A MATLAB TUTORIAL
Dr. L. Doyle and Dr. A. Kokaram,
ledoyle@tcd.ie and anil.kokaram@tcd.ie
Department of Electronic and Electrical Engineering,
University of Dublin Trinity College.
Contents

1 Read this first
   1.1 About Matlab ......................................................... 3
   1.2 Running Matlab ..................................................... 4
      1.2.1 Where is Matlab installed? ................................. 5
   1.3 Confidence Check .................................................... 5
   1.4 Current Directory ................................................... 6

2 Useful Matrix Notation ............................................... 7
   2.1 Getting Started ....................................................... 7
   2.2 Basic Matrices ...................................................... 7
   2.3 Indexing elements .................................................. 9
   2.4 Short Cuts ........................................................... 10
      2.4.1 Transpose ...................................................... 11
   2.5 The Dot notation ................................................... 11
   2.6 Functions ........................................................... 12
   2.7 Multidimensional Arrays ........................................... 12

3 Data Input/Output ..................................................... 13
   3.1 Mat and ASCII files ............................................... 13
   3.2 Raw Data I/O ........................................................ 14
      3.2.1 Short Cut I/O .................................................. 14
4 Scripts and Functions

4.1 Introduction ................................................................. 16
4.2 Scripts .................................................................. 17
4.3 Functions ................................................................. 17
  4.3.1 Global Variables ...................................................... 18

5 MATLAB Editor/Debugger .................................................. 20

6 Plotting ...................................................................... 21

6.1 Overview .................................................................. 21
6.2 Examples .................................................................. 21
6.3 PRINTING Plots .......................................................... 22
6.4 An Example Program ................................................... 22
6.5 A Program for Plotting Equations ................................. 23

7 TOOLBOXES ................................................................ 26

7.1 Toolboxes available in EEE ........................................... 26

8 Image Display and Manipulation ........................................ 28

8.1 Image I/O .................................................................. 28
8.2 Display ...................................................................... 29
8.3 Aspect Ratio ............................................................... 29
8.4 Reports .................................................................... 30

9 Efficient Matlab ................................................................ 31

9.1 Image Enhancement ...................................................... 32
9.2 Histogram Modification ................................................ 34
Chapter 1

Read this first

This document is a self help guide for Matlab 5. It is to be used as part teach-yourself sessions which expose you to Matlab programming concepts. It guides you through the most useful basic instructions in Matlab and also eventually leads you through an exercise which shows you how to write efficient code in Matlab.

It would help if you know some programming concepts already, and this document is aimed at the reader who has already taken at least three years of study at university in some numerate science or engineering discipline.

In this document >> indicates the Matlab command line prompt and characters or instructions which follow this are to be typed into the command line.

You are directed to follow through this document in sequence in order to learn a useful set of Matlab tools. However, the document could be used as a reference for common Matlab commands. Nevertheless you should note that Matlab has excellent help built into the package itself. There is an excellent on line guide to programming etc.

See >> help or >> help eye to view typical Matlab help.

You should be able to complete all the initial exercises in 3 hours. The case studies should take about 5 hours all together.

The document also indicates things that you should do with the symbol □.

1.1 About MATLAB in the EEE Department

Matlab (short for Matrix Laboratory) is a software package which was designed initially for use as a tool for signal processing research. It is now one of the most popular tools for signal
analysis in industry as well as research and education. It allows the user to manipulate signals using high level commands such as matrix multiply and convolution. The user is thus spared the tedious chore of writing low level C programs for these fundamental operations which everyone uses. Matlab also has a series of extremely easy to use Graphical User Interface commands which allow images to be displayed, graphs to be plotted etc. Programs written in Matlab are stored in files called Matlab scripts and they contain Matlab commands. Matlab script files are recognized by having the extension .m after the filename.

1.2 Running Matlab

To run Matlab, there are several options. It is assumed that the package is already installed. If you are a student at home, then you would have installed the package yourself, otherwise the computer officer in the Department would have installed the package already.

First of all you have to identify where the Matlab package can be invoked. This is done by locating the Matlab icon.

1. If you are working from a general purpose EEE Department computer e.g. in the Intel Lab in the Printing House, then the Matlab icon should either be on your desktop or in the Start→Programs menu.

2. If you are a 4th year or taught MSc student, then your default desktop should contain the Matlab icon.

3. If you are a PhD or MSc student then provided you have been allocated a default desktop, the icon should be on the desktop. If not then check the Start→Programs menu.

4. If all else fails, then navigate to x:/matlabr11/bin to find the Matlab icon.

If you cannot locate the icon, contact the computer officer responsible for the machines where you are working.

Now on an NT system, just double click on the Matlab icon to start Matlab.

On UNIX, typing matlab at the prompt should work. If this latter does not work, then contact your local sys admin\footnote{Tell him that the UNIX Matlab licence server is at maxwell.mee.tcd.ie}.
1.2.1 Where is Matlab installed?

