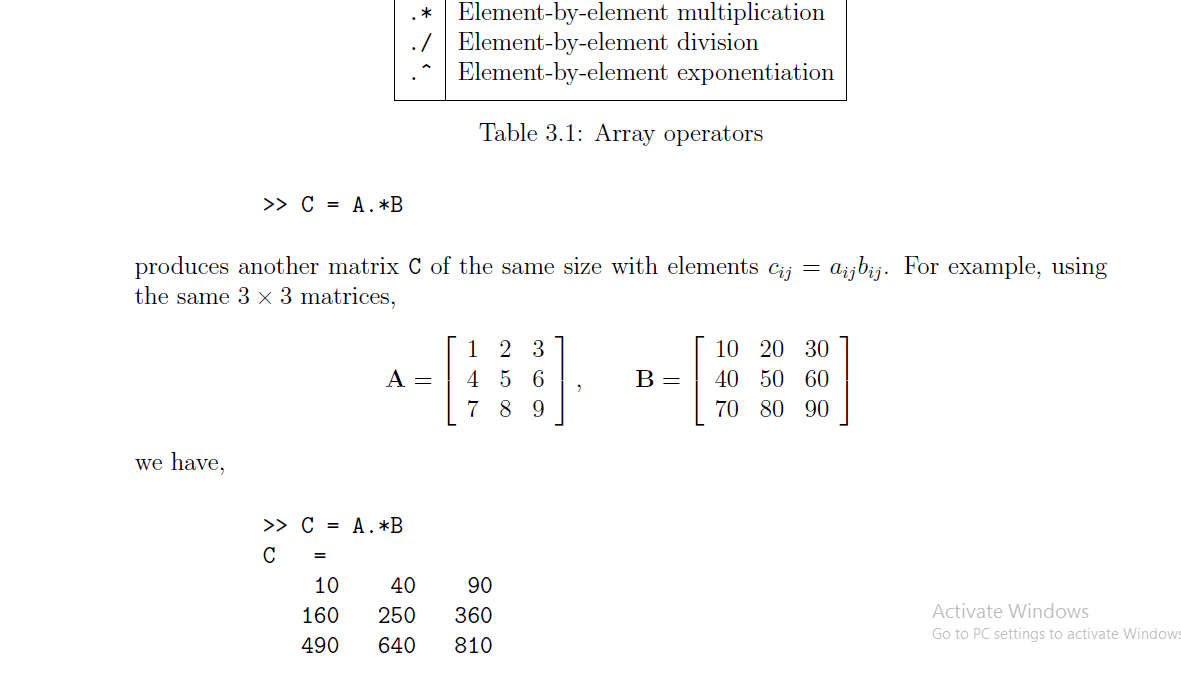
Array arithmetic operations

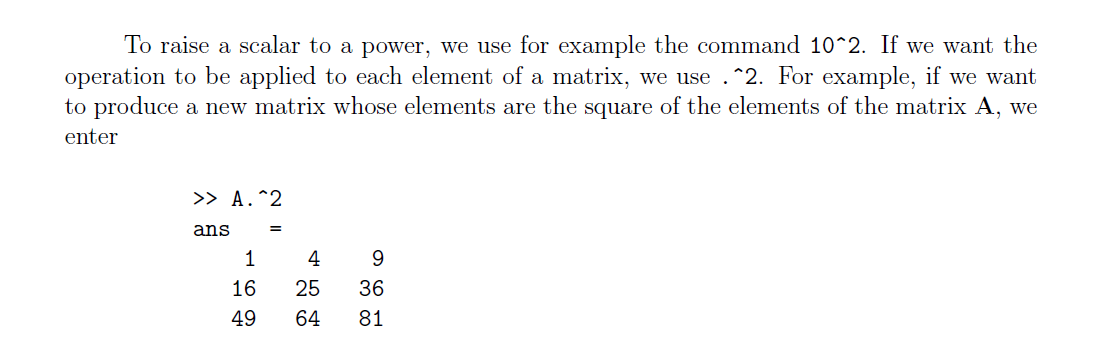
array arithmetic operations or *array operations* for short, are done

