Matlab Primer #1 – A Fast Start

SECTION I: Matlab and the Image Processing Toolbox

Using this Primer effectively
If you are already an experienced user of Matlab, most of the material in these two primers will be familiar to you. You can either work rapidly through them or skip them entirely and proceed direct to Chapter 1 of the book where we specifically discuss the fundamentals of how *images* are defined and handled within the *Matlab + Image Processing Toolbox* environment.

If you are new to Matlab, working through the two primers will quickly get you up to speed on the basics. In this primer (part 1), we will aim to achieve the following:-

- Explain the advantages of using Matlab + the Image Processing Toolbox as a means for practical exploration of image processing.
- Advise on installation, useful sources of information and describe the basic Matlab interface.
- Describe and manage the Matlab workspace.
- How to write Matlab M files – scripts and functions.
- How to obtain basic graphics output.
- How to build Matlab expressions.

Why use Matlab ?

Readers unfamiliar with Matlab are likely to ask why we should prefer it to the large number of other scientific computation and visualisation languages and packages which are now available. There are a number of reasons which together make Matlab the right choice for this book.

1. **Quick-to-learn:** Matlab is easy to learn. As you will hopefully see, the learning curve for Matlab is shallow. In our experience, you can learn and start doing really useful things more quickly than with any other language.
2. **Easy-to-use:** Matlab is easy to use. It is ***both*** an interactive environment for high-performance technical computation, graphical display and animation (thus offering nearly all the benefits of specialist "packages") and ***also*** a high-level programming language (providing the necessary flexibility for the user to develop his own applications).
3. **Integrated environment:** Matlab and its associated toolboxes (collections of specialist functions) form a completely integrated environment. In other words, you can develop complex software applications which may involve complex calculation, graphical display and input and interface to external hardware and never have to move outside the Matlab environment.

4. **Platform independent:** Matlab code and programs run *exactly the same* on any computer. Thus you need have no concerns about the portability of Matlab code. Matlab is currently available for all main PC operating systems.
5. **Scientific standard:** Outside the world of computer science *per se* (part of whose mission should be to make programming easier), Matlab is emerging as *the standard* in scientific and technical computation. The huge growth in the number of Matlab users in Universities, research institutions/organisations and companies since its inception in 1984 is ample testimony.
6. **Rapid Development:** The ease of use and shallow learning curve for Matlab itself and functionality within Matlab that may be new to you for any given task make it an ideal rapid development (and prototyping) environment. In many organizations it is used both for initial prototyping and/or mathematical verification of implementations that will ultimately be developed in an application-level development language such as C/C++/Java/Ada.

The Matlab Image Processing Toolbox

The ***Image Processing Toolbox*** is exactly what its name suggests – a toolbox of specially written functions, seamlessly integrated with Matlab, to perform many of the low and high level procedures which are required time and again whenever tasks of image processing are undertaken. The reason for our choice of ***Matlab + Image Processing Toolbox*** may be highlighted by considering two extreme approaches to learning practical, digital image processing.

The *"low-level"* approach requires the user to write relatively detailed code to perform even basic tasks. This can certainly lead to understanding if one is tenacious enough to stay the course but, overall progress in learning the *concepts* is slowed and at least as much is learned about the intricacies of the programming language as the actual image processing. At the other "*high-level"* extreme, is the push-button or *package* solution. One solves a relatively complex problem (e.g. deblurring a noisy image or segmenting an object from its background) by drawing on all the expertise of an expert who has packaged the solution into a single click of a button. The result may be quite good but there is no real dialogue. No real understanding emerges and it is often not clear how to slightly modify or extend the procedure to a solve similar but subtly different problem.

As far as the practical implementation of image processing goes, ***Matlab + Image Processing Toolbox*** is the time-honoured middle way. You can *concentrate on the issues, avoid unnecessary programming tasks* but you can't get away with not understanding what you are doing. For the more advanced reader/user, there is the further motivation that all the solutions and corresponding code which you may develop are entirely portable to the broader Matlab environment, will run on any machine and, if desired, can be automatically converted to industry standard application development languages such as ANSI C (see Matlab Mex Compiler).