In the EEE Department we run Matlab off of our network. We have binaries for all types of computing platforms including Windows/Intel, Linux/Intel, IRIX/SGI, UNIX/SUN, UNIX/HP, UNIX/RS6000.

We are allowed to run 40 concurrent invocations of Matlab on windows systems and 10 invocations on UNIX based systems. The Matlab software manages this by talking to a piece of software called a License Server whenever it is first invoked. The licence server then checks to see how many licences have been checked out and then assigns one to the new invocation if required.

This means, that although Matlab can be installed on every machine in the Department, it still needs to check with the license server before it allows the program to completely start up. If it cannot contact the licence server, or if there are not enough licences to go around, then you will be unable to use Matlab.

Installing Matlab on every machine in the Department is a waste of time and space. What has been done is to install it on one machine: the file server. In the Department NT domain this is called EESERVER, and matlab is installed there in //EESERVER/APPS/MATLABR11/BIN or x:/apps/matlabr11/bin.

Therefore, PC users in the Department, can log into the mee.tcd.ie domain, and so connect to the server and then run Matlab from that location without ever having to bother installing it.

Another superb reason for having just one installation is that we always keep that version up to date and it is always maintained. Therefore, you will find that the central EEE Department matlab installation is always the best thing to use.

On UNIX platforms, the license server is maxwell.mee.tcd.ie. Because we have several different types of UNIX machines though e.g. SUN, SGI, IBM; which all require different Matlab binaries, we have been forced to have a small number of Matlab installations on various machines. However, they all use one central license server.

1.3 Confidence Check

To make sure that Matlab has been invoked ok; after you start it do the following

□ >> bench

Matlab will run a series of numerical tasks which will take your machine through its paces
and then it will rank the machine against several others in a speed comparison. Its fun to see how fast your 1,000 pound Intel machine is compared to a 10,000 pound Sun workstation.

1.4 Current Directory

By default Matlab writes and saves files to the current directory. The current directory is shown using \texttt{pwd} (Present Working Directory).

You can navigate around your directories using \texttt{cd directoryname} . Alternatively you can navigate around using the Folder icon in the top menu of the Matlab command window. This icon shows up as two yellow folders in the top bar.
Chapter 2

Useful Matrix Notation

2.1 Getting Started

The Matlab interface is a command line interface rather like most BASIC environments. However Matlab works almost exclusively with matrices: scalars simply being 1-by-1 matrices. At the most elementary level, Matlab can be a glorified calculator:

```matlab
>> fred=6*7
>> FRED=fred*j;
>> exp(pi*FRED/42)
>> who
>> whos
```

Matlab is case sensitive so that the two variables FRED and fred are different. The result of an expression is printed out unless you terminate the statement by a ';'. j and i represent the square root of -1 unless you define them otherwise. lists the current environment variable names and provides details such as size, total number of elements and type. Note that the variable is set if you type a statement without an ‘=’ sign.

2.2 Basic Matrices

So far you have operated on scalars. Matlab provides comprehensive matrix operations:

```matlab
>> a=[3 2 -1 0 3 2 1 -3 4]
>> b=[2,-2,3 ; 1,1,0 ; 3,2,1]
```
>> b(1,2)
>> a*b
>> det(a)
>> inv(b)
>> (a*b)'-b'*a'
>> sin(b)

The above shows two ways of specifying a matrix. The commas in the specification of b can be replaced with spaces. Square brackets refer to vectors and round brackets are used to refer to elements within a matrix so that \( b(x,y) \) will return the value of the element at row \( x \) and column \( y \). Matrix indices must be greater than or equal to 1. \texttt{det} and \texttt{inv} return the determinant and inverse of a matrix respectively. The ' performs the transpose of a matrix. It also complex conjugates the elements. Use .’ if you only want to transpose a complex matrix. The * operation is interpreted as a matrix multiply. This can be overridden by using .* which operates on corresponding entries:

>> c=a.*b
>> d=a./b
>> e=a.^b

For example \( c(1,1) \) is \( a(1,1)\times b(1,1) \). The \texttt{Inf} entry in \( d \) is a result of dividing 2 by 0. The elements in \( e \) are the results of raising the elements in \( a \) to the power of the elements in \( b \). Matrices can be built up from other matrices:

>> big=[ones(3), zeros(3); a , eye(3)]

\( \text{big} \) a 6-by-6 matrix consisting of 3-by-3 matrix of 1’s, a 3-by-3 matrix of 0’s, matrix and the 3-by-3 identity matrix.

It is possible to extract parts of a matrix by use of the colon:

>> big(4:6,1:3)

This returns rows 4 to 6 and columns 1 to 3 of matrix \( A \). A colon on its own specifies \emph{all} rows or columns:

>> big(4,:)
>> big(:,)
>> big(:,[1,6])
>> big([3:5],[1,4:6])
The last two examples show how vectors can be used to specify which non-contiguous rows and columns to use. For example the last example should return columns 1, 4, 5 and 6 of rows 3, 4 and 5.

