APPENDIX B

Getting Started with MATLAB

MATLAB software is a computer program that provides the user with a convenient environment for many types of calculations—in particular, those that are related to matrix manipulations. MATLAB operates interactively, executing the user's command one-by-one as they are entered. A series of commands may be saved as a script and run like an interpretive program. MATLAB has a large number of built-in functions; however, it is possible for users to build their own functions made up of MATLAB commands and functions. The primary features of MATLAB are built-in vector and matrix computations including:

- Vector-matrix arithmetic.
- Matrix inversion and eigenvalue/vector analysis.
- Complex arithmetic and polynomial operations.
- Statistical calculations.
- Graphical displays.
- Control system design.
- Fitting process models from test data.

MATLAB has a number of optional toolboxes that provide specialized functions. These include: signal processing, control systems, system identification, optimization, and statistics.

MATLAB is available in versions that run on PCs, Macs, and workstations. The modern version that runs on PCs does so in the Windows environment. The seven exercises that follow are meant to give you the flavor of computing with MATLAB; they do not constitute a comprehensive tutorial. There are additional tutorial materials in the MATLAB manuals. A number of textbooks now feature MATLAB exercises. Also, on-line information is available for any command or function by typing: `help` *`name`*, where *`name`* identifies the command. Do not just look through these exercises; try them all and try variations that occur to you. Check the answers that MATLAB gives and make sure you understand them and they are correct. That is the effective way to learn MATLAB.

1. Assignment of Values to Variable Names

Assignment of values to scalar variables is similar to other computer languages. Try typing

```
a = 4
```

and

```
A = 6
```

Note how the assignment echoes to confirm what you have done. This is a characteristic of MATLAB. The echo can be suppressed by terminating the command line with the semicolon (`;`) character. Try typing

```
b = -3;
```

MATLAB treats names in a case-sensitive manner, that is, the name `a` is not the same as the name `A`. To illustrate this, enter

```
a
```

and

```
A
```

See how their values are distinct. They are distinct names.

Variable names in MATLAB generally represent matrix quantities. A row vector can be assigned as follows:

```
a = [ 1 2 3 4 5 ]
```

The echo confirms the assignment again. Notice how the new assignment of `a` has taken over. A column vector can be entered in several ways. Try them.

```
b = [ 1 ; 2 ; 3 ; 4 ; 5 ]
```

or

```
b = [ 1;
      2;
      3;
      4;
      5 ]
```

or, by transposing a row vector with the `'` operator,

```
b = [ 1 2 3 4 5 ]'
```

A two-dimensional matrix of values can be assigned as follows:

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

or

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

The values stored by a variable can be examined at any time by typing the name alone, for example,

```
b
```

or

```
A
```

Also, a list of all current variables can be obtained by entering the command

```
who
```

or, with more detail, enter

```
whos
```

There are several predefined variables, for example, `pi`.

It is also possible to assign complex values to variables, since MATLAB handles complex arithmetic automatically. To do this, it is convenient to assign a variable name, usually either `i` or `j`, to the square root of −1.

```
i = sqrt(-1)
```

Then, a complex value can be assigned, like

```
x = 2 + i*4
```

2. Mathematical Operations

Operations with scalar quantities are handled in a straightforward manner, similar to computer languages. The common operators, in order of priority, are

```
^      Exponentiation
* /    Multiplication and division
\      Left division (applies to matrices)
+ -    Addition and subtraction
```

These operators will work in calculator fashion. Try

```
2 * pi
```

Also, scalar real variables can be included:

```
y = pi / 4
y ^ 2.45
```

Results of calculations can be assigned to a variable, as in the next-to-last example, or simply displayed, as in the last example.

Calculations can also involve complex quantities. Using the `x` defined above, try

```
3 * x
1 / x
x ^ 2
x + y
```

The real power of MATLAB is illustrated in its ability to carry out matrix calculations. The inner product of two vectors (dot product) can be calculated using the `*` operator,

```
a * b
```

and likewise, the outer product

```
b * a
```

To illustrate vector-matrix multiplication, first redefine `a` and `b`,

```
a = [ 1 2 3 ]
```

and

```
b = [ 4 5 6 ]'
```

Now, try

```
a * A
```

or

```
A * b
```

What happens when the dimensions are not those required by the operations? Try

```
A * a
```

Matrix-matrix multiplication is carried out in likewise fashion:

```
A * A
```

Mixed operations with scalars are also possible:

```
A / pi
```

It is important to always remember that MATLAB will apply the simple arithmetic operators in vector-matrix fashion if possible. At times, you will want to carry out calculations item-by-item in a matrix or vector. MATLAB provides for that too. For example,

```
A ^ 2
```

results in matrix multiplication of `A` with itself. What if you want to square each element of `A`? That can be done with

```
A .^ 2
```

The `.` preceding the `^` operator signifies that the operation is to be carried out item-by-item. The MATLAB manual calls these *array operations*.

When the division operator (`/`) is used with matrices, the use of a matrix inverse is implied. Therefore, if `A` is a square, nonsingular matrix, then `B/A` corresponds to the right multiplication of `B` by the inverse of `A`. A longer way to do this used the *inv* function, that is, `B*inv(A)` ; however, using the division operator is more efficient since `X = B/A` is actually solved as the set of equations `X*A=B` using a decomposition/elimination scheme.

The "left division" operator (`\` , the backslash character) is used in matrix operations also. As above, `A\B` corresponds to the left multiplication of `B` by the inverse of A. This is actually solved as the set of equations A*X=B , a common engineering calculation.

