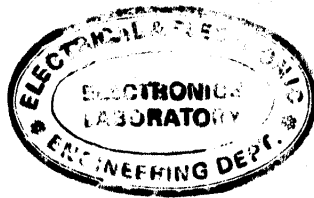


DEPARTMENT OF ELECTRICAL & ELECTRONIC ENGINEERING

AN INTRODUCTION TO MATLAB



May 2001

Contents

Chapter 1 Fundamentals

- 1.1 Introduction.
- 1.2 Expressions
- 1.3 Statements and variables
- 1.4 Elementary functions
- 1.5 Array operations
- 1.6 Graphics '
- 1.7 Command-line editor
- 1.8 Word processing

Chapter 2 MATLAB Scripts and menus

- 2.1 Introduction
- 2.2 Command Window menus
- 2.3 Figure Window menus
- 2.4 Creating script files
- 2.5 Function files
- 2.6 Further Examples

Chapter 3 Further MATLAB statements

- 3.1 Introduction
- 3.2 For loops
- 3.3 Conditional statements
- 3.4 Complex numbers
- 3.5 Polynomials

Chapter 4 Engineering: Problem Solving:

- 4.1 Engineering Challenges
- 4.2 Engineering Problem Solving
- 4.3 Storing data
- 4.4 Formatted output
- 4.5 While loops
- 4.6 Logical expressions

Chapter 5 Statistical Calculations

- 5.1 Data analysis functions
- 5.2 Standard deviation

An introduction to MATLAB

Chapter 1 Fundamentals

1.1 Introduction

The advent of the personal computer means that engineers no longer have to perform tedious and error-prone calculations, leaving more time for the analysis and interpretation of the results. For the Computer to be able to solve any problem it must be given a series of instructions, a computer *program* to follow. These instructions are given in a language, which the computer can process. Nowadays, there are any languages available. Indeed, there is a hierarchy of languages available so that the user can express the problem in a "user friendly" language as data for a program, this program translates the source data into data for yet another program to translate and so on until a sequence of instructions which the computer can understand is obtained.

The purpose of this course is twofold.

1. It provides a means of obtaining numerical and graphical results of calculations commonly arising in engineering.
2. It provides an introduction to some of the features of high-level computer languages, whilst still enabling results to be obtained quickly and easily.

MATLAB is an interactive program widely used for engineering calculations. It provides an effective environment for the beginner as well as the very sophisticated requirements of the professional engineer and scientist. The MATLAB user has direct access to the results of recent research in numerical methods and algorithms. The widespread use of MATLAB means that an interdisciplinary programming environment is available for teaching and research now. It is widely used in industry and academia.

MATLAB uses a variety of objects that allow the user to interact with the program. These objects include:

1. expressions
2. statements and variables
3. arrays
4. graphics and
5. scripts

MATLAB interprets and acts on input in the form of these objects.

To begin a session on a PC using Windows double-click on the MA TLAB program icon. This brings **up** the MA TLAB Command Window and the MA TLAB command prompt:

»

At this prompt a MA TLAB statement can be typed, When the <enter> key is pressed MA TLAB will respond.

You can exit MATLAB at anytime by selecting the MATLAB Command Window File Menu item Exit MATLAB.

MATLAB user guides and reference manuals are available and an extensive Help facility) can be accessed via the MATLAB Command Window Help Menu,

1.2 Expressions

The usual arithmetical operators:

- + addition.
- subtraction,
- * multiplication,
- /division, and
- ^ power

can be used in expressions. The rules of precedence are ^ first, then /, then *, then -, and then +. Expressions in brackets are evaluated first. Where two operations rank equally in the order of preference, then the calculation is performed from left to right.

Blank spaces around =, +, and - signs are optional, and are used to improve readability.

Arithmetic expressions allow MATLAB to be used in "**calculator mode**" as shown below:

```
» 100/5

ans =
    20

» 100/ 50/2

ans =
    10

» 100 + 5/2

ans =
   102.5

» (100 + 5)/2

ans =
    52.5

» 100 + 5"2/2

ans =
   112.5

» 100 + 5 ^ (2/2)

ans =
   105
```

In these examples, the result is assigned to the generic variable `ans` and is printed on the screen.

Variables can be used in expressions if they have previously been assigned a numerical value. as in the following example

```
»a = 10;    b = 100;
»a * b

ans =
   1000
```

Variables can be reassigned, as in the example

```
» a = 10;    b = 100;
» a = a*b

a =
    1000
```

MATLAB responds to invalid expressions with a statement indicating what is wrong and a helpful suggestion, as in the example

```
» b = 100; c = 5;
» (b + c/2
??? (b + c/2
```

A closing right parenthesis is missing,
Check for a missing ")" or a missing operator.