In general Matlab allows you to manipulate Matrices according to the normal rules. Therefore,

\[
\text{>> } w = [5;1] = [1 2; 3 4]; \\
\text{>> } l*w \\
\text{>> } l*l
\]

yields no surprises. Matlab is unhappy with

\[
\text{>> } l-w
\]

as expected.

The following is very useful indeed:

\[
\text{>> } k = l(:)
\]

You will see that this converts the matrix \(l\) into a vector formed by scanning the matrix in a columnwise fashion.

### 2.3 Indexing elements

Matrix indexing for multidimensional arrays is done in the normal way i.e. in row,column format. Thus \(\text{>> } A(j,i)\) is the element at the \(j\)th row of the \(i\)th column. It is also useful to note that Matlab allows you to index matrices using a single index. To understand how this works, you must first appreciate that Matlab stores arrays of numbers in a contiguous memory space starting from the top left of the array and working down to the bottom right by scanning each column in order from left to right. Thus the matrix

\[
\text{>> } A = [1 2 3; 4 5 6; 7 8 9]
\]

is actually stored in memory in the following continuous sequence 1 4 7 2 5 8 3 6 9.

This means that typing

\[
\text{>> } A(7)
\]

actually works, and yields the value 3 in this case, which is element \(A(1,3)\).
Those familiar with C will recognize this referencing as the typical mechanism to reference a 2D array stored as a 1-D raster scanned list of values. In this case $A(j, i) = A((i - 1) * 3 + j)$.

**Important**

In Matlab the first element of a vector is the element number 1. This is different from C in which the first element is element number 0. Thus in Matlab a $3 \times 3$ array has entries from $(1, 1)$ to $(3, 3)$, whereas in C the same array will have entries from $(0, 0)$ to $(2, 2)$.

Try the following

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

What’s happening?

As you can see, you can reference particular elements of an array using vector arguments.

### 2.4 Short Cuts

Sometimes you need to generate vectors of indices. You do this with the : notation. See what the following do.

```matlab
>> x = (1:10)
>> y = (1:2:10)
```

You should find that $x = [1 2 3 4 5 6 7 8 9 10]$ and $y = [1 3 5 7 9]$. The : separates the start value, the step (which can be non-integer) and the end value.

Generate the following vector in Matlab $[7 6.5 6 5.5 5 4.5 4 3.5 3]$.

$x = 7:-0.5:3$
2.4.1 Transpose

To transpose a matrix use ‘.’. Thus if \( a \) is a \( n \times 1 \) vector then

\[
\begin{align*}
    b &= a'; \\
    \text{causes } b \text{ to be } 1 \times n \text{ elements.}
\end{align*}
\]

2.5 The Dot notation

By now you know that Matlab allows you to manipulate matrices as symbolic objects. However there are often times when you need to operate on single elements in those arrays. For instance, say you need to calculate \( \sum_{i=1}^{N-1} a_i^2 \) where \( a_i \) is the \( i \) th element of vector \( a \).

You may be immediately reaching for your C book and thinking something like

\[
\begin{align*}
    >> s &= 0; \\
    >> \text{for } i &= 1:N \\
    >> \quad s &= s + a(i)^2; \\
    >> \text{end};
\end{align*}
\]

However, this is very very slow in Matlab as each line is interpreted separately. Matlab does however allow you to do this much faster with the very useful ‘.’ notation. Therefore the operation above is done simply by writing

\[
    s = \text{sum}(a.^2)
\]

Which takes the sum of the vector \( a.^2 \). This latter command squares each of the elements of \( a \) separately. It is very different from \( a'*a \), although in this case the result is the same.

Use >> help sum to find out what the Matlab function \texttt{sum} does.

To summarize: The dot notation tells matlab to perform the mathematical function appearing right after the the dot, on each element of the matrix separately.

\[
\begin{align*}
    \text{View the matrices } & B, C, D. \text{ Why are they different?}
\end{align*}
\]
Use the dot notation to calculate $\sum_{ij}(a_{ij} - a_{1ij})^2$ and check your answer.

Use the dot notation to calculate $\sum_{ij} a_{ij}/a_{1ij}$ (Hint: Dot notation works with matrix manipulation too.)

### 2.6 Some useful Matrix functions

Matlab allows you to take matrix inverse with the `inv` function. You should just quickly scan the help for the following functions and operators.

```
>> help /
>> help \n
>> help eye
>> help svd
>> help zeros
>> help ones
```

### 2.7 Multidimensional Arrays

Matlab can also handle multidimensional arrays. A nice example is a movie array. This consists of a series of 2D image frames (matrices) corresponding to discrete moments in time. Here is a simple sequence of 3, 2x2 image frames, allocated using Matlab notation.

```
>> sequence = rand(2,2,3);
>> sequence
```

Thus `>> sequence(:,2,3)` is the 2nd column of the 3rd 2D array. And `>> sequence(:,,:,2)` is the whole 2nd array.

