Laboratory 1 Getting Started with MATLAB

This laboratory presents some of the basics of MATLAB. It assumes no prior knowledge of MATLAB but does assume a familiarity with PCs and basic mathematics and computer science. Topics covered include interactive use of MATLAB using the command window, numbers and operations, saving and reloading work, using help, and MATLAB resources on the Internet. Writing files and programming is covered in the next laboratory.

You are expected to have read through these notes PRIOR to coming to the laboratory.

During the laboratory session you are expected to complete all of the exercises. Note that there is no sharp division between notes and exercises. It will be instructive to implement some of the examples provided in the notes.

The MATLAB Desktop

You start MATLAB by double clicking on the MATLAB icon that should be on the desktop of your computer. The MATLAB desktop appears with a number of windows. The default view shows the following windows:

Command Window Enter commands for processing.

Toolstrip Provides access to tools, demos and documentation.

Workspace Displays and allows manipulation of variables.

Current Directory For directory and file manipulation.

Layout -> Command History adds history of prior commands used in the command window.

Layout -> Default restores default view

The Command Window

The Command Window is the most important of the windows. It allows a user to enter commands for MATLAB to process. To enter a command, type the command (followed by `Enter`) at the prompt `>>`. For instance,

```
>> s= 1 + 2
s =
   3
>> y = sin(pi/4)
y =
   0.7071
```

In the second example the trigonometric sine function and the constant π are used. In MATLAB these are named `sin` and `pi`, respectively. Note that the results of these computations are saved as **variables** whose names are chosen by the user. Once created, these variables remain in the **workspace** (and can be viewed in the Workspace Window) until they are cleared. To see the value of any variable, enter its name in the Command Window. For instance,

```
>> s
s =
    3
```

A variable name begins with a letter, followed by letters, numbers or underscores. MATLAB recognizes only the first 31 characters of a variable name.

Notice that the results of the calculation are echoed to the screen unless the command is terminated by a semi-colon. For example,

```
>> y = sin(pi/4);
>>
```

If the calculation results are not assigned to a specified variable, the results are assigned to the variable ans. For example,

```
>> sin(pi/4)
ans =
   0.7071
```

To change the format of numbers displayed in the command window you can use one of the several formats that are available in MATLAB. The default format is called **short** (four digits after the decimal point.) In order to display more digits click on **Preferences-> Command Window->Numeric Format button**. Alternatively, you can use the command format in the command window. Observe the effects of

```
>> format long
>> y
```

To change back to the default (short) format use format short or, simply, format.

You may find the Home, End, Esc and ←, ↑, →, ↓ keys useful when entering commands. For example, previous commands are saved in a buffer and can be accessed with the ↑ and ↓ keys. (They can also be accessed via the Command History window).

When you close MATLAB, all unsaved information residing in the MATLAB workspace will be lost. For more advanced work, it is preferable to use a text editor to create an **M-file** consisting of a sequence of commands and then executing the M-file. This topic is covered in the next Laboratory "Programming in MATLAB".

Numbers, Variables and Operations

Two types of numbers are used in MATLAB: **real** numbers (stored in 8 bytes) and **complex** numbers (stored in 16 bytes). The numbers you have encountered so far are real.

Variables realmin and realmax denote the smallest and the largest positive real numbers in MATLAB. For instance,

```
>> realmin
ans =
  2.2251e-308
```

Complex numbers in MATLAB are represented in rectangular form, with the imaginary unit $\sqrt{-1}$ pre-defined as either i or j.

```
>> i
ans =
      0.0000 + 1.0000i
```

A complex number can be entered in several ways. The following commands are equivalent:

```
>> c = 2 + 3*j
>> c = 2 + j*3
>> c = 2 + 3j
```

However, the following does not work

```
>> c = 2 + j3
??? Undefined function or variable 'j3'.
```

You can re-define `i` (or `j` or any other such pre-defined variable) but, when you clear it, it reverts to its pre-defined value. It is best to avoid using `i` and `j` for other purposes. The function `exp` can be used to convert from polar to rectangular form. For example, to assign the value $10\angle\pi/4$ to x:

```
>> x = 10*exp(j*pi/4)
x =
   7.0711 + 7.0711i
```

The real, imaginary, magnitude, and phase components of a complex number can be found using `real`, `imag`, `abs`, `angle`, respectively. Note that angles are in radians.

