Purpose and Objectives

This lab is an introduction to MATLAB. You will learn how to use MATLAB to do calculations involving real and complex numbers and vectors and matrices, how to plot functions and how to write M-files.

Preparation

Little preparatory work is required but a review of calculations with complex numbers and of basic concepts in vectors and matrices will be useful.

These notes assume that you have a basic understanding of Windows and PCs. Seek assistance from a tutor on these matters if necessary.

Using MATLAB

To start MATLAB, find the MATLAB icon and double click it. A MATLAB command window will open and the command line prompt » will be displayed.

You can have matlab execute a command by entering it following a prompt. For more advanced work, it is preferable to use a text editor to create a M-file consisting of a sequence of commands and then executing the M-file. You will learn to use MATLAB in both ways in this Laboratory session. When you write M-files, you will need to save them to disk. To keep the disk tidy, enter the MATLAB command

```matlab
» cd \sigsys
```

or something similar, at the beginning of each MATLAB session. This will cause SIGSYS to be your default directory and any M-files you create will be saved there. If you wish, you may create a subdirectory off SIGSYS and use it instead, but DO NOT save your files anywhere else.
Experimental Work - A Guided Tutorial

On-line help is very useful. When you are reasonably adept with MATLAB, it virtually eliminates the need for hardcopy documentation. To get help with a function called func, use the command help func. Using help signal and help control will display a list of the functions available in the Signal and Control toolboxes respectively. The command help help gives more details on using help.

You may find the Home, End, Esc and ←, ↑, →, ↓ keys useful when entering commands. Previous commands are saved in a buffer and can be accessed with the ↑ and ↓ keys.

This section consists of a series of examples. Work along with the examples by entering the commands following the prompt. (Each command should be terminated with an ENTER).

Basic Calculations

```matlab
» 3*4.5
ans =
    13.5000
```

The results of the calculation are echoed to the screen unless the command is terminated by a semi-colon. If the calculation results are not assigned to a specified variable, they are, as here, automatically assigned to a variable called ans.

A variable is defined in the obvious way.

```matlab
» z = 3*sin(pi/6)
z =
    1.5000
```

sin is one of a number of elementary functions; to see what others are available, type help elfun. Pi is one of a number of pre-defined variables. Defined variables reside in memory until cleared. To display the value of a variable, simply type the variable name. Use help with the functions who and clear to learn something more about variables.

Complex numbers are expressed in terms of the pre-defined variables i or j (both defined as \( \sqrt{-1} \)). A complex number can be entered in several ways.

```matlab
» c = 2 + 3*j
c =
    2.0000 + 3.0000i

» c = 2 + 3j
c =
    2.0000 + 3.0000i

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

Observe that 3j works but j3 does not. You can redefine i (or j or any other such pre-defined variable) but when you clear it it reverts to its pre-defined value.
Some other examples involving complex numbers:

```plaintext
» x = j*exp(j*pi/4)
   x =
   -0.7071 + 0.7071i

» real(x)
   ans =
   -0.7071
```

Observe also the results of `imag(x)`, `conj(x)`, `abs(x)` and `angle(x)`. Note that angles are in radians.

**Matrices and Vectors**

MATLAB stands for MATrix LABoratory. The basic element is the matrix, vectors and scalars being considered as special cases of matrices.

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

```plaintext
» x = [1+j 2 3; 4 5 6]
   x =
   1.0000 + 1.0000i 2.0000 3.0000
   4.0000 5.0000 6.0000

» y = [1 j; 2-3j 3]
   y =
   1.0000 + 0.0000i
   0 + 1.0000i 2.0000
   2.0000 - 3.0000i 3.0000
```

Note that you must not use a space on either side of a ‘+’ or ‘-’ when entering a complex number in a matrix. Matrix multiplication must obey the usual rules; observe the effects of `x * y` and `y * x`. Observe also the result of `x'`.

Range generating statements are useful in generating some vectors.

```plaintext
» 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
```

Note that, if the step size is omitted, it is assumed to be equal to 1. Have a look also at the functions `zeros` and `ones`. Other useful functions are `size` and `length`.

The following example shows how to generate a vector containing the values $10 \sin(3\pi t)$ for $t = 0, 0.1, \ldots, 0.5$.