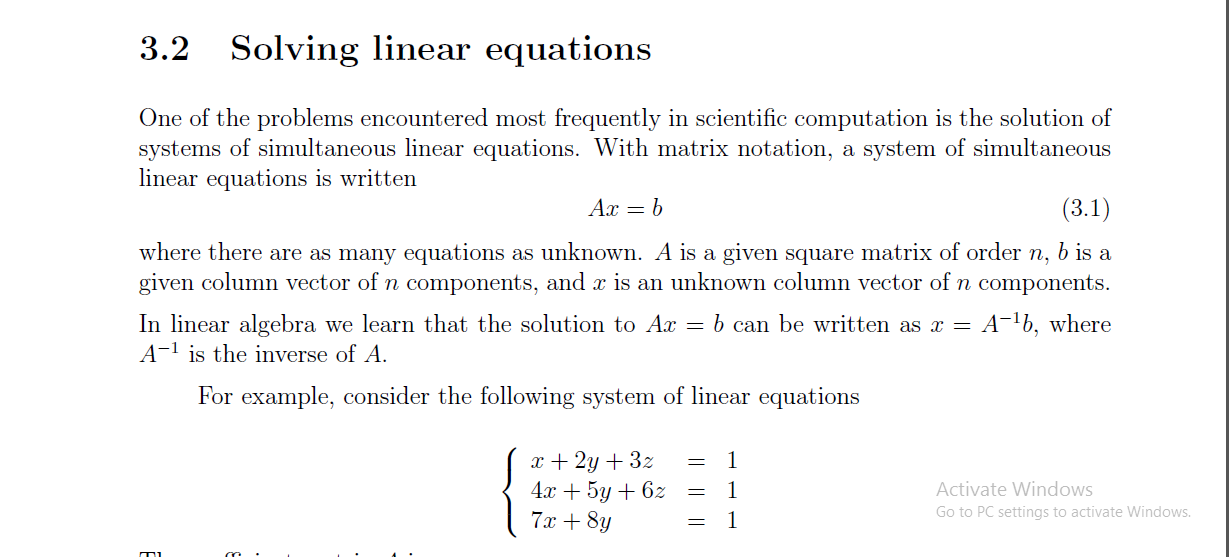
*element-by-element*. The period character, ., distinguishes the array operations from the

matrix operations. However, since the matrix and array operations are the same for addition

(+) and subtraction (*¡*), the character pairs (*:*+) and (*:¡*) are not used.







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

>> b = [1; 1; 1];

>> x = inv(A)\*b

x =

-1.0000

1.0000

-0.0000

Introduction to programming in

MATLAB

So far in these lab sessions, all the commands were executed in the Command Window.

The problem is that the commands entered in the Command Window cannot be saved

and executed again for several times. Therefore, a di®erent way of executing repeatedly

commands with MATLAB is:

1. to *create* a ¯le with a list of commands,

2. *save* the ¯le, and

3. *run* the ¯le.

¯les or *scripts* for short.

This section covers the following topics:

*²* M-File Scripts

*²* M-File Functions

A *script ¯le* is an external ¯le that contains a sequence of MATLAB statements. Script

¯les have a ¯lename extension .m and are often called M-¯les. M-¯les can be *scripts* that

simply execute a series of MATLAB statements, or they can be *functions* that can accept

arguments and can produce one or more outputs.

Control flow and operators

if expression

statements

end

discr = b\*b - 4\*a\*c;

if discr < 0

disp('Warning: discriminant is negative, roots are

imaginary');

end

if ... else ... end

discr = b\*b - 4\*a\*c;

if discr < 0

disp('Warning: discriminant is negative, roots are

imaginary');

else

disp('Roots are real, but may be repeated')

end

if ... elseif ... else ... end

discr = b\*b - 4\*a\*c;

if discr < 0

disp('Warning: discriminant is negative, roots are

imaginary');

elseif discr == 0

disp('Discriminant is zero, roots are repeated')

else

disp('Roots are real')

end

The ``for...end'' loop

for ii=1:5

x=ii\*ii

end

The ``while...end'' loop

x = 1

while x <= 10

x = 3\*x

end

x = 0:0.5:10;

y = x>^2;

plot(x, y)%plotting x and y  
xlabel(X)%naming x axis as x  
ylabel(Y)%naming y axis as y  
title('Graph of y=x^2')%Title of the graph