You can confuse yourself with as many array dimensions as you like, so `>> stuff = ones(3,4,1,6)` produces the relevant array. Of course you are advised to use these kinds of structures with caution.

Note once again that the Matlab help (see the Help menu) is very good on this topic.
Chapter 3

Data Input/Output

There are several ways of getting data in and out of the Matlab environment.

First of all you should note that Matlab can read and write many popular multimedia file formats.

Try the following

```matlab
>> help wavread
>> help wavwrite
>> help auread
>> help imread
>> help imwrite
```

You can also play wav files over the PC speakers through the sound card. The function to do this is `sound`.

Load in the data in the wav file at `y:/sounds/ape.wav` into a vector `f`. Then play it through the speakers using Matlab.

You will look at image display and manipulation in the next chapter.

3.1 Mat and ASCII files

Matlab allows you to read and write data in Matlab file format or ASCII format using the `save` command.

```matlab
>> help save
```
Typing `save` on its own, saves the entire workspace of variables to a file called `matlab.mat` in the current directory. Also, the command `load` restores the variables from a file of the same name in the current directory.

```
save <filename> x,t,y
```
saves the workspace variables `x`, `t`, `y` to the file `<filename>`.

Matlab also allows you to change the accuracy and type of variables which are stored to the file. Therefore, `save <filename> x -ASCII` uses ascii. And similarly `load <filename> -ascii` loads a list of ASCII numbers into a variable with the same name as the `<filename>`.

## 3.2 Raw Data I/O

Matlab allows you access to low level i/o functions like `fopen`, `fread`, `fwrite`, `fclose`, `fseek`, in the same way as C. When you open a file, you open it for reading or writing binary or text data, and Matlab returns to you a file handle which you then use for reading or writing the data from/to a file. You can use `fseek` to read or write to arbitrary points in a file. Try the following:

```
>> fin = fopen('y:\sequences\caltrain.big','b');
>> pic = fread(fin,[720 576],'uchar');
>> image(pic'); colormap(gray(256)); shg;
>> fclose(fin);
```

This is an image example, but it of course works just as well for any kind of data, as long as you know the data storage format beforehand. You can find out about `fwrite`, `fseek` etc as you write your own programs. Note that Matlab Help is again very good here.

### 3.2.1 Simpler raw data manipulation

To help you open, seek, read and write data; a couple of short cut .m functions have been written. These are called `fread3`, `fwrite3` and `freadack`. They are NOT standard Matlab functions. You do not need to know where they are, just that they work if invoked from the Matlab installation in this Department.

The functions pack the `fopen`, `fread`/`fwrite`, `fclose` commands into one unit, thus saving you some time. Use the help to see the arguments that the programs take. The same example above can be done as follows.
>> pic = fread3('y:\sequences\caltrain.big',0,0,720,576);
>> image(pic'); colormap(gray(256)); shg;

Again, take note of the data storage format codes. help fread3
Chapter 4

Scripts and Functions

4.1 Introduction

Matlab provides the for, while, if constructs. Matlab will wait for you to type the statement before it executes the construct.

```matlab
>> for x=1:1:10
    y(x)=x;
end
>> y
>> n=10;
>> while(n ~= 0)
    n=n-1
end
>> if (n<0)
    -n
elseif (n==0)
    n=365
else
    n
end
```

The three numbers in the for statement specify start, step and end. ~= is 'not equal to' and == is 'equivalent to'.

It is tedious to work in this fashion. To overcome this MATLAB allows you to write your own programs. Files that contain code in the MATLAB language are called M-files. You
CHAPTER 4. SCRIPTS AND FUNCTIONS

4.2 Scripts

Scripts are M-files that do not accept input arguments or return output arguments. They operate on data in the workspace. When you invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. For example, create a file called `magicrank.m` that contains these MATLAB commands:

```matlab
% Investigate the rank of magic squares (% symbol denotes comment)
r = zeros(1,32);
for n = 3:32
    r(n) = rank(magic(n));
end
r
bar(r)
```

Typing the statement

```
>> magicrank
```

causes MATLAB to execute the commands, compute the rank of the first 30 magic squares, and plot a bar graph of the result. After execution of the file is complete, the variables `n` and `r` remain in the workspace.

4.3 Functions

Functions are M-files that can accept input arguments and return output arguments. The name of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt. Here is an example of a function

```matlab
% Investigate the rank of magic squares (% symbol denotes comment)
r = zeros(1,32);
for n = 3:32
    r(n) = rank(magic(n));
end
r
bar(r)
```
function r = rank(A,tol)
% RANK Matrix rank.
% RANK(A) provides an estimate of the number of linearly
% independent rows or columns of a matrix A.
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
    tol = max(size(A)) * max(s) * eps;
end
r = sum(s > tol);

The first line of a function M-file starts with the keyword function. It gives the function
name and order of arguments. In this case, there are up to two input arguments and one
output argument. The first line of the help text is the H1 line, which MATLAB displays
when you use the look for command or request help about the file. The rest of the file is the
executable MATLAB code defining the function. The variables are all local to the function;
they are separate from any variables in the MATLAB workspace.