In addition to the two classes of numbers mentioned above, MATLAB has variables representing non-numbers: `inf` and `NaN`.

The `inf` represents **infinity**. Infinity is generated by overflow or by the operation of dividing by zero. The `NaN` stands for **not-a-number** and is obtained as a result of mathematically undefined operations such as 0.0/0.0 or $\infty - \infty$.

As well as numeric variables, you will use **logical** (boolean) variables. A logical variable has the value of 1 (TRUE) or 0 (FALSE). A simple example:

```
>> b = 3 > 2
b =
     1
```

Note the operator precedence.

The basic **arithmetic operators** in MATLAB are **addition** (+), **subtraction** (-), **multiplication** (*), **division** (\ or /) and **exponentiation** (^). MATLAB has two division operators / (**right division**) and \ (**left division**).

The **relational operators** are **less than** (<), **greater than** (>), **less than or equal** (<=), **greater than or equal** (>=), **equal to** (==) and **not equal to**(~=). Note the ~=, *not* !=. Do not confuse == (equal to) with = (assignment).

The **logical operators** in MATLAB are | (OR), & (AND) and ~ (NOT).

All of these operators operate on arrays (see next section).

Exercise 1

- What are the values of 8/2 and 8\2?
- Find the magnitude and phase in radians of 3+j6.
- What are the real and imaginary parts of $5\angle 1.2$, where the angle is in radians?

Exercise 2

In a circuit problem involving phasors, a voltage V is given by

$$V_S = E + ZI_L$$

If $E = 100 + j10$ V, $I_L = 5 + j5$ A and $Z = 2\angle 40° \ \Omega$, determine the magnitude and phase (in degrees) of V_S. Remember to convert between degrees and radians!

Answers: mag = 104.0590 V, phase = 13.3846°.

Getting Help

One of the nice features of MATLAB is its help system. To learn more about a function you are to use, say svd, you can use the function help in the Command Window.

```
>> help svd
SVD Singular value decomposition.
.....
See also SVDS, GSVD.
Overloaded methods
.....
```

There are other ways of obtaining information but the help command will prove invaluable.

If you do not remember the exact name of a function, use the command lookfor followed by the incomplete name of a function. In the following example we use a "word" sv

```
>> lookfor sv
hsv
(list)
```

You can also get help by browsing in theToolstrip window or by using the helpwin command in the Command window.

The helpwin command used with a function name as argument provides more extensive information than help.

Arrays

MATLAB stands for MATrix LABoratory. The basic element is the array. Matrices are, of course, two-dimensional arrays. Vectors and scalars are considered to be special cases of arrays. (If you look in the Workspace window, you will see the scalar numeric variables used so far described as double arrays – a "double" being a 64 bit floating point representation). MATLAB functions are designed to operate on arrays.

Square brackets are used when entering an array. Elements on the same row are separated by a space or comma, and rows are separated by a semi-colon or carriage return.

```
>> x = [1+j 2; 4 5*j]
x =
  1.0000 + 1.0000i   2.0000 + 0.0000i
  4.0000 + 0.0000i   0.0000 + 5.0000i
>> y = [1 j; j 2; 2-3j 3]
y =
  1.0000 + 0.0000i   0.0000 + 1.0000i
  0.0000 + 1.0000i   2.0000 + 0.0000i
  2.0000 - 3.0000i   3.0000 + 0.0000i
```

Note that you cannot use a space on either side of a '+' or '-' when entering a complex number in a matrix because a space is assumed to separate matrix elements.