The expression could now be retyped correctly. Alternatively, the keyboard editing facilities can be used. If the up-arrow key is pressed the previous MATLAB statement is rewritten on the screen and can be edited using the mouse in the usual way.

Most of the trigonometric and elementary mathematical functions are available for use in expressions, a partial list is:

<code>sin(X)</code>	sine of X
<code>cos(X)</code>	cosine of X
<code>asin(X)</code>	arcsine of X
<code>acos(X)</code>	arccosine of X
<code>tan(X)</code>	tangent of X
<code>atan(X)</code>	arctangent of X
<code>atan2(X, Y)</code>	four quadrant arctangent of X and Y
<code>abs(X)</code>	absolute value of X
<code>sqrt(X)</code>	square root of X

The arguments of the trigonometric functions are in radians.

1.3 Statements and variables Statements have the form

```
» variable=expression
```

The equals symbol implies the assignment of the expression to the variable. A typical statement is

```
» x = (1 4) <enter>
```

```
x =
```

```
    1    4
```

Here the numbers 1 and 4 are assigned to an array (that is, a list of numbers) with the variable name x.

The statement is executed immediately after the enter key is pressed. The array x is automatically displayed after the statement is executed. If the statement is followed by a semi-colon then the array is not displayed. Thus the statement

```
» x = [1 4]; <enter>
```

would not result in any output on screen. The assignment will have been carried out however and may be used later. The statement

```
» x
```

would then result in the output

```
x =  
    1    4
```

Array elements can be any MATLAB expression. For example

```
» x = [-1.3 3^2 (1+2+3)*4/5]
```

```
x =  
-1.3000    9.0000    4.8000
```

Individual array elements can be referenced with indices inside brackets, Thus continuing the previous example

```
» x(5) = -2*x(1)
```

```
x =  
-1.3000    9.0000    4.8000         0    2.6000
```

Notice that the size of the array has been automatically increased to accommodate the new element and that the undefined intervening elements, in this case $x(4)$, have been set to zero.

1.4 Elementary functions

Most of the trigonometric and elementary mathematical functions are available for use in expressions, a partial list is:

sin (X)	sine of the elements of X
cos (X)	cosine of the elements of X
asin (X)	arcsine of the elements of X
acos (X)	arccosine of the elements of X
tan (X)	tangent of the elements of X
atan (X)	arctangent of the elements of X
atan2 (X, Y)	four quadrant arctangent of the elements of X and Y
abs (X)	absolute value of the elements of X
sqrt (X)	square root of the elements of X
imag (X)	imaginary part of the elements of X
real (x)	real part of the elements of X
conj (X)	complex conjugate of the elements of X
log (X)	natural logarithm of the elements of X
log10 (X)	logarithm base 10 of the elements of X
exp (X)	exponential of the elements of X

These functions can be included in any expression. Note that the argument X of the functions is in general an array and so the result is an array with element-by-element correspondence to the elements of X as can be seen in the example, .

```
»sin ([0 1]) <enter>
ans =
    0    0.8415
```

Variable names begin with a letter, which may be followed by any number of letters and numbers (including underscores), although MATLAB only remembers the first 19 characters. It is good practice to use meaningful names, but not to use long names.

Since MATLAB is case sensitive, the variables A and a are different. All the predefined functions, such as those listed above have lowercase names.

MATLAB has several predefined variables including the following:

```
Pi  $\pi$ 
NaN not-a-number
Inf  $\infty$ 
i  $\sqrt{-1}$ 
j  $\sqrt{-1}$ 
```

Although it is not recommended these predefined variables can be overwritten, they can be reset to their default values by using the **clear** function, for example

```
» clear pi;
```

NaN results from undefined operations, for example

```
» 0/0
```

```
Warning: Divide by zero
ans =
    NaN
```

Variables are stored in the *workspace*. The **who** function gives a list of the variables in the workspace and the **whos** function gives additional information such as the number of elements in an array and the amount of memory occupied. The **clear** function by itself removes all variables and functions from the workspace, **clear variables** removes all variables from the workspace, and **clear name1 name2 ...**removes the particular named variables in the list.