What Matlab will be covered ?

Many good and comprehensive books now exist on the Matlab language. Although fundamentally simple and easy to use, Matlab has many powerful features and extras which we do not and cannot hope to cover in this *fast-start* introduction. Here, our aim is to give you the *essentials* that you need to get up and running and sufficient material is covered here to enable you (if you are diligent) to tackle the examples and exercises with confidence. However, the motivated reader is certainly strongly encouraged to broaden their knowledge by referring to other books specifically on the Matlab language as well as the comprehensive documentation set which is provided when you purchase Matlab (either in paper or electronic form).

SECTION II: Matlab Basics

Summary of Section Contents
Practicalities. Understanding the workspace. The 20 most essential commands in Matlab. Arrays and indexing. Reading, writing and saving data. Data types. Operators. Flow control. M files – functions and scripts. Key concepts in Matlab. Further references.

Practicalities – Installing the Matlab Software

Individual Licenses: For an individual purchasing a standalone license, the installation of Matlab is very simple. You will receive a CD and a personal license password or key. Insert the CD-ROM into your computer drive to start the install wizard, enter your password/key and follow the remaining very simple instructions. Alternatively you may prefer to download the software from the Mathworks web-site. The procedure is similar - with the addition that you must invoke the install wizard yourself once the software is downloaded. Refer to the Mathworks web-site (www.mathworks.com) for details.

Multi-user Licenses: Research Institutions, companies, Universities and other organizations whose employees and students use Matlab often have multi-user licenses. This type of license works by installing Matlab on a networked server, enabling a number of users to simultaneously run Matlab. The system administrator is then the person usually responsible for installing the software. Although installation is usually trouble-free, Mathworks provide an installation guide which forms part of the on-line documentation provided when you purchase Matlab and to which you may refer.
If you have difficulties in installing Matlab, you should email support@mathworks.com outlining the nature of your problem.

Getting help: As a legitimate user of Matlab you are entitled to avail yourself of the technical support which is provided by *The Mathworks* (the developers and suppliers of Matlab). You should email support@mathworks.com and can expect a response to any meaningful technical query (from the very simple to the complex) about the use and scope of Matlab. Within reasonable limits, you may also receive good advice on how best to achieve a technical goal or aim using Matlab. Don't be afraid to use this service when you need it – it is their job to provide this service and they are enthusiasts.

Upgrades and bug-fixes: Matlab is constantly being developed and refined. Current license holders may receive new releases of the product depending on the license type and arrangement. Upgrades and bug-fixes to the software are usually available first from the web-site (www.mathworks.com). Refer to the instructions contained therein.

Launching Matlab

Let us assume that you have successfully installed Matlab. Launching Matlab is simple. You simply double click on the program icon and the Matlab command window appears or alternatively type `matlab` at the command prompt in unix/linux based operating systems.

The Matlab Command window - Finding your way around

When you start Matlab, you should get a window looking something looking like the frame below. The style and layout of the drop-down menus is largely standard. Figure 1.1 indicates in general terms the basic usage and the specific purpose of the utilities provided by the menus.