```plaintext
» t = 0 : 0.1 : 0.5;
» f = 10 * sin(3*pi*t)
   f =
   0 8.0902 9.5106 3.0902 -5.8779 -10.0000
```
Functions such as \texttt{sin}, \texttt{exp}, \texttt{sqrt}, etc operate on an element by element basis.
Matrices can also be used as components in constructing a larger matrix. An example:

\begin{verbatim}
» v = [ zeros(1,4), ones(1,4) ]
  v =
  0 0 0 0 1 1 1 1
\end{verbatim}

As well as the normal matrix operations, point by point operations can be carried out using the normal operator preceded by a dot. For example, point by point multiplication is carried out using the \texttt{.*} operator. The two matrices involved must be exactly the same size.

\begin{verbatim}
» x = [1 2 3 4];
» y = [1 1 2 2];
» x .* y
  ans =
     1  2  6  8
\end{verbatim}

Point by point multiplication of two vectors is equivalent to the inner product. Observe, by contrast, the result of \texttt{x' * y}. (\texttt{x * y} is, of course, invalid).

Accessing elements of a matrix is carried out using an index element or list in parentheses. The elements of a matrix are indexed from 1.

\begin{verbatim}
» 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
\end{verbatim}

Note the use of range statements to pick off a range of elements, and the use of the colon \texttt{:} to denote the entire range. Another example:

\begin{verbatim}
» v = [2 1 -1 5 7];
» f = v(1:2:5)
  f =
    2 -1  7
» f(1:2:7) = v(2:5)
  f =
     1  0 -1  0  5  0  7
\end{verbatim}
The last command shows how ranges can be used on the left hand side of an assignment statement. The unspecified elements of \( f \) are unaffected; in this case, \( f \) did not exist prior to the command, so the unspecified elements are set to zero.

**Polynomials and Related Matters**

In MATLAB, polynomials are conventionally represented by a row vector of the coefficients of the powers in descending order. The roots of a polynomial are conventionally written as a column vector. The following example shows how to determine the roots of \( 2x^4 + 6x^3 - 9x + 24 \) using the function \texttt{roots}.

\[
\begin{align*}
\texttt{p} & = [2 \ 6 \ 0 \ -9 \ 24]; \\
\texttt{r} & = \texttt{roots(p)} \\
\end{align*}
\]

\[
\begin{align*}
\texttt{r} = & \quad -2.4067 + 1.1077i \\
& \quad -2.4067 - 1.1077i \\
& \quad 0.9067 + 0.9420i \\
& \quad 0.9067 - 0.9420i \\
\end{align*}
\]

The inverse function \texttt{poly} produces the polynomial from the roots. Observe the effects of \texttt{poly(r)}. To multiply this polynomial by \( x + 2 \), use \texttt{conv}:

\[
\begin{align*}
\texttt{conv(p, [1 \ 2])} \\
\texttt{ans} = \\
2 \quad 10 \quad 12 \quad -9 \quad 6 \quad 48 \\
\end{align*}
\]

The function \texttt{polyder} calculates the derivative:

\[
\begin{align*}
\texttt{polyder(p)} \\
\texttt{ans} = \\
8 \quad 18 \quad 0 \quad -9 \\
\end{align*}
\]

and \texttt{polyval} evaluates the polynomial at specified values of the independent variable:

\[
\begin{align*}
\texttt{x} & = [-1, \ 0, \ 1, \ 4]; \\
\texttt{polyval(p, x)} \\
\texttt{ans} = \\
29 \quad 24 \quad 23 \quad 884 \\
\end{align*}
\]

The following example illustrates the solution of a set of linear simultaneous equations. Suppose we wish to find \( x \) and \( y \), where

\[
\begin{bmatrix}
1 & 1 \\
1 & 2
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\begin{bmatrix}
5 \\
0
\end{bmatrix}
\]

\[
\begin{align*}
\texttt{A} & = [1\ 1; \ 1\ 2]; \\
\texttt{b} & = [5; \ 0]; \\
\texttt{inv(A) * b} \\
\texttt{ans} = \\
10 \\
-5
\end{align*}
\]

\texttt{inv(A)} produces the inverse of a square matrix. \texttt{inv(A)*b} can also be written \texttt{A\b}, i.e. \texttt{A\b} denotes multiplying on the left by the inverse of \( A \). (Similarly, \texttt{/A} can be used to multiply on the right by the inverse of \( A \).)
Plotting

Plots can be drawn in MATLAB using the `plot` function. The command `plot(x, y)` plots values of the vector `x` on the horizontal axis and values of `y` on the vertical axis.