This example illustrates one aspect of MATLAB functions that is not ordinarily found in
other programming languages - a variable number of arguments. The rank function can be
used in several different ways:

\[ \gg \quad \text{rank}(A) \]
\[ \gg \quad r = \text{rank}(A) \]
\[ \gg \quad r = \text{rank}(A,1.e-6) \]

Many M-files work this way. If no output argument is supplied, the result is stored in
ans. If the second input argument is not supplied, the function computes a default value.
Within the body of the function, two quantities named \texttt{nargin} and \texttt{nargout} are available
which tell you the number of input and output arguments involved in each particular use of
the function. The rank function uses \texttt{nargin}, but does not need to use \texttt{nargout}.

\subsection{Global Variables}

If you want more than one function to share a single copy of a variable, simply declare the
variable as global in all the functions. Do the same thing at the command line if you want the
base workspace to access the variable. The global declaration must occur before the variable is actually used in a function. Although it is not required, using capital letters for the names of global variables helps distinguish them from other variables. For example, create an M-file called `falling.m`:

```matlab
function h = falling(t)
    global GRAVITY
    h = 1/2*GRAVITY*t.^2;
```

Then interactively enter the statements:

```matlab
>> global GRAVITY
>> GRAVITY = 32;
>> y = falling((0:.1:5)');
```

The two global statements make the value assigned to GRAVITY at the command prompt available inside the function. You can then modify GRAVITY interactively and obtain new solutions without editing any files.
Chapter 5

MATLAB Editor/Debugger

MATLAB has an Editor/Debugger for creating and debugging these files. To start the Editor/Debugger, opening it to a particular file, select Open from the File menu in the Command Window, or click the Open File (folder icon) button on the toolbar.

The MATLAB Debugger helps you identify programming errors in your MATLAB code. Using the Debugger, you can:

1. View the contents of the workspace at any time during function execution
2. View the function call stack
3. Set breakpoints in code
4. Execute M-file code line by line

Full details of Editor/Debugger use can be found at:

Chapter 6

Plotting

6.1 Overview

There are very many ways in which to plot information in MATLAB. Everything you need to know about plotting can be found by typing the following:

```matlab
>> help graph2d
>> help graph3d
>> help specgraph
>> help graphics
```

6.2 Examples

Basic Example 2D

```matlab
>> x = rand(1,100);
>> y = (1:1:100);
>> plot(x,y);
>> grid;
>> xlabel('TIME');
>> ylabel('SIGNAL STRENGTH');
>> title('Plot of TIME versus SIGNAL STRENGTH');
>> hold
>> x2 = rand(1,100);
>> plot(x,y,'r');
```
Visualising Functions of Two Variables

To display a function of two variables, \( z = f(x, y) \), generate \( X \) and \( Y \) matrices consisting of repeated rows and columns, respectively, over the domain of the function. Then use these matrices to evaluate and graph the function. The \texttt{meshgrid} function transforms the domain specified by a single vector or two vectors \( x \) and \( y \) into matrices \( X \) and \( Y \) for use in evaluating functions of two variables. The rows of \( X \) are copies of the vector \( x \) and the columns of \( Y \) are copies of the vector \( y \). Example: To evaluate the two-dimensional sinc function, \( \sin(r)/r \), between \( x \) and \( y \) directions:

\[
\begin{align*}
&>> \ [X,Y] = \text{meshgrid}(-8:.5:8); \\
&>> R = \text{sqrt}(X.^2 + Y.^2) + \text{eps}; \\
&>> Z = \sin(R)./R; \\
&>> \text{mesh}(X,Y,Z)
\end{align*}
\]

Other Example - 3D

\[
\begin{align*}
&>> t = 0:pi/10:2*pi; \\
&>> [X,Y,Z] = \text{cylinder}(4*cos(t)); \\
&>> \text{subplot}(2,2,1) \\
&>> \text{mesh}(X) \\
&>> \text{subplot}(2,2,2); \text{mesh}(Y) \\
&>> \text{subplot}(2,2,3); \text{mesh}(Z) \\
&>> \text{subplot}(2,2,4); \text{mesh}(X,Y,Z)
\end{align*}
\]

6.3 PRINTING Plots

The \texttt{Print} option on the File menu and the print command both print MATLAB figures. The Print menu brings up a dialog box that lets you select common standard printing options. The print command provides more flexibility in the type of output and allows you to control printing from M-files. The result can be sent directly to your default printer or stored in a specified file. A wide variety of output formats, including PostScript, are available.

6.4 An Example Program

Consider the following problem: Display the 2D Fourier transform intensity of a square slit.
Examining the diagram, we observe that the desired path involves navigating through a series of intersections. Each intersection is represented by a node, and the edges connect these nodes. The path starts at the initial node and ends at the final node. The edges are labeled with distance values, indicating the length of each segment of the path.