Matrix addition, subtraction and multiplication must obey the usual rules. Observe, for example, the results of the commands

```
>> x*y
>> y*x
```

See the Appendix for more details. Most importantly, with matrices one must distinguish between normal matrix multiplication (* operator) and division (/ operator) versus a point-by-point multiplication (.* operator) and division (./ operator). For example,

```
>> a = [1 2;3 4]
>> b = [3 3;3 3]
>> x*y
ans =
    9    9
   21   21
>> x.*y
ans =
    3    6
    9   12
```

Exercise 3

Write down explicitly the calculations involved in computing x*y and x.*y shown above.

The transpose of a matrix is obtained using the operator **.'** (a period followed by a dash). For example**,**

```
>> x = [1+j 2; 4 5*j];
>> x.'
   ans =
  1.0000 + 1.0000i   4.0000 + 0.0000i
  2.0000 + 0.0000i   0.0000 + 5.0000i
```

The complex conjugate transpose is obtained with the operator ' (dash). Thus

```
>> x'
```

```
ans =
  1.0000 - 1.0000i   4.0000 + 0.0000i
  2.0000 + 0.0000i   0.0000 - 5.0000i
```

Range generating statements are often useful.

```
>> x = 0:5
x =
    0    1    2    3    4    5
>> x = 1:0.2:2
x =
   1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

If the step size is omitted, it is assumed to be equal to 1. Use `help colon` for details.

There are a number of functions for generating special matrices. Observe, for example, the results of the following commands:

```
>> zeros(4,6)
>> y = ones(2,3)
>> eye(3)
```

The size of a matrix or vector can be determined using the functions `size` and `length`. With `y` defined as above, observe the results of the following commands:

```
>> size(y)
>> [nr, nc] = size(y)
>> length(y)
```

Note that `length` returns the larger of the two size values. Note also how more than one variable can be assigned by a function call by enclosing them in square brackets separated by commas.

Functions such as `sin`, `exp`, `sqrt` operate on a matrix, element by element, to produce a matrix of the same dimension. For example, to generate a vector containing the values $10\sin(3\pi t)$ for $t = 0, 0.1, \ldots, 0.5$:

```
>> t = 0:0.1:0.5;
>> f = 10*sin(3*pi*t)
f =
    0    8.0902    9.5106    3.0902   -5.8779  -10.0000
```

Matrices can be used as components in constructing a larger matrix. An example:

```
>> v = [[1 2;3 4] zeros(2,3) ones(2,4)]
v =
    1    2    0    0    0    1    1    1    1
    3    4    0    0    0    1    1    1    1
```

As well as the normal matrix operations, we repeat again that point by point operations can be carried out using the normal operator preceded by a period. For example, point by point multiplication is carried out using the `.*` operator. The two matrices involved must be exactly the same size (or one must be a scalar).

```
>> a = [1 2;3 4;5 6]
a =
    1    2
    3    4
    5    6
>> b = [1 0;2 1;3 3]
b =
    1    0
    2    1
    3    3
>> a.*b
ans =
   1    0
    6    4
   15   18
```

Accessing elements of a matrix is carried out using index elements or lists in parentheses. In MATLAB, the elements of a vector or matrix are always indexed from 1. For example,

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
A =
    1    2    3    4
    5    6    7    8
    9   10   11   12
>> A(2,3)
ans =
    7
>> A(3,2:4)
ans =
   10   11   12
>> A(:,4)
ans =
    4
    8
   12
```

Note the use the colon **:** to select a range of elements or an entire column (or row). More examples:

```
>> v = [2 1 -1 5 7]
v =
    2    1   -1    5    7
>> u = v(1:2:5)
u =
    2   -1    7
>> g(1:2:7) = v(2:5)
g =
    1    0   -1    0    5    0    7
```

The last command shows how ranges can be used on the left hand side of an assignment statement. The unspecified elements of g are unaffected; in this case, g did not exist prior to the command, so the unspecified elements are set to zero.