```matlab
» t = 0:0.1:10;
» f = 5*exp(-0.8*t).*cos(3*t+pi/4);
» plot(t, f)
```

By default, MATLAB draws a solid yellow line. Observe what happens if you enter `plot(t, f, 'c.')` or `plot(t, f, 'gx')`.

If the vector to be plotted is complex, the real and imaginary values are plotted on the horizontal and vertical axes respectively.

```matlab
» z = 2 + 5j;
» plot(z, 'x')
```

i.e., `plot(z)` is equivalent to `plot(real(z), imag(z))`. Another example involving a complex vector:

```matlab
» t = 0:0.001:1;
» f = 3*exp((-2+j*3*pi)*t);
» plot(f)
```

Most of the time, you will use `plot`. However, for a quick plot of a function, `fplot` may be useful. Some examples:

```matlab
» fplot('sin', [0 2*pi])
» fplot('[sin(x) cos(x)]', [0 2*pi])
```

For discrete-time signals, the `stem` function is a useful alternative to `plot`. Try

```matlab
» t = 0:0.1:2;
» f = 5*exp(-0.8*t).*cos(3*t+pi/4);
» stem(t, f)
```

If you use `stem(f)` instead, the plot is made against the indices of the vector `f`.

M-files

To write a new M-file, use the File/New/M-file menu in MATLAB. To open an existing file, use File/Open M-file.

The simplest type of M-file, called a script file, consists of a list of commands such as you typed in the above examples. Create a new M-file and enter the three commands of the previous example. Save the file as, say, `temp.m` (it must have a terminator `.m`). Run the file by typing `temp` at the MATLAB command line prompt.
More sophisticated M-files are functions which can accept variables and also return
variables. Create and save a M-file called sum2.m containing the following text:

\[
\text{function } s = \text{sum2}(a, b) \\
% \text{SUM2 Returns the sum of squares of two inputs} \\
% \text{This is simply a test of writing M file functions} \\
% \text{Written by ........... Date ..........} \\
\]
\[
s = a^2 + b^2; 
\]

Note that the file name must match the function name. Note also the comment lines
starting with \%. Comment lines up to the first non-comment line are returned in response
to the help command. Try help sum2. Test the function by entering commands at the
command line. This M-file works if \(a\) and \(b\) are the same size and square (this includes
the case of \(a\) and \(b\) being scalars). What happens if this is not the case? How would the
function behave if the \(^2\) were replaced by \(^\cdot\)?

Modify this M-file to create one called sum23.m containing the following text:

\[
\text{function } [s, c] = \text{sum23}(a, b) \\
% \text{SUM23 Returns sum of squares and of cubes} \\
% \text{Written by ........ Date ........} \\
\]
\[
a2 = a^2; \\
b2 = b^2; \\
s = a2 + b2; \\
c = a*a2 + b*b2; 
\]

and test it with a command such as \([sq\; cu] = \text{sum23}(2, 3)\). Note how more than one
variable can be returned. Note also that the variables \(a2\) and \(b2\) used inside the function
are \textit{local} to the function (they do not exist outside the function).

\textbf{Control Flow}

You can control the flow of a MATLAB program as you can in a language such as C.
MATLAB contains \texttt{for}, \texttt{if} and \texttt{while} statements. Here is an example of a \texttt{for} loop:

\[
\texttt{>> for } n = 1:5 \\
\texttt{\quad x(n) = } n^*n; \\
\texttt{\quad end} \\
\texttt{\quad x} \\
\texttt{x = } \\
\texttt{\quad 1 4 9 16 25} 
\]

Avoid overusing \texttt{for} loops. Working with vectors directly is often more concise and more
efficient, e.g., the lines above could be replaced by

\[
\texttt{\quad n = 1:5;} \\
\texttt{\quad x = n.*n} \\
\texttt{x = } \\
\texttt{\quad 1 4 9 16 25} 
\]