To determine the shortest path, we can use Dijkstra's algorithm. This algorithm iteratively selects the node with the smallest unvisited distance and expands it, updating the distances of its neighbors if a shorter path is found. By repeating this process, we can find the shortest paths from the source node to all other nodes in the network.

In the given example, the graph is represented as an adjacency matrix, where each element $a_{ij}$ denotes the direct distance from node $i$ to node $j$. The algorithm then traverses the graph, calculating the cumulative distance from the starting node to each node it visits.
%user Defined Polynomial Function
[rows,cols] = size(x);
for index=1:cols,
    y(index) = 4*x(index)^3+2*x(index)^2-200*x(index)-50;
end;

The first line defines a 'eqn' as a function: the square brackets enclose the comma separated output variable(s) and the round brackets enclose the comma separated input variable(s). The % in the second line means that the rest of the line is a comment. However, as the comment comes immediately after the function definition, it is displayed if you type help eqn.

The function anticipates x being a row vector so that size(x) is used to find out how many rows and columns there are in x. You can check that the root is close to 7.0 by:

>> eqn([6.0:0.5:8.0])

Note that eqn requires an argument to run which is the vector [6.0 6.5 7.0 7.5 8.0].

The for loop in Matlab should be avoided if possible as it has a large overhead. can be made more compact using the ‘.’ notation. Delete the lines in eqn.m and replace them with:

function [y] = eqn(x)
%COMPACT user Defined Polynomial Function
y = 4*x.^3 + 2*x.^2-200*x-50

Now if you type >> eqn([6.0:0.5:8.0]) it should execute your compact eqn.m file.

Edit and save in your matlab ploteqn.m directory:

x_est = 7.0;
delta = 0.1;
while(delta > 1.0e-4)
    x = x_est-delta:delts/10:x_est+delta;
    fplot('eqn',x);
    disp('mark position of root with mouse button')
    [x_est,y_est] = ginput(1)
    delta=delta/10;
end;
This uses the function `fplot` to plot\(^1\) the equation specified by function `eqn.m` between the limits specified by `x`. `ginput` with an argument of 1 returns the x- and y-coordinates of the point you have clicked on. The routine should zoom into the root with your help. To find the actual root enter:

```matlab
>> poly = [4 2 -200 -50];
>> format long
>> roots(poly)
>> format
```

which will print all the roots of the polynomial: \(4x^3 + 2x^2 - 20x - 50 = 0\) a 15 digit format. `format` on its own returns to the 5 digit default.

\(^1\)Note that the function ‘plot’ is an alternative.
Chapter 7

TOOLBOXES

7.1 Toolboxes available in EEE

MATLAB Toolboxes are collections of M-files that extend MATLAB’s capabilities to a number of technical fields. We have the following toolboxes on the system in the department.

1. Signal Processing
   This toolbox contains a customizable framework for digital signal processing (DSP) and is based on filter design and spectral analysis techniques. The toolbox contains tools for algorithm development, signal and linear system analysis, and time-series data modeling.

2. Communications
   This toolbox contains a set of tools for the design, analysis, and simulation of digital and analog communication systems.

3. Image Processing
   This toolbox provides a suite of robust digital image processing and analysis functions.

4. Wavelet
   The Wavelet Toolbox provides a collection of routines for examining local, multiscale, and nonstationary phenomena. Wavelet methods offer additional insight and performance in any application where Fourier techniques have been used.

5. Statistics
The Statistics Toolbox provides an environment for analyzing historical data, modeling systems to predict their behavior, developing statistical algorithms, and learning and teaching statistics.

6. **Optimization**

The Optimization Toolbox extends the MATLAB environment to provide tools for general and large-scale optimization of nonlinear problems. Additional tools are provided for linear programming, quadratic programming, nonlinear least-squares, and solving nonlinear equations.

7. **System Identification**

The System Identification Toolbox provides tools for creating mathematical models of dynamic systems based on observed input/output data. The Toolbox features a flexible graphical user interface that aids in the organization of data and models.

Typing `help` and the toolbox name gives a list of all the functions available in that toolbox. For example typing

```matlab
>> help wavelet
```

gives help on the Wavelet Toolbox.
Chapter 8

Image Display and Manipulation

Images are always manipulated as arrays of numbers. The smallest element of an image is called a Pixel, and for an 8bit grey scale image each Pixel is an integer between 0 and 255. Matlab has an image processing toolbox which allows you to do a large number of image manipulation functions. The most useful facility however is the number of functions for reading and writing different image types. Do >> help images/images for a list of functions in the image processing toolbox.

Even without the image processing toolbox, Matlab is still very useful for image manipulation because there is an inbuilt image read function and image display capacity.

8.1 Reading and Writing Images

The Chapter on Data I/O also suffices to describe image reading and writing. However the functions imread and imwrite specifically allow you to read and write different image file formats.

For raw image data, stored as raster scans (like many of those in y:\sequences and y:\images, the usual fopen, fread, fwrite, fread3, freadack etc suffice.