All computations in MATLAB are performed in *double precision*. However, the screen output can be displayed in several formats. The default contains four digits past the decimal point for non-integers. This can be changed using the format function as shown in the example below

```
» pi
ans =
    3.1416
»format long; pi
ans =
    3.14159265358979
```

```

>format short e; pi
ans =
    3.1416e+000

>format long e; pi
ans =
    3.141592653589793e+000

```

1.5 Array operations

Arrays with the same number of elements can be added and subtracted. This means adding and subtracting the corresponding elements in the arrays as in the following example

```

> x = [1 2 3];   y = [4 5 6];
> z = x + y

z =
    5     7     9

```

This kind of operation is called an element-by-element operation and is very useful for setting up tables of values for graph plotting. Attempting to perform element-by-element operations on arrays containing different numbers of elements will result in an error message.

Addition, subtraction, multiplication and division of an array by a scalar (an array with one element) is allowed and results in the operation being carried out on every element of the array. Thus continuing the previous example

```

> w = z -1

w =
    4     6     8

```

When the element-by-element operations involve multiplication, division and power, the operator is preceded by a dot. This is to avoid ambiguities which would otherwise arise in more advanced work. An example involving element-by-element multiplication and power is as follows:

```

> A= [1 2 3];
> B= [-6 7 8];
> A.*B

ans =
   -6    14    24
> A.^2

ans =
    1     4     9

```

If the semi-colon is omitted the array will be output on the screen. This facility is useful for generating tables of data. The following example generates a table of x against y where $y = x \sin(x)$, for $x=0, 0.1, 0.2, \dots, 0.5$

```

> x = [0: 0.1: 0.5]
x =
    0    0.1000    0.2000    0.3000    0.4000    0.5000
> y = x.*sin(x)
y =
    0    0.0100    0.0397    0.0887    0.1558    0.2397

```

1.6 Graphics

Graphics play an important role in the design and analysis of engineering systems. The objective of this section is to present the basic x-y plotting capability of MA TLAB. A *graph display* is used to present plots and is brought up automatically when the **plot** function is used. The user can switch from graph display to command display using the mouse or the **clg** function and back to the graph display using the mouse or the **shg** function.

Available plot functions include:

<code>plot (x, y)</code>	plots the array x versus the array y,
<code>semilogx (x, y)</code>	plots the array x versus the vector y with \log_{10} scale on the x-axis,
<code>semilogy (x, y)</code>	plots the array x versus the vector y with \log_{10} scale on the y-axis,
<code>loglog (x, y)</code>	plots the array x versus the vector y with \log_{10} scale on both axes.

The axis scales and line types are automatically chosen. Graphs may be customised using the following functions:

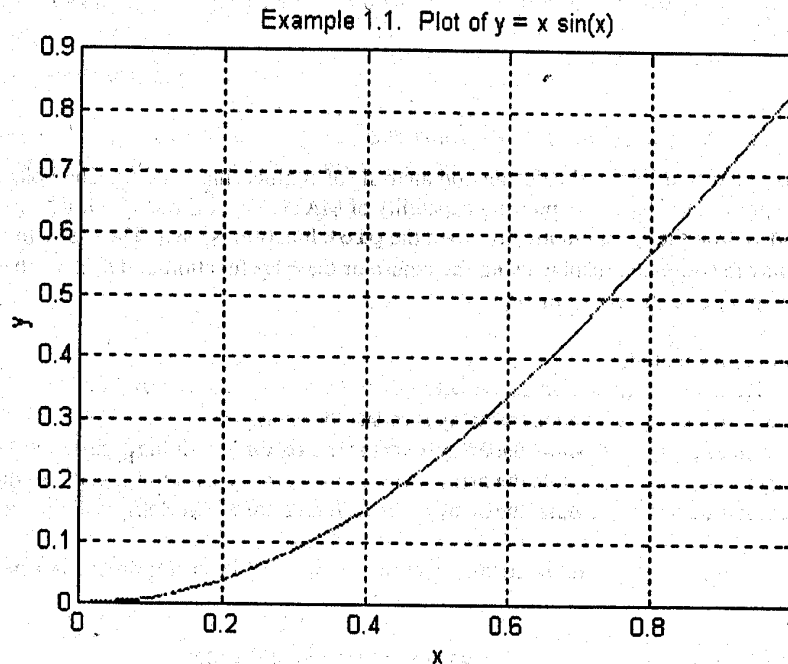
<code>title (, text')</code>	puts 'text' at the top of the plot,
<code>xlabel (, text I)</code>	labels the x-axis ,with 'text',
<code>ylabel (, text')</code>	labels the y-axis ,with 'text',
<code>text (p, q, 'text', 'sc')</code>	puts 'text' at (p,q) in screen co-ordinates,
<code>subplot</code>	divides the graphics ,window, and
<code>grid</code>	draws grid lines on the current plot.

Screen co-ordinates define the lower left corner as (0.0, 0.0) and the upper right corner as (1.0, 1.0).