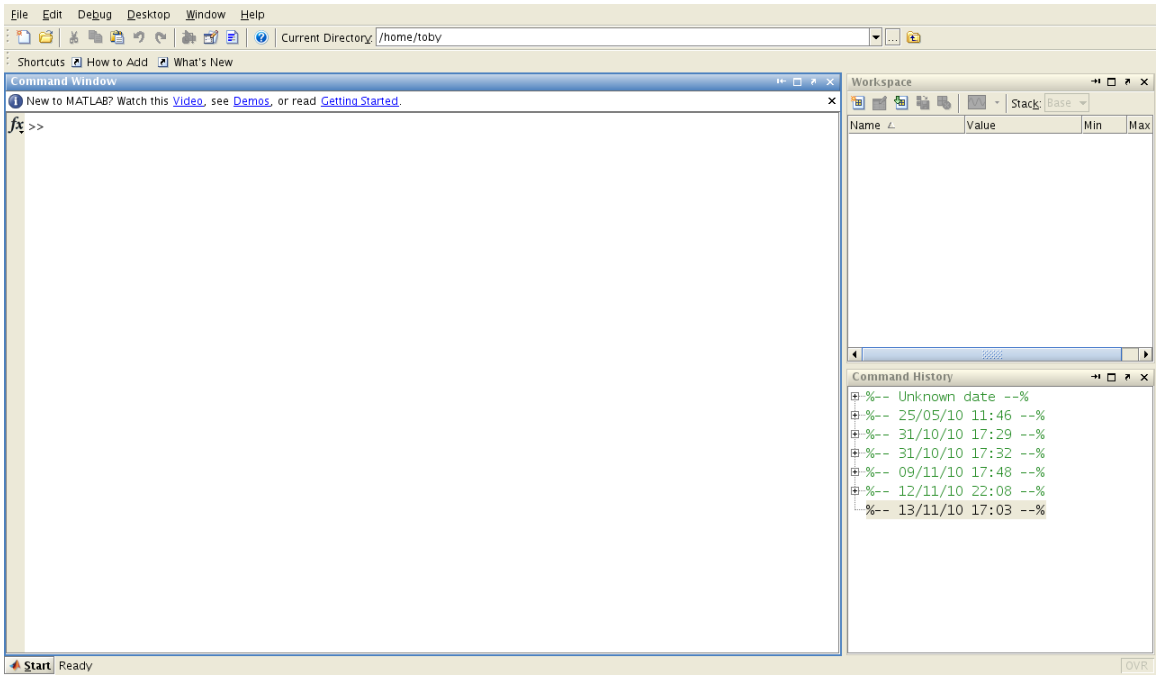


Figure 1.1 : The basic start-up work-space for Matlab

We assume that you have started Matlab – the command window is on the screen before you and Matlab is ready to receive input (as above). Before we start exploring Matlab, we first want to tie up two simple but important practical issues which often cause confusion and frustration for beginners. These concern :–

i) the *working directory* (i.e. the directory or folder in which you are running Matlab
ii) Matlab's *search path*.

The Working Directory

When you start Matlab it runs under a default directory/folder which is determined at the time of installation. Issue the command **pwd** ("present working directory") to find out what your default working directory/folder is –

```
>> pwd                          %Show present working directory
```

As you work through the following pages, you will want to save Matlab programs and data that you have created.. *It is best to save your files and data in your own, specially created and named folder.*

Change the current working directory to the one of your choice to which you have *write permission*. Consider the example below. Typing **pwd** at the prompt on our computer :–

```
>> pwd

ans =

e:\MATLAB6p1\work
```

Matlab responds by showing that the default working directory on my computer is e*:\MATLAB6p1\work*. To change this to my own chosen directory e*:/chris/my_matlabstuff*, we use the command **cd** (change directory).

```
>> cd e:/chris/my_matlabstuff

ans =

e:/chris/my_matlabstuff
```

Matlab confirms that the present working directory is e:/chris/my_matlabstuff. Please note that the directories in this example are of a *Microsoft Windows* operating system form and are more likely to be of the form */home/chris* or */usr/local/matlab* on *unix/linux* operating systems.

The Matlab Search Path

When you type a Matlab command at the prompt and press return, Matlab searches within a certain set of folders/directories to see if it can recognise the given command. The specific folders which are searched in this way are determined by the *search path*.
The default search path comprises a limited number of directories (or folders) in which Matlab has organized its own built-in M files and the present working directory.

Typing the Matlab function **path** will show the current Matlab search path on your computer. Try typing :–

```
>> path
```

Sooner or later, you will want Matlab to recognize commands (or more precisely functions[1]) that are not contained within the default search path. The search path can be displayed and modified in several ways. You can add directories to the existing search path using the command **addpath** with suitable arguments supplied. If, for example, you want Matlab to recognize files that you have written and which reside in the directory *c:/chris/my_matlabstuff* you should type :–

```
>> addpath('c:/chris/my_matlabstuff') ;
```

You can also achieve this by typing –

```
>>editpath
```

which brings up the appropriate path browser for you to select the paths you wish to add.