As a reminder do the following (for raw data).

□

>> pic = fread3(‘y:\sequences\caltrain.big’,0,0,720,576)’;
>> image(pic’); colormap(gray(256)); shg;

The image is now loaded as an array pic. The transpose at the end of the fread3
statement is required because the raw data images in y:\sequences etc are stored as raster scans of raw data, from top left to bottom right scanning each line from left to right. To read this data into Matlab’s column wise scanning system, the picture has to be read into a transposed form.

To write out the data, you could use `imwrite` and write it in any of several formats e.g. jpeg, bmp etc. Alternatively, you could write out the data again as a raw format.

```matlab
>> fwrite3(pic,’mypic.raw’,0,0);
```

### 8.2 Display

Matlab allows Display through the command `image` and you create different windows for display using the `figure` command.

```matlab
>> image(pic’); colormap(gray(256)); shg;
```

Using `image` displays the array argument in the current figure window. The `colormap` command sets the RGB colour of each of the possible 256 values of each pixel element for the current figure window. `shg` shows the current figure window.

```matlab
>> gray(256)
```

As you can see from the above, the colormap is a $256 \times 3$ array, assigning particular RGB combinations to each of 256 entries. Note that the allowed range of RGB values in Matlab are between 0 and 1.

### 8.3 Aspect Ratio

It is often the case that you require pixels to be displayed as square. This is done with the `axis image` command.

```matlab
>> figure(2); image(pic’); colormap(gray(256)); axis image; axis off; shg;
```

The `axis off` command removes the axis labels from the image, which are often superfluous.
8.4 Creating images for Reports

It is often the case that you need to save the displayed image in a figure window in some file format which can be included in an electronic document. If you are using Ms Word then of course, saving your images as .bmp would be best. For other packages, like \LaTeX, encapsulated postscript is best (.eps files). Printing a figure to a file is done using the \texttt{print} command. Thus

\begin{verbatim}
>> figure(2);
>> print -deps mypic.eps
\end{verbatim}

prints the plot or image in the current figure window into a .eps file.

A better way of doing this is

\begin{verbatim}
>> figure(2);axis image;axis off;
>> set(gca,'position',[0.01 0.01 0.98 0.98]);
>> print -deps mypic.eps
\end{verbatim}

The \texttt{set(gca)} command allows the image to occupy the full area of the figure and gives a more complete usage of the space allocated for the image in the document.

\texttt{help gca} and \texttt{help gcf} for more about Handle Graphics in Matlab. Handle Graphics are an extremely powerful aspect of Matlab, it allows you to change the font used on axes labels, the thickness of lines etc. This is especially important for electronic publication of figures and images.
Chapter 9

Efficient Matlab

Matlab should be thought of as a tool for testing ideas quickly and not necessarily the best tool for implementing a solution to a problem. For instance it is highly unlikely that you would wish to implement a decimation front end to a software radio using Matlab running on a StrongArm processor in a HP Journada. However, you can observe the behavior of many types of different front end structures from an algorithmic viewpoint in Matlab and even test for speed in some cases.

You will therefore be using Matlab primarily for the ease with which it is possible to write and analyse relatively complex algorithms and not for the speed at which Matlab will run that quickly written program. Given this, it will seem strange that it is worthwhile thinking about efficient Matlab code. Nevertheless, one of the most annoying features of Matlab development is that what seems like simple code, takes such a long time to run. With some very simple tricks and some attention to tailoring your code to the things which Matlab does best, you can write very efficient and fast running code in Matlab.

This chapter therefore starts with some pointers to things which you should look out for when writing .m scripts and functions. Then there is a design exercise in which you are given a task and asked to write a .m file to implement it as efficiently as possible.

**Tips for Efficient Matlab**

1. Avoid `for` loops
2. Use Matrix manipulation as much as possible
3. Remember that you can index Matrices using vectors of indexes
4. Beware of using the more complex Matlab functions blindly; they may not be very efficient, and you can sometimes write your own more quickly.
5. Allocate variables at the start of your program. (More to do with programming style than anything else).


7. Warning: Guard against making your code so compact that it is hard to understand. Adopt the 10 minute rule: *If you can’t see how to do something efficiently in Matlab in 10 minutes, then implement it any way you can.*

### 9.1 Case Study: Stretching Grey Scale

The program `ackstretch1.m` at `w:\isd\matlab\tutorial\` implements a simple enhancement technique which stretches the grey scale values of an 8 bit input image to occupy the entire range from 0 to 255. It is written extremely inefficiently.

The algorithm is as follows

1. Measure the minimum (mn) and maximum (mx) values of the input image.
2. Map each pixel in the input image onto 0 to 255 using \( y = m(x - mn) \) where \( m = 255/(mx - mn) \) and \( y, x \) are the values of the input and output pixels respectively.

□ Rewrite the program efficiently. (Hint: All the for loops can be removed. See also Matlab functions `min`, `max`, `round`.

