Contents

1 Starting Matlab 2

2 Entering Matrices, Basic Operations 2

3 Editing Command Lines 4

4 Getting Help 4

5 Interrupting, Quitting Matlab 5

6 Special Matrices 5

7 Submatrices 6

8 Creating Script Files 6

9 Plotting Curves And Surfaces 7

10 Diary 9

11 Relational Operators, Conditional Statements 9

12 Functions 10

13 Iterations - Loops 13

14 Reading and Writing Data To Files 15

15 Summary Of Other Useful Commands 16
1 Starting Matlab

In the E-Calc Lab Matlab may be started by

- The computer when booted will have you logged in.
- After logging in, click the Start button and then choose Programs and then Matlab 6.1 and Matlab will start.

Starting Matlab will open a window which is the primary interface for Matlab. Within this window there is a sub-window called the command window, with >> as a prompt. All Matlab commands are entered at this prompt. Closing this window, closes the session.

To execute a script file, first change the directory to the place where the file resides by giving the command cd z:\dirname\dirname2 where z: is the drive name and \dirname\dirname2 is the path. Finally the script is executed by entering the filename at prompt.

2 Entering Matrices, Basic Operations

Matrices are entered by listing the elements row by row, with the rows separated by semicolons and the entries separated by blank spaces (or commas); e.g. type the following and hit return after each line

\[
A = \begin{bmatrix} 3 & 3 & 3; 2 & 0 & 0; 1 & 0 & 1; 5 & 5 & 5 \end{bmatrix} \\
B = \begin{bmatrix} 1 & 0 & 0; 0 & 0 & 2; 4 & 4 & 4 \end{bmatrix} \\
C = \begin{bmatrix} 2 & 0 & 2 & 0 \end{bmatrix} \\
D = \begin{bmatrix} 2; 0; 2 \end{bmatrix}
\]

defines a 4x3 matrix \(A\), a 3x3 matrix \(B\), a 1x4 matrix \(C\), and a 3x1 matrix \(D\). The three basic operations on matrices are addition, scalar multiplication, and multiplication.

\[
E = A*B \\
F = 3*E - 5*A \\
F + 3 \\
B^3 \\
\text{sin}(A)
\]

Note the use of the sine function on a matrix - operations of that form can be quite convenient. However, occasionally there can be some confusion about such use. For example, \(A^2\) represents the matrix multiplication of \(A\) with itself, not the square of each entry of \(A\) - to square each entry of \(A\) one must use \(A.^2\) (note the dot after the \(A\)). In general, a dot will result in the operation being performed entry by entry instead of on the whole matrix. Try the following examples

\[
A = \begin{bmatrix} 1 & 1; 1 & 1 \end{bmatrix} \\
B = \begin{bmatrix} 1 & -1; 1 & -1 \end{bmatrix} \\
A*B \\
A.*B \\
A^2 \\
A.^2
\]

A times B
term by term product
Matlab is a numerical package unlike Maple which stores everything symbolically. So Matlab cannot operate on matrices which have some symbols such as $3x$ as one of its entries where as Maple can. Of course this handicap of Matlab is also a strength because then its algorithms can be geared towards numerical quantities resulting in faster algorithms.

Matlab stores numbers such as $3/7$ in their approximate decimal form $0.4285714\cdots$. However, many of us prefer to deal with fractions rather than a long decimal expansion. Matlab has the ability to print its output in rational format (which can be convenient) instead of the decimal format. Also, I recommend using the compact format simultaneously - it removes extra blank lines making the output more compact. I recommend the \texttt{format rational compact} when you want fractional output, \texttt{format short compact} when you want 5 digit accuracy in output and sometimes \texttt{format long compact} when you want 15 digit accuracy in output. Try

\begin{verbatim}
3/7
format rat compact
3/7
4/9
format long compact
4/9
format
\end{verbatim}

Other operations on matrices which we will introduce are the transpose, inverse, determinant, sum, and norm of vectors.

\begin{verbatim}
format rat compact  I prefer rational output where appropriate
A = [ 1 2 3 ; 4 5 6 ; 1 3 4 ]
rref(A)              reduced row echelon form of A
det(A)               determinant of A
transpose(A)         transpose of A
B = inv(A)           B is the inverse of A
B*A,    A*B           checking B is the inverse of A
\end{verbatim}

Here are some other useful remarks concerning Matlab.