If the left hand side of an assignment is an array and the right is a scalar, the scalar value is "expanded" to fill the matrix. Some examples:

```
>> A = zeros(3,4)
A =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> A(1:2,:) = 7
A =
     7     7     7     7
     7     7     7     7
     0     0     0     0
>> A(2:3,[2 4])=3
A =
     7     7     7     7
     7     3     7     3
     0     3     0     3
```

As well as assigning values to specified elements of an array, we can remove blocks of an array by assigning the empty array []. For example, to remove the second row of the A matrix:

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> A(2,:) = []
A =
     1     2     3     4
     9    10    11    12
```

Exercise 4

(a) Set up a 6 by 8 matrix A in which all elements on the top, bottom, left and right edges are zero and all other elements are equal to 5. You should need only two commands.

(b) Set up a row vector B containing the digits of your SID as elements. Now reverse the order of the digits in B, first using fliplr and then without using the MATLAB function `fliplr`.

(c) Use the function `magic` to generate a 10 by 10 magic matrix C. A magic matrix is one where all rows, columns and diagonals add to the same number. Use the function `sum` to verify that all columns of C add to 505. Repeat for the rows. Repeat for the main diagonal with the help of the function `diag`.

(d) Generate a table on screen with 10 rows and 3 columns. The first column contains the integers 1 to 10, the second contains the square root of the number in the first column and the third contains the square.

Exercise 5

Consider the linear equations

$$x_1 + 3x_3 = 10$$
$$8x_1 + 2x_2 - 5x_3 = -3$$
$$-x_1 + 5x_2 + 3x_3 = 18$$

or, in matrix form, $Ax = b$, where A is a 3 by 3 matrix and x and b are 3-vectors.

Set up the matrix A and the vector b and solve for x. (Use the function inv to obtain the inverse of A. Alternatively, you can use A\B to do a "matrix left divide").

Answer: $x = [1\ 2\ 3]^T$

Strings

A character string in MATLAB is a special numerical array of ASCII values. It is created by enclosing text in single quotes. Use two quotes " to include a quote in the string.

```
» mystr = ' MATLAB''s great!'
mystr =
MATLAB's great!
```

To see the numerical representation of mystr use the function double:

```
>> myascii = double(mystr)
myascii =
  Columns 1 through 12
    32   77   65   84   76   65   66   39  115   32  103  114
  Columns 13 through 16
   101   97  116   33
```

The reverse operation is carried out by the function char.

```
>> char(myascii)
ans =
MATLAB's great!
```

Strings can be concatenated in the same way as arrays:

```
>> a = 'If only pigs'; b = ' could fly.';
>> [a b]
ans =
If only pigs could fly.
```

Concatenation can also be carried out with the function strcat.

Some examples on conversion between numbers and strings:

```
> s = ['The square root of ', int2str(2), ' is ', num2str(sqrt(2))]
s =
The square root of 2 is 1.4142
```

```
>> s = sprintf('The square root of %d is %0.5g', 2, sqrt(2))
s =
The square root of 2 is 1.4142
>> str2num('1.378e2')
ans =
  137.8000
```

There are a large number of functions for testing and manipulating strings. See the on-line documentation. Most operate on arrays of strings.

Polynomials

In MATLAB, a polynomial is conventionally represented by a row vector of the coefficients of the powers in descending order. Thus the polynomial $p(x) = 2x^4 + 6x^3 - 9x + 24$ can be represented as

```
» p = [2 6 0 -9 24];
```

To add or subtract two polynomials, you cannot normally simply add or subtract the coefficient vectors. If one vector is shorter than the other, it must be padded out with leading zeroes. Suppose we define $q(x) = x^2 + 2$:

```
» q = [1 0 2];
```

We form the sum with

```
>> p + [0 0 q]
ans =
    2    6    1   -9   26
```

To multiply two polynomials, we convolve the coefficient vectors. Thus to form the product $q(x)p(x)$:

```
>> conv(p, q)
ans =
    2   6   4   3   24   -18   48
```

The function polyval evaluates a polynomial at specified values of the independent variable. To find the values of $p(x)$ at x = -1, 0, 1 and 4:

```
» x = [-1, 0, 1 4];
» polyval(p, x)
ans =
   29   24   23  884
```

The function polyder calculates the derivative:

```
» polyder(p)
ans =
   8  18   0  -9
```

The roots of a polynomial are conventionally written as a column vector. The following example shows how to determine the roots of $p(x)$ using the function roots.