□ What is the improvement in speed with your efficient code? >> `help etime`

The program is as below.

```matlab
%read in the data
temp = imread('w:\isd\matlab\tutorial\sail_poor_contrast.bmp','bmp');
%The imread function loads in data as UINT8 data type, you need to
%convert this to DOUBLE to be able to do Maths on it
pic = double(temp);

%display it in figure 1
figure(1);image(pic);colormap(gray(256));axis image;axis off;
title('Original image');

%get size of input image
```
[vres,hres] = size(pic);

%now measure max and min values
mn = pic(1,1); mx = pic(1,1);

for i = 1:hres,
    for j = 1:vres,
        if ( pic(j,i) < mn ) mn = pic(j,i); end;
        if ( pic(j,i) > mx ) mx = pic(j,i); end;
    end;
end;

%calculate m
m = 255/(mx-mn);

%stretch the input image into an output image
stretched_pic = pic;
for i = 1:hres,
    for j = 1:vres,
        stretched_pic(j,i) = round((pic(j,i)-mn)*m);
    end;
end;

%display the results
figure(2);image([pic stretched_pic]);
colormap(gray(256));axis image;axis off;

%put up a title on Figure(2);
label = ['Original (left) range ',num2str(mn),':',num2str(mx),'
        Stretched (right)'];
title(label);
9.2 Case Study 2: Histogram Modification

Another more powerful contrast enhancement method for images is Histogram Equalisation. See pages 453-459 in *Two-Dimensional Signal and Image processing by Jae Lim* for a further explanation of the method. The idea, (as with stretching above), is to redistribute the grey scale intensities of pixels in the image so that the overall grey scale distribution is evenly spread throughout the range 0 to 255 in this case. The operation uses Histograms of pixel intensities. This Histogram measures the number of pixels which have a particular intensity value. Thus for an 8 bit image, the histogram is typically an array of 256 elements (corresponding to intensity values 0 to 255) and the entry in each element is the number of pixels in the image which have that value. Thus \( H(100) \) is the number of pixels which have the value 100 in the grey scale image.

The transformation for Histogram Equalisation is derived so as to make the Histogram of the output image match a uniform distribution as close as possible. That means that the output histogram should have the same frequency for all bins, thus making the contrast in the image as reasonable as possible. This is done, not by matching Histograms, but by matching the Cumulative distribution of the pixel intensities. The idea is that the output image has a cumulative distribution which matches closely a straight line from frequency 0 at intensity 0 to a frequency of \( N \times M \) at intensity 255, where \( N, M \) are the horizontal and vertical sizes of the input image.

The Cumulative Distribution function (Cudf) is just the cumulative sum of the Histogram bins. Thus letting \( C_j \) be the frequency of the Cudf at value \( j \), then \( C_j = \sum_{i=0}^{j} H_i \). To design the intensity transformation, we need to derive the target Cudf. This will be \( C'_j = NMj/255 \). To make the mapping itself the idea is to design a look-up-table (LUT) such that \( y = \text{LUT}(x) \) where \( x, y \) are input and output pixels respectively, and the Cudf of \( x \) is \( C \) and the Cudf of \( y \) is \( C' \).

The routine for making the LUT is as follows

1. for each pixel intensity \( i \) (0 . . . 255)
2. Examine each value \( C_j \) for \( j = 0 \ldots 255 \) and find the best match to \( C'_i \)
3. Assign \( \text{LUT}(j) = i \)

Finding the “best match” is done by measuring the difference \( |C_j - C'_i| \) and picking the combination which gives the minimum difference.

---

1See [http://www.joeljeffery.co.uk/joel/XMRI/Chapter%203.htm](http://www.joeljeffery.co.uk/joel/XMRI/Chapter%203.htm) for fast information.
CHAPTER 9. EFFICIENT MATLAB 9.2. HISTOGRAM MODIFICATION

☐ Write your own, efficient Histogram Equalisation .m file. Make sure it does the following

1. Read in an image file

2. Display it in Figure 1 with a title that says Original image.

3. Measure $H_i$ the histogram of the input image

4. Calculate $C_i$ the Cumulative distribution function of the input image using $H_i$ (Hint: help cumsum)

5. Plot your measured $C_i$ in Figure 2 with a title Original Cudf.

6. Make the target Cudf $C^t$ as above

7. Find the LUT for mapping $C_i$ onto $C^t$

8. Transform the input image pixel intensities using your LUT

9. Measure $H_i^o$ the histogram of the output image

10. Calculate $C_i^o$ the Cumulative distribution function of the output image using $H_i^o$ (Hint: help cumsum)

11. Plot your measured $C_i^o$ in Figure 3 with a title Transformed Cudf.

12. Display your transformed image as Figure 4 with a title Hist Eq Image.

It turns out that the Image Processing toolbox has a function which does histogram equalisation.

☐ How fast is it compared to your code?

To help you get started see the .m script w:\isd\matlab\tutorial\ack_slow_histeq.m.