- Normally Matlab will display the result of executing a command but if the command is ended by a \texttt{;} then the result is not displayed. This can be convenient when several commands are to be executed and one is interested only in the final result.

- Several commands may be given on the same line as long as the commands are separated by \texttt{;} or \texttt{,}.

- Any text after the \% symbol is ignored by Matlab - so comments are inserted in Matlab code by starting that line with the \% symbol.

- Commands too long to fit on a line are continued by typing \ldots{} at the end of the lines.

We give a few examples to explain these comments.

\begin{verbatim}
a = 2; b = 3; c = a+b, d = a-b only c and d will be displayed
A = [ 1 1 1 ; 2 2 2 ; 3 3 3 ] + ... hit Return and continue typing
    [ 0 0 0 ; 0 1 0 ; 1 1 1] hit Return
3*5 \% ABC54F Matlab ignores any text on a line after a \% symbol
disp(’ This is the way to print phrases in Matlab ’)
\end{verbatim}
3 Editing Command Lines

To correct errors in a command line use the Delete or Backspace key to delete characters. To modify a command line use the ← or → keys on the right of the keyboard to move to the correct position, delete the appropriate characters and type the correct ones. If the arrow keys don’t work then use Control B and Control F to move backwards and forward. As an example we show how to modify a previous command line. Suppose, we wish to modify the command used above

\[ a = 2; \ b = 3; \ c = a+b, \ d = a-b \]

to

\[ a = 5; \ b = 7; \ c = a+b, \ d = a-b \]

This may be done by

- Press the Control and P keys simultaneously (or press ↑ key) once, to get the previous command issued. You should get disp(....). Repeat this - you get [ 0, 0, ...]. Repeat this until you get the line we want to modify.
- Using ← key or Control B move cursor left till it is on 2. Delete 2 and type 5.
- Similarly delete 3 and insert 7. Now hit Return to execute the statement.

Above, we used Control P repeatedly to get previously used commands and may use Control N to get the next command in the stack of command lines. At many sites the ↑ and ↓ keys may be used instead. Summarizing
Moving Left - Control B (i.e. back) or ← key
Moving Right - Control F (i.e. forward) or → key
Moving Up - Control P (i.e. previous command) or ↑ key
Moving Down - Control N (i.e. next command) or ↓ key
To get more info about these editing commands try help cedit.

4 Getting Help

Matlab has online help available in several ways. A very useful command is helpdesk. Suppose you wish to find a command to sum the entries of a matrix. Within Matlab type

```
helpdesk
```

and a browser will start up. In the area Go to MATLAB function type sum and click Go. A page will give information on the sum command of Matlab. Read it and then see why the following works. So go to the Matlab window and type

```
format change to standard format
A = [ 1 2 3; 7 5 2]
sum(A) sums the columns of A
sum(sum(A)) sum of all the entries of A
```
The `helpdesk` command will not work if you have the student version of Matlab.

In addition to the `helpdesk` command there are two other commands - `lookfor` and `help`. The `lookfor` command is useful if you want to find a command to do something but you do not know the name of the command. If you already know the command you want to use and want to find its syntax then use `help`. For example, suppose you wanted to find the command for plotting a surface. Try

```
help plot  this plots curves in two dimensions
lookfor plot more than a pageful of appropriate commands
more on  enable scrolling
lookfor plot read each line and find correct command
```

To end the output from a `lookfor` type `q`. Actually, the `lookfor` command can be tedious, and I recommend using `helpwin` or `helpdesk`.

### 5 Interrupting, Quitting Matlab

To interrupt a Matlab calculation press the Control and C keys simultaneously. If you are sure you want to quit Matlab then type `quit` and hit Return.

### 6 Special Matrices

Certain matrices show up often and it would be tedious to enter them by hand. Matlab has some convenient commands to generate some of the commonly occurring matrices.