For example, if c is a column vector with values 0.1, 1.0, and 10, the solution of A * x = c , where A has been set above, can be obtained by typing

```
c = [ 0.1 1.0 10 ]'
x = A \ c
```

Try that.

3. Use of Built-In Functions

MATLAB and its Toolboxes have a rich collection of built-in functions. You can use on-line help to find out more about them. One of their important properties is that they will operate directly on vector and matrix quantities. For example, try

```
log(A)
```

and you will see that the natural logarithm function is applied in array style, element by element, to the matrix A. Most functions, like *sqrt, abs, sin, acos, tanh, exp,* operate in array fashion. Certain functions, like exponential and square root, have matrix definitions also. MATLAB will evaluate the matrix version when the letter m is appended to the function name. Try

```
sqrtm(A)
```

A common use of functions is to evaluate a formula for a series of arguments. Create a column vector t that contains values from 0 to 100 in steps of 5,

```
t = [ 0 : 5 : 100 ]'
```

Check the number of items in the t array with the *Length* function,

```
length(t)
```

Now, say that you want to evaluate a formula y = f(t), where the formula is computed for each value of the t array, and the result is assigned to a corresponding position in the y array. For example,

```
y = t .^ 0.34 - log10(t) + 1 ./ t
```

Done! [Note the use of the array operators adjacent decimal points.] This is similar to creating a column of the t values on a spreadsheet and copying a formula down an adjacent column to evaluate y values.

4. Graphics

MATLAB's graphics capabilities have similarities to those of a spreadsheet program. Graphs can be created quickly and conveniently; however, there is not much flexibility to customize them.

For example, to create a graph of the t,y arrays from the data above, enter

```
plot(t, y)
```

That's it! You can customize the graph a bit with commands like the following:

```
title('Plot of y versus t')
xlabel('Values of t')
ylabel('Values of y')
grid
```

The graph appears in a separate window and can be printed or transferred via the clipboard (PCs with Windows or Macs) to other programs.

There are other features of graphics that are useful, for example, plotting objects instead of lines, families of curves plots, plotting on the complex plane, multiple graphs windows, log-log or semilog plots, three-dimensional mesh plots, and contour plots.

5. Polynomials

There are many MATLAB functions that allow you to operate on arrays as if their entries were coefficients or roots of polynomial equations. For example, enter

```
c = [ 1 1 1 1 ]
```

and then

```
r = roots(c)
```

and the roots of the polynomial $x^3 + x^2 + x + 1 = 0$ should be printed and are also stored in the `r` array. The polynomial coefficients can be computed from the roots with the *poly* function,

```
poly(r)
```

and a polynomial can be evaluated for a given value of `x`. For example,

```
polyval(c, 1.32)
```

If another polynomial, $2x^2 - 0.4x - 1$, is represented by the array `d`,

```
d = [ 2 -0.4 -1 ]
```

the two polynomials can be multiplied symbolically with the convolution function, *conv,* to yield the coefficients of the product polynomial,

```
cd = conv(c,d)
```

The deconvolution function, *deconv,* can be used to divide one polynomial into another, for example,

```
[ q,r ] = deconv(c,d)
```

The `q` result is the quotient, and the `r` result is the remainder.

There are other polynomial functions that may become useful to you, such as the residue function for partial fraction expansion.

6. Statistical Analysis

The Statistics Toolbox contains many features for statistical analysis; however, common statistical calculations can be performed with MATLAB's basic function set. You can generate a series of (pseudo)random numbers with the *rand* function. Either a uniform or normal distribution is available:

```
rand('normal')
n = 0 : 5 : 1000 ;
```

(Did you forget the `;` !!!)

```
num = rand(size(n)) ;
```

You probably understand why using the semicolon at the end of the commands above is important, especially if you neglected to do so.

If you would like to see a plot of noise, try

```
plot(num)
```

These are supposed to be normally distributed numbers with a mean of zero and variance (and standard deviation) of one. Check by

```
mean(num)
```

and

```
std(num)
```

No one is perfect! You can find minimum and maximum values,

```
min(num)
max(num)
```

There is a convenient function for plotting a histogram of the data:

```
hist(num,20)
```

where 20 is the number of bins.

If you would like to fit a polynomial to some data by least squares, you can use the *polyfit* function. Try the following example:

```
t = 0 : 5
y = [ -0.45 0.56 2.34 5.6 9.45 24.59 ]
coef = polyfit( t, y, 3 )
```

The values in `coef` are the fitted polynomial coefficients. To generate the computed value of `y`,

```
yc = polyval( coef,t )
```

and to plot the data versus the fitted curve,

```
plot ( t,yc,t,y,'o' )
```

The plot of the continuous curve is piecewise linear; therefore, it does not look very smooth. Improve this as follows:

```
t1 = [ 0 : 0.05 : 5 ] ;
yc = polyval(coef, t1)
plot(t1, yc, t, y, 'o')
```

7. This and That

There are many, many other features to MATLAB. Some of these you will find useful; perhaps others you will never use. We encourage you to explore and experiment.

To save a copy of your session, MATLAB has a useful capability called *diary*. You issue the command

```
diary problem1
```

and MATLAB opens a disk file in which it stores all the subsequent commands and results (not graphs) of your session. You can turn the diary feature off:

```
diary off
```

and back on with the same file:

```
diary on
```

After you leave MATLAB, the diary file is available to you. It is common to use an editor or word processor to clean up the diary file (getting rid of all those errors you made before anyone can see them!) and then print the file to obtain a hard copy of the important parts of your work session, for example, key numerical results.

Exit MATLAB with the `quit` or `exit` commands. It is possible to save the current state of your work with the `save` command. It is also possible to reload that state with the `load` command.