You can add new directories to the search path at any time during a Matlab session. It can occasionally get tedious to add the necessary folders to your search path every time you start a Matlab session To avoid this, you can add selected folders to the search path for an indefinite period of time by editing the Matlab startup file

The Matlab Startup File

When Matlab starts up, it executes a script file called *startup.m*. By creating and/or editing this file and adding the appropriate paths to Matlab's search path using ***addpath*** as indicated above, you can effectively avoid having to set the path each time you start Matlab. On Microsoft Windows this file resides in the Matlab startup directory (*My Documents\MATLAB*,or *Documents\MATLAB* depending on OS version) or within the directory from which Matlab is started on unix/linux OS platforms. *For the following practical session we strongly suggest that you create a folder of your choice to contain your Matlab work and set the search path to include this folder.*

[1] "Commands" are really primitive functions. We will discuss *functions* and M files shortly but the distinction need not concern us here.

Getting started - The Matlab workspace

The first and simplest way to use Matlab is to enter commands at the Matlab prompt (usually one at a time) and allow Matlab to respond accordingly. Enter the following sequence of Matlab commands :–

```
                                    What's Happening
>> A=[1 2; 3 4]                     %define the 2 x 2 matrix A
>> B=[2 1; 0 1]                     %Define the 2 x 2 matrix B
>> C=A+B                            %Let C be the addition of the two matrices
```

Matlab responds by assigning the given matrices to the variables A and B and variable C was set equal to their sum. Note that in each case, Matlab prints the result of the operation on the screen. The variables A,B and C now exist in the *Matlab workspace*.

Typing the command *whos* at the Matlab prompt shows what variables currently exist in the workspace :–

```
>> whos

  Name      Size            Bytes  Class

  A         2x2                32  double array
  B         2x2                32  double array
  C         2x2                32  double array

Grand total is 12 elements using 96 bytes
```

We can use the quantities A,B and C in subsequent calculations or operations. Enter the following (including the semi-colon) :–

```
>> D=A*B-C ;                        %D = matrix product of A and B with C subtracted
>>I=imread('cameraman.tif') ;        %Read in tif image and assign to variable I
```

This time you will notice that Matlab does not print the results to the screen. The semi-colon after the command *suppresses* the output. If you omit the semi-colon in the two lines above, both D and all the pixel values of the image I (all 65536 of them!) will print to the screen.

The variables D and I now also exist in the workspace.

```
>> whos
  Name      Size            Bytes  Class

  A         2x2                32  double array
  B         2x2                32  double array
  C         2x2                32  double array
  D         2x2                32  double array
  I       256x256           65536  uint8 array
```

Saving, clearing and loading Data

The easiest way to save the results of a Matlab session is to use the **save** command :–

```
>> save mysession1
```

The **save** command will save **all the variables** we created above in a file called *mysession1.mat* inside the folder/directory in which you are currently running Matlab. The .mat extension on the saved file indicates that this is a Matlab data file.

You may wish to **clear** certain variables from the workspace –

```
>> clear A B C D;  whos
```

This will show that only I (the image) now remains in the workspace – A,B,C and D have all vanished. Note also that we can type multiple commands on a single line provided they are separated by a semi-colon. The semi colon effectively indicates to the Matlab interpreter that a given command has been entered. Typing **clear** with no specific variables named will clear all variables from the workspace.

```
>> clear;  whos
```

We see that all the variables have been erased and the workspace is now empty.
We have saved the results of our short Matlab session up to now in the file mysession1.mat. How then can we restore these results to the workspace ? We achieve this via the Matlab function **load**. Try -

```
>> load mysession1;  whos
```

The variables A,B,C,D and I now exist again in the workspace.