- `zeros(m,n)` produces an mxn matrix of zeros and `zeros(n)` produces an nxn matrix of zeros. Similarly `ones(m,n)` and `ones(n)` produce mxn and nxn sized matrices of ones.
- `eye(n)` produces the identity matrix of order nxn and `eye(m,n)` produces an mxn matrix with ones along the diagonal.
- If x is a (row or a column) vector then `diag(x)` is a diagonal matrix with x as the diagonal; if A is a square matrix then `diag(A)` is a column vector consisting of the diagonal of A.
- `rand(n)` and `rand(m,n)` are nxn and mxn random matrices with entries uniformly distributed on the interval [0, 1].

Examine the result of executing the following - is it what you expected.

```
zeros(2,3), zeros(2)
eye(3), eye(4,3)
x = [1 2 3 4], y = [3; 3; 3; 3;]
diag(x), diag(y)
A = 5*ones(3) + 2*eye(3)
diag(A)
rand(3), rand(3), rand(2,3)
```
Submatrices

Matlab has excellent commands to create new matrices from already occurring matrices. Suppose \( A, \ B, \ C \) are appropriate sized matrices, then, recalling that a comma separates the entries in a row and that a semicolon separates the columns, \([ \ A, \ B, \ C \] \) will stack the matrices \( A, \ B, \ C \) side by side. The command \([ \ A; \ B; \ C \] \) will stack \( A \) on top of \( B \) on top of \( C \). Try

\[
\begin{align*}
A &= [1 \ 2; \ 3 \ 4] \\
B &= [5 \ 6; \ 7 \ 8] \\
C &= [9 \ 10; \ 11 \ 12] \\
[A, \ B, \ C] & \quad \text{stacks } A, B, C \text{ side by side} \\
[A; \ B; \ C] & \quad A \text{ on top of } B \text{ on top of } C
\end{align*}
\]

Matlab also has excellent commands to modify parts of matrices. \((i, j)\) refers to the \((i,j)\)th entry of a matrix; \((2:4, 3:5)\) refers to the submatrix formed by the entries in rows 2 to 4 and columns 3 to 5 of some matrix; \((2:4,:))\) refers to submatrix formed by the entries in the rows 2 to 4 and all columns. Examine

\[
\begin{align*}
B &= [ \ \text{ones}(3) \ ; \ \text{eye}(3) \ ] \quad \text{a } 6 \times 3 \text{ matrix} \\
B(2,2) &= 5 \quad \text{change (2,2) entry of } B \text{ to 5} \\
B(:,3) &= 7 \times \text{ones}(6,1) \quad \text{change 3rd column of } B \text{ to sevens.} \\
B(4,:), &= 3 \times \text{ones}(1,3) \quad \text{change 4th row of } B \text{ to threes} \\
C &= B(2:4,1:2) \quad \text{appropriate submatrix of } B \\
A &= [ \ \text{ones}(3) \ ; \ \text{eye}(3) \ ] \quad \text{a } 3 \times 6 \text{ matrix} \\
A(2:3,3:end) &= \text{2nd and 3rd rows of columns 3 to 6} \\
A(:,3:5) &= \text{3rd to 5th columns of } A
\end{align*}
\]

One other way of creating matrices is based on the idea of a loop - you will see it again later. For example to create a row matrix containing the numbers 2, 5, 8, 11, 14, ..., up to 21 i.e. numbers starting at 2 incremented by 3 up to 21 use

\[
\begin{align*}
x &= [2 \ : \ 3 \ : \ 21] \quad \text{a row starting at 2, increase by 3, up to 21} \\
x &= [30 \ : \ -4 \ : \ 15] \quad \text{a row starting at 30, decrease by 4, up to 15}
\end{align*}
\]

Creating Script Files

In Matlab, one works interactively when experimenting with a few commands. However, more frequently, one wishes to experiment with a long sequence of commands which may need modification. It would be very tedious to retype essentially the same commands for every experiment. Instead one stores these commands in a file and executes the file as shown below.

