**Getting Started in Matlab**

**What Is MATLAB?**

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

**MATLAB is an interactive system whose basic data element is an array** that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for *MATrix LABoratory*.

MATLAB features a family of add-on application-specific solutions called ***toolboxes***.Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology.

Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

**The MATLAB system consists of five main parts:**

* **Development Environment** The set of tools and facilities that help you develop MATLAB functions and files. Many of these tools are graphical user interfaces. The MATLAB desktop: Command Window; command history; an editor and debugger; browsers for help; the workspace; and files and the search path.
* **The MATLAB Mathematical Function Library** A vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.
* **The MATLAB Language** This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create large and complex application programs.
* **Graphs** MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.
* **The MATLAB Application Program Interface (API)** MATLAB allows you to write C and Fortran programs that interact with MATLAB. The MATLAB Data API provides a way for applications running outside of MATLAB to work with MATLAB data through a MATLAB-neutral interface.

**The desktop includes these panels:**

* Current Folder – Access your files.
* Command Window – Enter commands at the command line, indicated by the prompt (>>).
* Workspace – Explore data that you create or import from files.
* Command History – View or rerun commands that you entered at the command line.
* Help (Documentation) – You can find help on almost everything on MATLAB functions and tools from here. You may type the name of the function or short description of your query.
* When you right click on each of the panels above or click on  you will see more options.
* Details – Gives you details about the file you are working on.

**Some Useful Initial Tips**

* If you do something time consuming (perhaps an error) and want to stop Matlab type Ctrl + C
* Commands are case-sensitive so typing sqrt(4) will return the square root of 4 but Sqrt(4) will just return an error.
* Anything preceded by the % symbol is not executed by Matlab – useful for comments.
* Pressing the up and down arrows will recall previous commands which can be edited. You can also copy and paste commands from the Command History.
* To continue a line, type “. . .”
* 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.
* You can clear the screen by typing clc.

**Using MATLAB Help**

* Search for and view documentation and demos for all your MathWorks products. The Help browser is an HTML viewer integrated into the MATLAB desktop.
* The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information. These are the key features: Contents: View the titles and tables of contents of the documentation. Search: Look for specific words in the documentation. Demos: View and run demonstrations for your MathWorks products.

**MATLAB - Call and Response Interface**

**Working on the Command Line**

Example

1 + 2 + 3

ans =

7

**Working with Variables**

Assignment operator is =

x = 5

Overwriting a variable

t = 5

t = t+1

t =

6

**Managing Workspace**

The contents of the workspace persist between the executions of separate commands

who

whos

clear n

clear all

**The Colon Operator**

Generating a sequence of numbers using colon (:)

A = 1:5

A = -2.5:2.5 % Increment is 1 by default

To create a sequence with a step value the syntax is first:step:last

A = 10:5:50 % increment of 5

A = 3:0.2:4 % increment of 0.2

c = 10:-0.5:4 %decrement of 0.5

p = 0:pi/4:pi

**Linspace**

To generate linearly spaced vector, use linspace. Syntax follows as linspace(a,b,n). This is useful when we want to divide an interval into a number of subintervals of the same length

A = linspace(10,50) % this will generate 100 points between 10 and 50 inclusive

A = linspace(10,50,9) % this is same as 10:5:50

A = linspace(10,50,10)

**Vectors and Matrices**

Everything in MATLAB is stored as a matrix. A matrix is a two-dimensional array. A row vector is 1 by n matrix and a column vector is a n by 1 matrix c = 5; is a one by one matrix

There are several ways to create matrices :

* Enter an explicit list of elements.
* Load matrices from external data files.
* Generate matrices using built-in functions.
* Create matrices with your own functions in M-files.

The few basic conventions for entering matrices is :

* Separate the elements of a row with blanks or commas.
* Use a semicolon, ; , to indicate the end of each row.
* Surround the entire list of elements with square brackets, **[]**

A = [5 2 ; 3 1]

A =

5 2

3 1

A = [1:4; 5:8; 9:12]

B = magic(4) % using in-built function magic

B = rand(2,3) % 2 by 3 matrix of uniformly distributed random numbers between 0 and 1

B = 2\*rand(2,3) % 2 by 3 matrix of uniformly distributed random numbers between 0 and 2

C = randi([1 5],2,3) % uniformly distributed integers between 1 and 5

D = randn(4,4) % 4x4 normally dist. random no.s (mean = 0, std dev. = 1 and var. = 1)

***See Appendix for more Matrix creating functions***

**Operators**

In MATLAB two types of operations can be carried out on matrices.