M files – scripts and functions

Typing directly into the command window is a useful way to get familiar with Matlab and is ideal for trying out relatively simple sequences of commands. Using Matlab in this way is somewhat similar to having a very advanced desktop calculator – you don't need to write "real programs" but just issue simple commands and you get the response back immediately. However, as you develop longer and more complex procedures, the limitations of this approach become evident and you will want to save long sequences of Matlab commands for future use – i.e. the next step up is to ***write programs***. Matlab programs are called "m-files" and there are two basic kinds – scripts and functions.

Matlab Script Files
The simplest form of Matlab program is a so-called *script* file. A script file is a text file containing a sequence of valid Matlab commands. The following example defines the recipe for how to write and execute (i.e. run) a script file. Try it out.

Creating and running a Matlab Script file

- In the Matlab command window, left-click the *file* drop-down menu and select *New – Blank M-file*.
- The Matlab text editor appears.
- We type a sequence of Matlab commands :–

```
                                  What's Happening
clear;                            %First clear workspace of all variables
A=imread('cameraman.tif');        %Read in first tif format image
B=imread('circuit.tif');          %Read in second tif format image
imshow(A); pause(3);              %Display 1st image, wait 3 seconds
imshow(B); pause(3);              %Display 2nd image, wait 3 seconds
B = imresize(B, size(A));         % resize image B to the size of A
C=imadd(A,B); imshow(C);          %Display 8-bit sum
```

- We save the text file using the drop-down menu on the text editor (*file – save as*). Save this file (into a folder which is on your current Matlab search path) with the name *add_image.m.*
- To run this file, type the name of the script file at the Matlab prompt and press return -

```
>>add_image
```

You will find that the individual images are first displayed for 3 seconds each followed by their 8-bit truncated sum image shown below :-

We make two general comments :–

- Anything written on a line to the right of a % symbol is *not interpreted* as commands by Matlab. In other words, anything to the right is a *comment* which is inserted simply for explanatory purposes and to make the program comprehensible for the program user. You can edit the script you just created and include the comment lines in your script, if you wish.

- All variables which are declared/created when a script file is executed exist in the Matlab workspace when execution is complete. (Type *whos* to confirm that the variables A,B and C all exist in the workspace)

Matlab Function Files

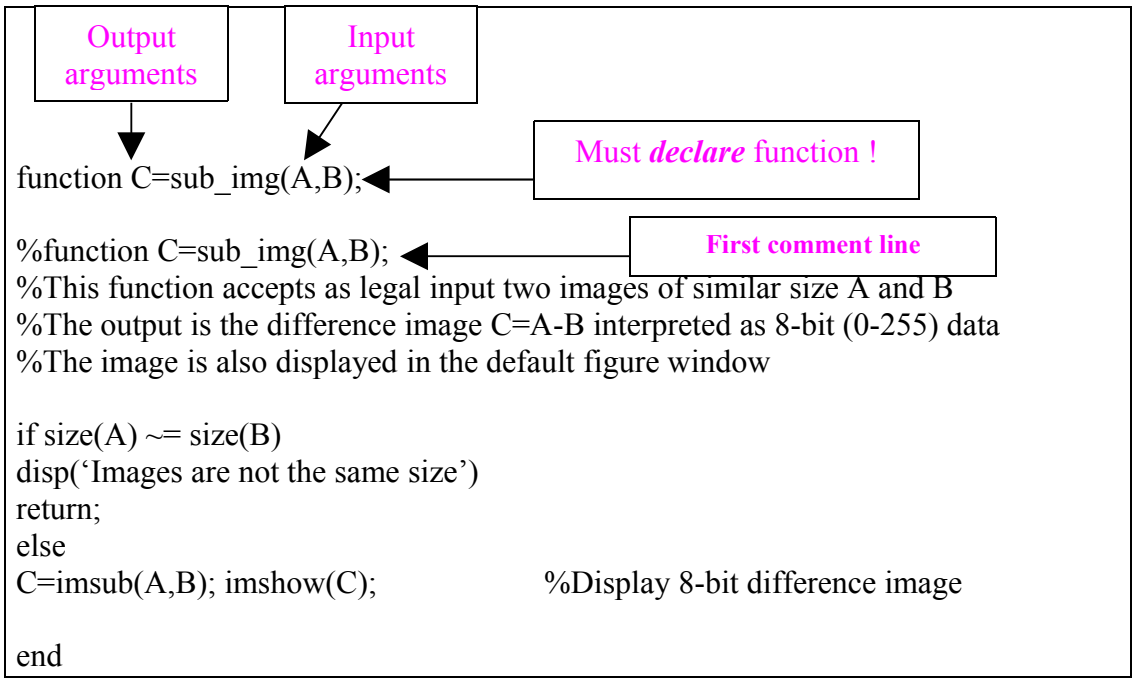
The second kind of m-file is the Matlab function. Functions are also sequences of Matlab commands but differ from scripts in an important way. The key difference between scripts and functions is that when a function is executed *only the declared output arguments are returned to the workspace.* Any other variables created within the function vanish when Matlab returns to the workspace.