We now create a file named junk.m with the content shown below. After starting Matlab, type edit and an editing window will popup. In that window start typing what is given below.

\[
\begin{align*}
% \ 1ST \ PROJECT \ - \ M305 \\
% \ \text{JANE \ DOE} \\
% \\
% \ \text{testing \ whether \ } AB = BA \ \text{for \ two \ square \ matrices} \\
% \ \text{create \ two \ random \ } 3 \times 3 \ \text{matrices \ with \ integer \ entries \ between \ 0 \ \text{and \ 10} \\
% \\
% \ \text{use \ help \ floor \ or \ try \ floor(3.12) \ to \ find \ out \ about \ floor \ command}
\end{align*}
\]
A = floor( 10*rand(3) )
B = floor( 10*rand(3) )

% test whether AB = BA
disp('A*B is'), A*B
disp('B*A is'), B*A
disp('A*B - B*A is'), A*B - B*A

When you are done, check that you have typed everything correctly, and then from the File menu choose Save. You will be prompted to choose a Folder/Directory where you want the file saved, and you will also be asked to name the file. You should save this file in the scratch drive z: in your folder and the name you choose is junk.m.

To execute the commands in the file junk.m, within Matlab type junk and the commands in the above file will be executed (change to directory of file location). Type junk again and note the new A and B - because A and B are generated randomly every time. So we can execute these commands as many times as we want without retyping the commands. Note, lines beginning with % are ignored by Matlab and one uses them to write comments.

For future use, to edit a file which was created in the past, type edit in Matlab, and then in the menu of the editing window, choose File Open and then choose the appropriate Folder/Directory and file.

9 Plotting Curves And Surfaces

To draw the graph of a function of one variable use the plot command. It needs two vectors - the vector of x coordinates and the vector of the corresponding y coordinates (they don’t have to be called x and y). For example, to draw the graph of \( x^2 + 2 \sin(x) \) over the interval \([-1,2]\), we generate the x and y vectors and then use plot.

\[
x = \text{linspace}(-1, 2, 30)
y = x.^2 + 2*\sin(x)
\]

plot(x,y)

Note the use of the dot after the x in the calculation of y (remember \(x^2\) stands for matrix square rather than squaring each term of a matrix). To obtain the coordinates of any point on this graph, in the Matlab window type

ginput(1)

and a crosshair will popup when the mouse is placed over the graph. Click the point whose location you want and the coordinates will be given in the Matlab window.

To draw two graphs in the same window e.g. \( x^2 \) over \([-1,1]\) and \( \sin(x) \) over \([-2,2]\) use

\[
x1 = \text{linspace}(-1,1,30);
y1 = x1.^2;
x2 = \text{linspace}(-2,2,30);
y2 = \sin(x2);
plot(x1, y1, x2, y2)
\]

Note that the new graph replaces the old graph. Suppose we want to retain this graph for future use then from the File menu of the latest graph window choose New Figure and a new graph window
shows up. Future graphs will be drawn on this window. We next draw the graph of the cycloid which is given in parametric form as

\[ x = \theta - \sin \theta, \quad y = 1 - \cos \theta, \quad 0 \leq \theta \leq 4\pi \]

t = linspace(0, 4*pi, 120);
x = t - sin(t);
y = 1 - cos(t);
plot( x, y, 'r-' ) draw a ('r') red graph with ('-') solid lines

and note that the graph is drawn in the new window instead of overwriting the previous window.

Now suppose we want to examine the graphs in the first graphical window more carefully. Activate the first graphical window by clicking it on the top bar. We can give it a Title, choose labels for the x and y axes and perform various other such actions. Try

```
title( 'Test Figure')
xlabel( 'Time after lift off' ) label for horizontal axis
ylabel( 'Temperature outside' ) label for vertical axis
```

Using the `axis` command the horizontal and vertical axis sizes may be chosen to focus on the part of the graphs which are of interest.

The above actions may also be performed interactively by editing the graphical window. In the graphical window, click the \ to enable editing of the graphical window (clicking it again will disable editing). Now go to the Tools menu and choose the operation you wish to perform - use the Help menu if necessary. Disable the editing after you are done with the editing.