* element-wise operations
* multi-dimensional array operation The operators are:

| **Operator** | **Operation** |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Matrix multiplication |
| ^ | Matrix power (exponentiation) |
| / | Right matrix divide (Normal) |
| \ | Left matrix divide (a/b = b\a for scalar). For matrix, left divide is a bit complicated |
| ' | Transpose of Matrix |
| .^ | Element-wise Array power |
| .\* | Element-wise Array multiplication |
| ./ | Element-wise division |
| .\ | Element-wise left division |

MATLAB follows a strict order of precedence when performing arithmetic operations

| **Precedence** | **Operator** |
| --- | --- |
| 0 | Parentheses () |
| 1 | Exponential ^ |
| 2 | Unary +, Unary - and logical negation ~ |
| 3 | Multiplication \* and .\*. Divison / and ./ |
| 4 | Addition + and Subtraction - |

TRY this for an example:

A = magic(5);

B = A \* 5 +12;

C = A \* (5+12);

***Exercise - Question 1(i) to (vi)***

**Relational and Logical Operators**

**Relational Operators** perform element-by-element comparisons between two arrays. The output is a logical array where 1 represents TRUE and 0 represents FALSE

| **Operator** | **Operation** |
| --- | --- |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| '==` | Equal to |
| ~= | Not equal to |

Examples:

X = 5;

X >= [1 2 3; 4 5 6; 7 8 10]

% is same as

X = 5\*ones(3,3);

X >= [1 2 3; 4 5 6; 7 8 10]

**Logical Operators** perform element-wise logical operations on their inputs to produce a like-sized output array. They return elements set to **0**, representing logical False, and anything nonzero representing logical True i.e.,**1**.

A = [0 3 2 0 5];

B = [1 2 0 0 6];

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & | Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements | A & B = 01001 |
| | | Returns 1 for every element location that is true (nonzero) in either one or the other, or both arrays, and 0 for all other elements | A |
| ~ | Complements each element of the input array, A | ~A = 10010 |
| Xor | Returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements | xor(A,B)=10100 |

There are other logical operators that work with only scalars values. && and ||. These are called short-circuit logical operators. Try:

A = [0 3 2 0 5];

B = [1 2 0 0 6];

A && B % will return an error

any(A) && any(B) % will return True

x = 5; y = 12;

x & y % will return True (1)

x && y % will also return True (1)

**Matrix manipulation**

Sum commonly used functions on matrices. See appendix for more functions

A = [3 1];

B = sum(A);

A = [5 2; 3 1];

sum(A) % sum of elements acrosss the columns of A

diag(A) % elements on the diagonal of matrix

sum(diag(A)) % sum of elements on the diagonal of matrix

***See Appendix for more functions***

***Exercise - Question 2, 3, 4, 5, 6 (Refer to appendix for functions)***

**Data Import and Export**

**Saving and Loading MAT-files**

To save variables int he workspace, use save <filename>. Omiting <filename>, MATLAB will save with name matlab.mat

save <filename>

save <filename> <var1> <var2> <var3>...<varn> % specify the variables you want to save

save <filename> '-ascii' '-append'

who % will list the variables you have in your workspace

whos % will display variables as well as their attributes

**Working with Spreadsheets**

With the help of xlswrite and xlsread we can write data into spreadsheet and read an existing one into matrix form.

% WRITE INTO XLS FILE

d = {'Time', 'Temp'; 12 98; 13 99; 14 97}; % this is a type of data cell array

xlswrite('tempdata.xls', d);

% READ FROM XLS FILE

ndata = xlsread('tempdata.xls'); % will read only numeric data from the file

[ndata, text] = xlsread('tempdata.xls');

The newer versions now take the writetable and readtable functions to read all types of data files.

% WRITE INTO CSV FILE

LastName = {'Sanchez';'Johnson';'Li';'Diaz';'Brown'};

Age = [38;43;38;40;49];

Smoker = 1;0;1;0;1];

T = table(LastName,Age,Smoker);

T.Smoker = categorical(T.Smoker);

writetable(T,'tempdata1.csv');

% READ FROM CSV FILE

T1 = readtable('tempdata2.csv'); T1

T1 = readtable('tempdata1.csv','readVariableNames',false); T1

**Missing Data**

NaN means **Not a Number**. Where MATLAB a non-numeric value, it will put NaN.

A = xlsread('testdata2.xls');

y = xlsread('age.xls');

nanmax(y)

nansum(y)

y(isnan(y)) = []; % will remove the NaN elements from y

Some more useful NaN functions are in the ***Appendix***.

**Accessing Elements of a Matrix**

**Subscripts**

The element in row i and column j of A is denoted as A(i,j)

A = [5 2; 3 1];

A(1,2)

The end keyword A(1,end)

**Linear indexing**

The matrices and arrays in MATLAB are stored as a single column of elements.

B = [2 6 9; 4 2 8; 3 5 1]

B =

2 6 9

4 2 8

3 5 1

is actually store in memory as the sequence 2,4,3,6,2,5,9,8,1. The element B(3,2) = 5 can also be identified as element 6 in the actual storage sequence. B(6) is referred as linear indexing.