```
» r = roots(p)
r =
   -2.4067 + 1.1077i
   -2.4067 - 1.1077i
```

```
0.9067 + 0.9420i
0.9067 - 0.9420i
```

The inverse function poly produces the polynomial from the roots. Thus poly(r) will produce the original vector of coefficients.

Introduction to Graphics

It is strongly advised that you enter the code shown in this section in the Command window and view the resulting displays which have been omitted for brevity.

The ability to display data quickly and in a wide variety of ways is one of MATLAB's most powerful features. The most commonly used function is plot. The following example graphs sin(x) for $0 \le x \le 5\pi$:

```
>> x = linspace(0, 5*pi, 51);
>> y = sin(x);
>> plot(x, y)
```

The first command uses the function linspace to creates a vector of 51 values (50 intervals) linearly spread over the range specified. When plot is used, as here, with two vector arguments, it plots the first along the abscissa and the second along the ordinate. 51 points are plotted, joined by straight lines. The graph appears in a **Figure window**.

The appearance of the graph can be controlled in a number of ways. The colour, marker and linestyle can be specified. Observe the effects of the command:

```
>> plot(x, y, 'ro--')
```

The string tells MATLAB to use the colour red, o as a marker and a dashed line. The default colour and linestyle are blue and solid respectively. If no marker is specified, none is used. If a marker is specified, but no linestyle, no line is drawn. Use help plot for details.

A title, axis labelling, a legend, grid lines and other features can be added to a graph. For example, to add a title, use:

```
>> plot(x, y)
>> title('Graph of sin(x)')
```

The addition of such features will be explored in the exercises below. It is also possible to add features directly in the Figure window using the **Insert** menu.

Suppose we wish to plot sin(x) and cos(x) on the same graph. One way:

```
>> x = linspace(0, 5*pi, 51);
>> y = sin(x);
>> z = cos(x);
>> plot(x, y, x, z)
```

One alternative to the last command is

```
>> plot(x, [y; z])
```

Here there are only two arguments. The second is a matrix so each row of the matrix is plotted versus x. Different colours, etc, can be specified for individual graphs. Observe, for example, the effect of

```
>> plot(x, y, 'o', x, z, 'x')
```

One Figure window can hold more than one set of axes. Suppose we wish to plot sin(x) and cos(x) on separate axes, one above the other. We can use subplot to divide the window into a matrix of small axes. Suppose x, y and z are as defined above. The following additional commands produce the desired result:

```
>> subplot(2, 1, 1), plot(x, y)
>> subplot(2, 1, 2), plot(x, z)
```

Exercise 6

Define vectors to represent the functions $x(t) = \exp(-0.8t)$, $y(t) = \cos(8t)$ and $z(t) = 10x(t)y(t) = 10\exp(-0.8t)\cos(8t)$. Use values of t ranging from 0 to 5.0 at intervals of 0.05.

Plot z versus t. Add a title "An exponentially decaying sinusoid", label the x-axis "Time", label the y-axis "Value of z", set the range of z values to be -12 to 12 and turn on the grid lines. Use the functions title, xlabel, ylabel, axis (or ylim) and grid.

Exercise 7

Use subplot plot x, y and z of Exercise 6 on separate axes in the same Figure window, one above the other.

Miscellaneous Information

This section contains additional information about MATLAB, much of which will not be used in this laboratory but may be useful later.

Suppressing Output to the Screen

You can suppress output to the screen by terminating a command with a semicolon.

Interrupting a Running Program

To interrupt a running program press the **Ctrl** and **c** keys simultaneously. Sometimes you have to repeat pressing these keys a couple of times to halt execution of your program. This is not a recommended way to exit a program, however, in certain circumstances it is necessary if you get into an infinite loop.

Long Command Lines

To enter a command that is too long to be typed in one line, use three periods, **...** , followed by Enter. For example,