Finally, suppose we want to add a graph to a graphical window. Then first activate the window, but if we now plot the new graph it will replace the current graph in that window. So we first tell Matlab to `hold on` to that window and then plot the graph. So to add the graph of \( x + \sin x \) over \([-1,1]\) to Figure No. 1, we first activate that window (if it is not the active window), then

```
hold on
x = linspace(-1, 1, 30);
y = x + sin(x);
plot(x, y, 'go') puts green circles at data points
hold off
```

and the new graph is added to the previous graphs. Now make sure you open a new graphical window from the File menu or the next plot you draw will wipe out the graphs in the current window.

Three dimensional plotting is analogous to the two dimensional plotting. To plot the parametric curve (Helix)

\[ x = \cos t, \quad y = \sin t, \quad z = t, \quad 0 < t < 4\pi \]

use

```
t = linspace(0, 3*pi, 60);
x = cos(t);
y = sin(t);
z = t;
plot3(x, y, z, 'r-') solid red curve
```
One can view this curve interactively by clicking the rotation associated button on the graphical window. Click that button and view the curve from different viewpoints.

To draw plots of functions of two variables (i.e. surfaces) the command `surf` is used. Suppose we wish to draw the plot of \( x^2 + y^2 - 2x \) over the region \([0, 2] \times [-1, 1]\). `surf` needs three inputs - a vector \( x \) consisting of the subdivision points along the \( x \) axis, a vector \( y \) consisting of the subdivision points along the \( y \) axis, and a matrix \( Z \) whose \((i,j)\)th entry is the value of the function at \((x_j, y_i)\) where \(x_j\) and \(y_i\) are the \(j\)th and \(i\)th entries of \(x\) and \(y\) respectively.

\( x \) and \( y \) are constructed as before using `linspace`, and \( Z \) may be constructed using a nested loop (one loop inside another). However, loops can slow down the computation so it is better to achieve our goal using vectors. The first step in the construction of \( Z \) is to construct two matrices \(X\) and \(Y\) so that their \((i,j)\)th entries are \(x_j\) and \(y_i\) respectively. Then the construction of \(Z\) follows easily by applying the function to the corresponding entries of \(X\) and \(Y\). Matlab has a function `meshgrid` which efficiently constructs the matrices \(X\) and \(Y\). Try

\[
x = \text{linspace}(0, 2, 30);
y = \text{linspace}(-1, 1, 30);
[X,Y] = \text{meshgrid}(x,y);
Z = X.^2 + Y.^2 - 2*X;
surf(x, y, Z)
\]

Just as for `plot` and `plot3`, the axes may be labeled, a title may be given, the figure rotated, color scheme chosen etc. Use `help surf` for instructions.

### 10 Diary

In some situations, we may wish to have a record of the Matlab commands used and the output generated. This may be for example needed when solving a home work or a lab problem for the course. This is accomplished with the `diary` command in Matlab. Suppose the record generated is to be stored in the file named `mywork`. Try the following

```matlab
diary mywork                  starts a diary stored in the file mywork
a = [1 2 3 5 8];
b = [3 4 5 6 10];
a+b
2*a - 3*b
a*b'
diary off                   end of record - no more stuff stored
```

The Matlab commands given above and the resulting output has been stored in the text file `mywork`. On X-terminals, this file may be printed with the command (within Matlab)

```
!lpr mywork          don't forget the exclamation mark
```

and it will be printed at the nearest printer or in the basement of Smith Hall. On PCs use the standard print commands.

### 11 Relational Operators, Conditional Statements

When comparing two numbers \(x\) and \(y\) the following comparisons may be made
Here is an example of a conditional statement in Matlab.

```
if (x > 7)
y = x^2;
z = x^3;
elseif ( (x > 1) & (x < 3) )
y = x^3;
z = 0;
else
    y = 0
    z = x^3
end
```

The general syntax for a conditional statement is

```
if (condition)
    statements
elseif (condition) optional
    statements optional
else optional
    statements optional
end required
```

More elseif sections can be added before the else if needed. Use the help if and help switch to find out more.