The simplest way to think of a function is as a computational routine which requires *certain input arguments (i.e.*data/variables) *to produce required output arguments.* In the traditional programming language sense this is a *sub-routine.* Thus, we require from the function that it return the output quantities we require, given the necessary input. Intermediate quantities which might need to be generated to achieve this aim are of no ultimate interest.

Creating and running a Matlab function file

Function m files are created using the text editor in exactly the same way as scripts. Try creating the model example function below.

- Select *file - New – Blank M-file* from the drop-down menu in the Matlab command window. (alternatively select *file - New – Function M-file* for a pre-defined function declaration layout for you to edit)
- The Matlab text editor appears.
- **We must declare the function on the first line.** This declaration is what distinguishes a script from a function. The function declaration requires that you specify the output and input arguments by chosen names. The required syntax is shown below.




```
function C=sub_img(A,B);

%function C=sub_img(A,B);
%This function accepts as legal input two images of similar size A and B
%The output is the difference image C=A-B interpreted as 8-bit (0-255) data
%The image is also displayed in the default figure window

if size(A) ~= size(B)
disp('Images are not the same size')
return;
else
C=imsub(A,B); imshow(C);                %Display 8-bit difference image

end
```

Note the following:

- After the function declaration on line 1, we have a sequence of *comment lines* beginning with a %. Strictly, these are optional but highly recommended as a means of describing the basic purpose of the function and the required input and output arguments.
- After the comments, appears a sequence of Matlab commands.

We save the text file using the drop-down menu on the text editor (*file – save as*). You should choose a name for the function file which exactly matches the declared function name in the first line of the file. For the example above, use sub_img.m.

To run this file and see what it does, type the following at the Matlab prompt -

```
                                                    What's Happening
>> clear;                                           % clear workspace
>> A=imread('cameraman.tif'); B=imread('ic.tif');   %Read in images
>> D=sub_img(A,B);                                  %Execute function
>> whos                                             %Display variables
```

You should see displayed the truncated 8-bit difference image of A and B. The Matlab response to **whos** shows that variables A,B and D exist in the workspace.

Getting help on Matlab functions

It is possible to get help on any Matlab function or operator simply by typing ***help "function_name"***. For example, to get help on how to use Matlab's general purpose function for reading images into the workspace ***imread*****,** we type

```
>> help imread
```

The (in this case) rather lengthy response, shows the variety of permissible syntaxes for using the function and the acceptable image formats.

You can make good use of the help facility for **your own functions** in exactly the same way. Try typing -

```
>> help sub_img
```

Matlab responds by printing to the screen the first *unbroken block* of comment lines starting with the first comment line (%) that appears within the file :-

```
function C=subims(A,B);
This function accepts as legal input two images of similar size A and B
The output is the difference image C=A-B interpreted as 8-bit (0-255) data
The image is also displayed in the default figure window
```

As you become more familiar with Matlab and start writing your own specialized functions, it is good to get into the habit of documenting the use of your functions in this way. This is a very useful way of reminding yourself and other users of the basic purpose of functions (both Matlab's and your own) and the required input/output syntax.