```
>> x = sin(1) - sin(2) + sin(3) - sin(4) + sin(5) -...
sin(6) + sin(7) - sin(8) + sin(9) - sin(10)
x =
   0.7744
```

Viewing Variables

All variables used in the current MATLAB session are retained in the workspace. You can view the content of the workspace in the Workspace window. You can also check contents of the workspace in the command window. For instance, the command who can be used to list all current variables. You can also use whos to generate a more informative list.

Saving and Reloading Data

To save your current workspace select **Save Workspace as…** from the **File** menu. It will be saved as a ***.mat** file in the current directory. Remember that the file you just created must be located in MATLAB's search path. Another way to save your workspace is to use the `save` command in the command window. A saved .mat file can be loaded into the workspace using the `load` command.

If you wish to have a record of your session, you can use the command `diary filename` in the command window. All commands and variables created from then on will be saved in your file. Use `help` for more information.

Demos

To learn more about MATLAB capabilities you can execute the `demo` command.

Exercise 8

Logical operations on arrays produce logical arrays. As an example, suppose we have two numerical vectors x and y. The command

```
>> b = x > y
```

creates a logical vector b, of the same length, which contains a 1 in those positions where the element of x is greater than the element of y.

Suppose we have a vector y of the same length as x. How could you determine the number of elements of y which are equal to the corresponding elements of x? If `x = [0 5 -3 7 1 8 10]` and `y = [1 5 3 0 0 8 -2]`, for example, the answer should be 2.

Exercise 9

In MATLAB you can use a logical vector b to index a numeric vector V of the same length. V(b) will consist of only those elements of V corresponding to a 1 in b. For example if `V = [1 2 3 4 5 6]` and the logical vector `b = [0 1 0 0 1 1]`, then `V(b) = [2 5 6]`.

Use the command `x = rand(1, 10)` generate a row vector containing 10 random numbers in the range 0 to 1.0. Now use commands (they could be combined into one command) which will remove all elements less than 0.5. Repeat for the case when only elements in the range 0.4 to 0.6 inclusive are to be retained.

Exercise 10

Plot the geometric figure described by

$$\frac{(x-3)^2}{36}+\frac{(y+2)^2}{81}=1$$

in the x-y plane. Hint: a parametric representation is

$$x(t)=3+6\cos(t),\ y(t)=-2+9\sin(t) \qquad 0\le t\le 2\pi$$

Add a grid and use `axis` to make intervals on the x and y axes equal.

MATLAB Resources on the Internet

If your computer has an access to the Internet you can learn more about MATLAB and also download user supplied files posted in the public domain. Some pointers are provided below to information related to MATLAB.

The MathWorks Web site: http://www.mathworks.com/.

Matlab community: http://au.mathworks.com/matlabcentral/?s_tid=hp_ff_community

Pages of Matlab links:

http://www.mathworks.com/matlabcentral/linkexchange (Matlab Central)

Appendix – Basic Operations on Matrices

Suppose we have two matrices A and B. Mathematically, the operations A+B and A-B are valid only if the two matrices have the same dimensions (same number of rows and columns). However, MATLAB allows one of the operands to be a scalar. If B is a scalar, for example, A+B is formed by adding B to every element of A.

The two matrices can be multiplied A*B only if the number of columns in A is equal to the number of rows in B. In general, A*B and B*A are not the same even if they exist. Again, it is permissible for either A or B to be a scalar. The scalar multiplies every element of the matrix.

There is no division operation in matrix algebra. However, it may be possible to multiply one matrix by the inverse of another. The inverse of B, if it exists, is the matrix C such that B*C = C*B = unit matrix. For an inverse of B to exist it is necessary (but not sufficient) that B be square. A square matrix which has no inverse is said to be "singular". To multiply A on the left by the inverse of B if it exists, ie to form $B^{-1}A$, we could use the MATLAB code `inv(B)*A` or we could make use of the left division operator and write `B\A`. Similarly, A B^{-1} would be coded as `A*inv(B)` or `A/B`.