The condition can take a simple form such as x<1 or x==y or can be a compound expression such as (x<1) & (y>2) or such as made by the use of a "logical or" (x<1) \(\lor\) (y>2). Be careful about the use of 1, the condition A\(\lor\)B means "A or B or both A and B".

We will show the use of conditional statements in the next section.

### 12 Functions

A function is a set of commands which takes an input of a certain type (e.g. a matrix) and gives as output a quantity of a certain type (e.g. a matrix) associated with the input (e.g. the inverse of the input matrix). Matlab already has builtin functions such as `transpose(A)`, `inv(A)`, however we may need functions which are not builtin.

We start by writing a function `f` whose input is a real number `x` and the output is `x^2 + sin(x) - 5`. However, we want to apply this function also to matrices so that if `x` is a matrix then `f(x)` should be the matrix obtained by applying `f` to each entry of `x`. So create a file named `f.m` with the following content

```
function value = f(x)
```

% input: matrix or real number x
% output: matrix obtained by applying a \rightarrow a^2 + \sin(a) - 5 to
% each entry of x
value = x.^2 + x - 2 ;
return; % marks the end of the function

After saving the file as f.m we now try the function f. PC users make sure that the active folder
is the location of f.m. In Matlab try the following commands

x = 2;
f(x)
x = [1 2 3; 0 2 0]
f(x)

Are the answers what you expected? By the way, note the use of x.^2 in the code of the function
f - that will work whether x is a matrix or a real number. It is very convenient to write functions
which have this flexibility (where possible).

Here are some general remarks about functions.

- A function is created in a separate file with the name of the file being the name of the function
  except the file name has the suffix .m .
- The first line will have the form
  
  function {output variables} = {name of function}(input variables)

- The function is ended with the return;
- Make sure you use ; at the end of each command so that the intermediate output is not
  displayed (unless you want to see the intermediate output).
- It is good practice to describe the input variables and also describe what the output will be.

We now write a more complicated function named matsum which returns the sum of all the
positive/negative entries of a matrix. We will use the following observation to avoid using loops
(described later). If A is a matrix, then for any matrix A which is already defined

A = type in any matrix
S = A>3;

creates a matrix S which is 1 at the entries where A > 3 and 0 at all other entries. Hence the
commands

A = type in any matrix
S = A>3;
B = S.*A

creates a matrix which has the same entries as A for entries greater than 3 and has a zero for all
other entries.

We will also use the built-in function sum. For a vector x, sum(x) gives the sum of the entries of
x and for a matrix x, sum(x) will return a row vector which contains the sum of the column entries
of x . So for a matrix x, sum( sum(x) ) would give the sum of the entries of x. However, we
want the sum of the positive and negative entries of x .
function [possum, negsum] = matsum(A)
%
% input: matrix A
% output: a 1x2 matrix [possum, negsum]
%       possum - sum of the positive entries of A
%       negsum - sum of negative entries of A
%
% construct matrix of positive entries in A - other entries will be zero
S = A>0;
posA = A.*S;
%
% construct matrix of nonpositive entries of A - other entries will be zero
negA = A - posA;
%
% compute the sum of the positive and negative entries of A
possum = sum( sum(posA) );
egsum = sum( sum(negA) );
%
%end of function matsum
return;

After saving the file as matsum.m, in the command window try

C = [1 2 3; -4 -5 -6]
[p,n] = matsum(C) \ p assigned sum of positive entries of C, n the negative
help matsum \ compare with the lines in the file matsum.m

A Matlab function recognizes only those variables created within it (including the input variables). Variables assigned values outside a function definition will not be recognized inside a function. If you want to use an outside variable then it has to be explicitly declared global within the function. By convention, in Matlab, global variables are assigned names in upper case.