Saving and printing graphics

After getting to grips with the basics, one of the earliest things you are likely to want to do is to save and print graphics output such as graphs and images. Run your script file **add_image.m** again (>> add_image) to create the figure window containing the truncated 8-bit addition of the images of the cameraman and the integrated circuit.

To print a hard copy of the figure, select the print option from the Command Window menu (*file – print*). This will invoke the default printer for your system.

To save the figure window to disk on your computer, you have two options –

- Select the **export** option (*file – export*) on the command window menu. You are prompted for the type of format in which you would like to save the graphic and the folder/directory in which you want it saved.

- You use the Matlab **print** function with the specified options. **print** allows you to save an image or graph in a very large number of standard formats. Type

  For example, typing :-

```
>> print –f1 -djpeg 'integcamera.jpg'
```

saves the displayed figure (Figure 1 specified as -f1) to the current working directory as a JPEG format image file with name *integcamera.jpg*. An alternative is the Matlab *imwrite* (>> doc imwrite) to write image variables from the workspace directly to an output image file (without surrounding image border).

You may view the many possible formats for saving Matlab graphics output by using the online help facility (>> help print).

So far in this first section, we have shown you some of the basic Matlab commands that you need to manage the workspace. These commands (plus some other useful ones) are summarised in the table below. For a more detailed explanation, just use the help function (>> help "function name") at the command window.

The 20 Most Useful Basic Matlab Commands	
Command/function	**Purpose**
whos	Shows the variables existing in the workspace
save	Saves current variables to disk
load	Retrieves saved variables and loads them into Matlab
clear	Clear variables from workspace
; (semi-colon)	Suppresses printing to screen
help	Help on syntax and use of specific Matlab functions
path	Sets and shows the Matlab directory search path
pwd	Displays the present working directory (pwd)
diary	Saves your Matlab session to a text file
more on/off	Turn Matlab scrolling on/off to one page at a time
print	Print the current active figure to a file
input	Prompt user for input.
lookfor	Search for
Pause	Pauses program execution for specified time
drawnow	Forces matlab to display graphics as they are created
keyboard	Returns control from file to the command window
type	Display the contents of an M file on the screen
quit	Quit Matlab

SECTION III: Building Basic Expressions in Matlab

Expressions in Matlab employ a combination of *variables, operators, numbers* and *functions*. For example, in the following expression :-

```
>> x=1:10, y=sin(pi.*x./2)
```

x and ***y*** are variables, **.* and ./** are operators, ***pi*** and ***2*** are numbers and ***sin*** is a function (N.B. ***pi*** is a special case of a built in numerical constant in Matlab).

Implicit Vectorization

If you type the expression above at the prompt, you will immediately notice an interesting and important feature of Matlab. The variable ***x*** is a 10 element vector containing the integers from 1 to 10 inclusive. When we write $y = \sin(\frac{\pi x}{2})$ as in the example above, Matlab evaluates the expression *at each value of the input vector* **x,** producing an output vector also of 10 elements. This is sometimes called **implicit**

vectorization. The nice thing about implicit vectorisation is that it helps to avoid ungainly loops and allows us to write more compact, intuitive expressions[2].

Try out the following examples :–

```
>>x=0:pi/30:pi                          %Make 30 element vector x from 0-pi
```

```
>>y=cos(x).^2; plot(x,y)        %Sample cos^2x at given values and plot
                                %Generate cos^2 x and graph function
```

```
>>A=[1 2; 3 4], B=[0 1; –1 0]     %Create 2 x 2 matrices  A and B
>>A-B                             %Subtract matrix B from A. ans=[1 1; 4 4]
>>A.^2                            %Square each element of A. and=[1 4; 9 16]
```

A summary description follows of the way in which Matlab defines and deals with *variables, operators, numbers* and *functions*.

Variables

Matlab *does not require type declarations or dimension* statements (as are common requirements of other high level programming languages). In this sense we can describe Matlab as a *loosely typed* programming environment/language. When a new variable name is encountered, the appropriate storage space required is automatically allocated.

```
>>clear;                          %Clear workspace
>>A=[1 0; 0 1]                    %Storage for the 2x2 array A is automatically created
```

Further, the required storage for the variable can be *dynamically* altered. Try the following :-

```
>>A(3,:)  = [5 6]                 %A third row is added to A – dynamic allocation occurs
```

Variable names must begin with a letter and may extend to a maximum of 31 alphanumeric characters. N.B. Matlab is case sensitive – B and b refer to different variables.