B(3,2)

B(6) % will produce same result

B(3:2:7) % will produce elements on the right to left diagonal

ans =

3 2 9

**Logical Indexing**

The logical vectors created from logical and relational operations can be used to reference subarrays. This is referred to as Logical Indexing

B = [2 6 9; 4 2 8; 3 5 1];

B(B>4) % will give position of elements where greater than 4

B(B>4)=-5

B =

2 -5 -5

4 2 -5

3 -5 1

A = magic(4)

B = isprime(A)

B =

4×4 logical array

0 1 1 1

1 1 0 0

0 1 0 0

0 0 0 0

A(~B) = -1

A =

-1 2 3 13

5 11 -1 -1

-1 7 -1 -1

-1 -1 -1 -1

A(~isprime(A)) = -1

A =

-1 2 3 13

5 11 -1 -1

-1 7 -1 -1

-1 -1 -1 -1

**Colon and end keyword**

B = [2 6 9; 4 2 8; 3 5 1]

B(:,2)

ans =

6

2

5

B(:,end)

ans =

9

8

1

B(:,end-2)

ans =

6

2

5

B(1:2,end)

ans =

9

8

sum(B(:,end))

ans =

18

B(1:4,4) =-1

B =

2 6 9 -1

4 2 8 -1

3 5 1 -1

0 0 0 -1

***>>Questions 7 - 20***

**Graphs**

MATLAB provides a wide variety of techniques to display data graphically. Interactive tools enable you to manipulate graphs to achieve results that reveal the most information about your data. You can also annotate and print graphs for presentations, or export graphs to standard graphics formats for presentation in web browsers or other media.

**Creating and Editing graphs**

The type of graph you choose to create depends on the nature of your data and what you want to reveal about the data. MATLAB predefines many graph types, such as **line, bar, histogram,** and **pie graphs**. There are also 3-D graphs, such as surfaces, slice planes, and streamlines. There are two basic ways to create graphs in MATLAB:

* Creating graphs interactively.
* Use the command interface to enter commands in the Command Window or create plotting programs. You might find it useful to combine both approaches. For example, you might issue a plotting command to create a graph and then modify the graph using one of the interactive tools.

**Creating plot Interactively**

x = -1:0.1:1; % Define range of x

y = x.^3; % y for each value of x

plottools

**Creating plot using command interface**

figure;

x = -1:0.05:1; % Define range of x

y1 = x.^3;

plot(x,y1);

hold on

y2 = x.^4;

plot(x,y2);

axis([xlim ylim],[-1.5 1.5 -1.2 1.2]);

y2 = x.^5;

plot(x,y3,'b-o'); % blue pointers

xlabel('x')

ylabel('y')

legend;

legend('x.^3','x.^4','x.^5')

title('Various values of x and y')

hold off;

figure

**Creating plots with subplot function**

p1 = subplot(3,1,1);

plot(p1,x,y1);

p2 = subplot(3,1,2);

plot(p2,x,y2);

p3 = subplot(3,1,3);

plot(p3,x,y3,'b-o');

**Basic Program Components**

A method or procedure of computation which usually involves a series of steps is an algorithm. One of the key tasks of computational thinking is the development and implementation of an algorithm to solve the problems.

People from any background should be able to translate a “problem” into a code /program/algorithm. MATLAB provides a flexible environment for non-programmers to write small programs in order to solve particular problem.



**Expressions**

In MATLAB, an expression is a phrase which contains **Variables, Operators and Functions**. Unlike other programming languages, these expressions involve entire matrices.

You can find a full list of Operators and Elementary Operations-Functions on MathWorks documentation and website.

**Program Control Statements**

Program control is divided into these four categories:

* Loop Control -- for, while, continue, break
* Conditional Control -- if, switch
* Error Control -- try, catch
* Program Termination -- return

We will only look at if and for.

**Conditional Control -** ***if***

The basic syntax is:

if <expression>

<statements>

end

if <expression>

<statement set 1>

else

<statement set 2>

end

if <expression1>

<statement set 1>

elseif <expression2>

<statement set 2>

elseif <expression3>

<statement set 3>

.

.

.

elseif <expressionn>

<statement set n>

else

<statement set anything else>

end

***See script if\_ex.m***

**Loop Control –** ***for***

The basic syntax is:

for index = start:increment:end

statements

end

***See script for\_ex.m***

**Scripts and Functions**

**Scripts** - Scripts are a series of statements. These are stored as .m files. Open a script file.

**Functions** - Are another .m files in MATLAB, that we can create for something very specific.

% is used to enter comments in a script. Any line starting with % is not executable:

function f = fact(n) % Function definition line

% Compute a factorial value. - this is Heading line

% fact(n) returns the factorial of n - this is Help text

% usually denoted by n!

% Put simply, fact(n) is prod(1:n) - this is Comment

f = prod(1:n); Function body

***See function circles.m***

***>>Questions 21 - 26***