Functions can be called recursively i.e. a function can call itself. However, recursive calls are inefficient and should be avoided if possible. Here is an example of a recursive function - it computes the factorial of an integer using the relation $factorial(n) = n * factorial(n - 1)$.

function p = myfact(n)
% input : positive integer n
% output : p, the factorial of n

if (n<=1)
    p = 1;
else
    p = n*fact(n-1);
end % ending if

return; %ending myfact

Store this in a file named myfact.m and then try it within Matlab by typing

myfact(3) - should be 6
myfact(5) - should be 120
As in Fortran or C, Matlab allows loops and conditional statements. In Matlab, as far as possible, avoid loops of any kind (for or while) and also conditional statements - these slow down Matlab. If the same goal can be achieved by operating on vectors then that is much faster. However, sometimes one is forced to write an iteration.

If a certain code is to be executed repeatedly as long as a certain condition is valid (e.g. when solving an ODE over an interval i.e. computing the solution until the end of the interval is reached) then iteration is used. Consider the following problem - Compute \( \sum \frac{1}{n} \) where the sum is taken over all integers between 5 and 200 for which \( \sin(n) \) is between 0.5 and 0.6. This problem may be solved using a for loop. Type the following in a separate file (script file) then execute that file.

```matlab
% Problem I
% sums 1/n for all integers n between 5 and 200 for which sin(n) is
% between 0.5 and 0.6. Will compute a separate sum for other n
% Note - Matlab assumes n is given in radians (not degrees)

% initialisation
sum56 = 0;
sumother = 0;

% computation
for n = 5:1:200
    if ( (0.5 < sin(n)) & (sin(n)< 0.6) )
        sum56 = sum56 + 1/n;
    else
        sumother = sumother + 1/n;
    end %ending the if statement
end % ending the for statement

% print the sums
sum56, sumother
```

Note the statement `n=5:1:100`. It says to take values of \( n \) from 5 to 100 incrementing \( n \) by 1 i.e. \( n = 5, 6, 7, \ldots \) etc. Also, note that all the intermediate statements were ended by semicolons because we were not interested in the results of the intermediate calculations.

In the previous example a certain piece of code was iterated a known number of times. However, there are situations where it is not known beforehand the number of times a certain code is to be executed. In those cases a while loop is used. Consider the problem - compute the sum of \( \cos(t) \) for all values of \( t \) starting at \( t = 1 \), \( t \) incremented by 0.1, and until we meet the first \( t \) for which \( \cos(t) \) is negative.

```matlab
%Problem II
% sums cos(t) for all t starting at t=1, t incremented using a step size
% of 0.1, until cos(t)<0.

t = 1;
cosum = 0;

while (cos(t) > 0)
```

Note the statement `n=5:1:100`. It says to take values of \( n \) from 5 to 100 incrementing \( n \) by 1 i.e. \( n = 5, 6, 7, \ldots \) etc. Also, note that all the intermediate statements were ended by semicolons because we were not interested in the results of the intermediate calculations.
\[
\text{cossum} = \text{cossum} + \cos(t); \\
t = t + 0.1; \quad \%\text{ increment t} \\
\text{end} \quad \%\text{ ending the while statement}
\]

\% print the sum
\text{cossum}

Note that here we do not know beforehand the number of times the body of the loop will be executed. The general syntax for a \textbf{for} loop is

\text{for variable=expression} \\
\text{\hspace{0.5cm} statements} \\
\text{end}

and the syntax for a \textbf{while} statement is

\text{while expression} \\
\text{\hspace{0.5cm} statements} \\
\text{end}

Computationally, the programs will run faster if wherever possible, the conditional statements and loops are replaced by vector statements, because vector statements may be executed in parallel whereas conditional or loop statements have to be executed one by one. Let us see how to solve Problem I without the use of any conditional statements or loops. We will use the conditional statement trick used earlier. Go carefully through each line of the program below.