[2] If you've programmed before in Fortran, Pascal or C/C++, you'll immediately recognise the difference.

Operators

Expressions may employ any of the familiar arithmetic and precedence operators summarized in the table below :–

Operator	Description
+	Addition
-	Subtraction
./	Division (scalar)
.*	Multiplication (scalar)
.^	Power (scalar)
‘	Complex conjugate transpose
( )	Specify evaluation order
/	Division (matrix)
*	Multiplication (matrix)
^	Power (matrix)

We build expressions using these operators in the usual way :–

```
>>x=0:10, y=exp(-sqrt(x))
>>A=[1 2; 3 4], B=[0 1; –1 0], C=[2 1; 2 1];
>>(A-B)-(B+C)                                    %ans=[-1 –1; –1 3]
```

Matrix and scalar operators

In part two of this primer, we will see that Matlab (derived from **Mat**rix **Lab**oratory) is particularly powerful for dealing with matrices.

Warning !
It is particularly important to be aware from the outset of the difference between the scalar operations for *division, multiplication and power exponentiation* **(*./ .* .^*)** which include a dot before the conventional symbol and and their matrix counterparts **(*/ * ^*)** which do not !

Try out the following commands and carefully digest the difference between them :–

```
>>A=[1 2; 3 4], B=[0 1; –1 0], C=[2 1; 2 1];
>>A.^2                    %Each element of A is squared.  ans=[1 4; 9 16]
>>A*A                     %Matrix multiplication of A and A. ans=[7 10; 15 22]
>>A^2                     %Matrix multiplication implied. ans=[7 10 15 22]
>>A.*B                    %Element by element multiplication. ans=[0 2; 3 0]
>>A*B                     %Matrix multiplication. Ans=[2 1; 4 3]
>>A./C                    %Matrix Division
                %Division of corresponding elements. ans=[0.5 2 1.5 4]
```

Numbers

Numbers can be expressed using conventional decimal notation or scientific notation. Matlab handles complex numbers naturally without any need for special declarations. Some examples of legal numbers in Matlab are ;–

```
0.003   -67      17.56789          (standard decimal)
3.55e+7         -1.23e-6           (scientific notation)
6+3i    -0.142857j      3e+5j      (complex notation)
```

Note that Matlab will interpret i or j as the imaginary quantity $\sqrt{-1}$ when written beside a number with no operator between them (e.g. 3i means $3\sqrt{-1}$). If an operator appears between them (e.g. 3.*i), this will be interpreted as the corresponding complex quantity *provided* i is not a variable with a preassigned value. For example ;-

```
>>z1=3+4i, z2=5+12.*j                %z1 and z2 are complex
```

```
>>j=1:5, z3=3+4j, y=3+4.*j           %z3 complex, y is real vector
```

Functions

Matlab provides a large number of standard, elementary mathematical functions (for example *exp, sqrt, log, sin* etc). For a complete list of the elementary functions provided type :–

```
>> help elfun
```

at the Matlab prompt. As we discussed earlier, functions do not have to be elementary like *sin* or *log*. Matlab provides many high level functions and you will soon learn to write your own.

Building Expressions

Matlab expressions are built by combining variables, operators, numbers and functions in a logically consistent way. Here are just a few simple examples of legitimate Matlab expressions :–

```
>>clear;
>>theta=pi./4;
>>z=exp(i.*th)                                  %ans=0.7071 + 0.7071i
>>wz=z+z.^-1                                    %ans=1.4142
>>imag(wz)                                      %ans=0
>>1+( log(sin(theta)) –sin(log(theta)) ).^i     %ans=0.9735 - 0.0341I
```

In the next part of this primer, we will take a more detailed look at the Matlab language and explore the key programming constructs.