\% Efficient solution of Problem I \\
\% sums 1/n for all integers n between 5 and 200 for which sin(n) is \\
\% between 0.5 and 0.6. Will compute a separate sum for other n

\text{n} = \left[5 : 1 : 200\right]; \quad \%\text{ generates vector 5 6 7 ...100}
\text{sinn} = \sin(n); \quad \%\text{ takes sine of 5, 6, 7, ... .}
\% compute vector s which is 1 at entries where sinn lies between 0.5 and 
\% 0.6 and 0 elsewhere
\text{s} = \left(0.5 < \text{sinn}\right) \& \left(\text{sinn} < 0.6\right);

\% Construct vector v56 whose entries are same as sinn entry if the sinn 
\% entry lies between 0.5 and 0.6. Entry is 0 otherwise.
\text{v56} = \text{sinn.*s};

\% sum of entries of sin(n) between 0.5 and 0.6
\text{sum56} = \text{sum(v56)}

\% sum of all other entries
\text{sumother} = \text{sum(sinn)} - \text{sum56}

\%End of solution of Problem I

We now give another example of the idea of using conditional statements for matrices. Suppose we wish to write a function

\[
f(x) = \begin{cases} 
2 - x & \text{for } x < 2 \\
x^2 - 4 & \text{for } x \geq 2 
\end{cases}
\]

One solution is (create the file \texttt{f.m} with)
function y=f(x)
% input : x real
% output : y real as defined above

if (x<2)
    y = 2 - x;
else
    y = x^2 - 4;
end

return

This function is correct but only works for real $x$. If we wanted to plot the graph of $f$ over an interval say [0,4] we could not use

```matlab
x = linspace(0,4,100);
y = f(x); % Matlab will complain
plot(x,y) % will not work because y not defined
```

because the way $f$ is defined above will not work for vector $x$ - see where in the code for $f$ does it fail.

The solution is to write $f$ so that it works even if $x$ is a vector in the sense that $f(x)$ results in a vector obtained by applying $f$ to each entry of $x$. This may be accomplished as follows (save in file newf.m)

```matlab
function y = newf(x)
% input : x matrix
% output : y matrix same size as x with f applied to each entry of x

% create vector which gives 2-x for entries of x < 2 and zero else where
y1 = (2-x).*(x < 2);

% create vector which gives $x^2 - 4$ for entries of $x \geq 2$ and zero else where
y2 = (x.^2-4).*(x >= 2);

y = y1 + y2;
return
```

Now try to plot the graph of $f$ or rather newf.

```matlab
x = linspace(0,4,100);
y = newf(x);
plot(x,y)
```

14 Reading and Writing Data To Files

Suppose we have a matrix $A$ whose contents we wish to save in a file named myfile.doc. In Matlab, type

```matlab
A = [ 1.0 2.05 3.6; -2.0 -3.0 2.1 ]
dlmwrite( 'myfile.doc', A, ', ' )
```
This results in each row of $A$ written on a separate line of the file `myfile.doc` with the entries separated by a space (that is the purpose of the last argument of the `dlmwrite` function). To see that this has indeed happened after executing the command, open a separate window and examine the contents of the file `myfile.doc`.

Next, suppose we have a file named `myfile.doc` which contains numbers separated by blanks and we want to load these numbers into a matrix named $B$. Use

$$B = \text{dlmread( 'myfile.doc', ' ')}$$

Note the syntax, the `dlmread` command needs two inputs - the name of the file and the character separating the entries in the file. This command results in the first line of `myfile.doc` forming the first row of $B$, the second line forming the second row, etc.

For other forms of input and output the commands `fprintf` and `fscanf` are used and a separate handout will be given for this.

### 15 Summary Of Other Useful Commands

Below we give the commands which are useful for a variety of purposes. A detailed description together with examples are available from the `helpdesk` command within Matlab.

- The solution of the linear equation $Ax = b$ for $x$, is given by $A\backslash b$. From `helpdesk` choose `Subject` and then choose `Operators and Special Characters` to get more information.

- To solve a single nonlinear algebraic equation use the `fzero` function and to solve a system of nonlinear algebraic equations use the `fsolve` command in the Optimization Toolbox of Matlab. On the `helpdesk` page, the Optimization Toolbox reference is in the right column.

- To solve a system of ODEs several commands are available. Start with `ode45` and see the other commands listed when you read the documentation on `ode45`.

- To integrate numerically a function of one variable use the commands `quad` or `quad8`.

- To minimize functions of one or more variables use the commands in the Optimization Toolbox of Matlab.

- Various statistical functions are available from the Statistics Toolbox.

**Prepared by Dr. Rakesh**