Example 1.1

A basic x-y plot of $y = x \sin(x)$ versus x is obtained as follows:

```
> x = [0: 0.1: 1.0];
> y = x.*sin(x);
> plot (x, y)
>title('Example 1.1. Plot of Y = x sin(x)')
> xlabel ( 'x' )
> ylabel ( , y' )
> grid
```



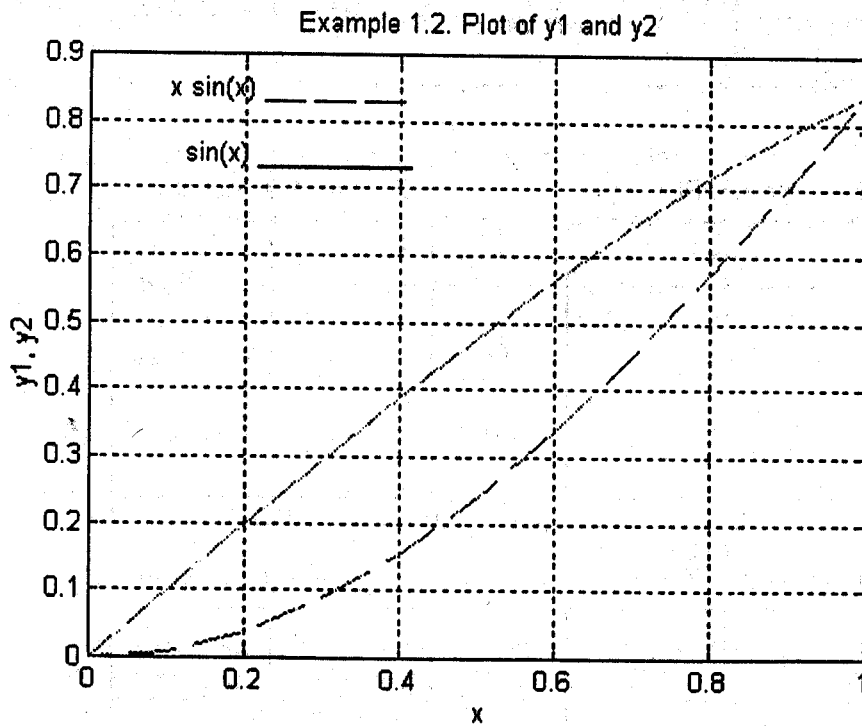
Plots may include more than one line and line types may be specified in the plot statement using the characters listed below :

- solid line,
- dashed line.
- : dotted line. and
- . dash dot line.

Use is made of this facility in the following example.

Example 1.2

```
» x = [0: 0.1: 1.0];  
» y1 = x.*sin(x); ,  
» y2 = sin(x);  
» plot{x,y1, '___',x,y2, '-. ' )  
» title{'Example 1.2. Plot of y1 and y2'}  
» xlabel (' x ' )  
» ylabel{'y1, y2'}  
» text{0.1,0.85, 'y1 = x sin(x) ---'}  
» text{0.1,0.75, 'y2 = sin(x) '-. ' )  
» grid
```

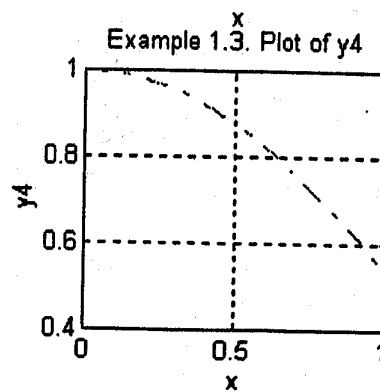
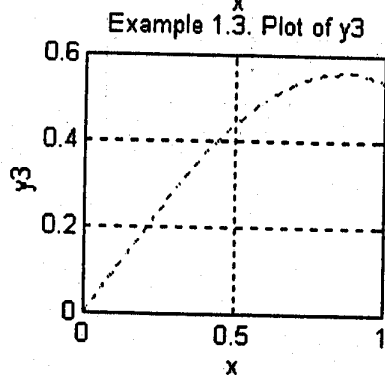
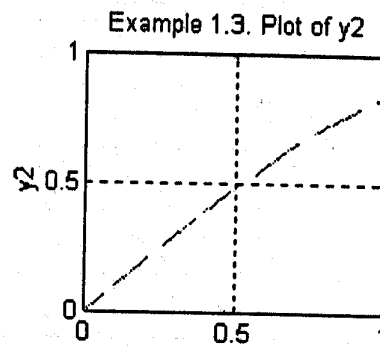
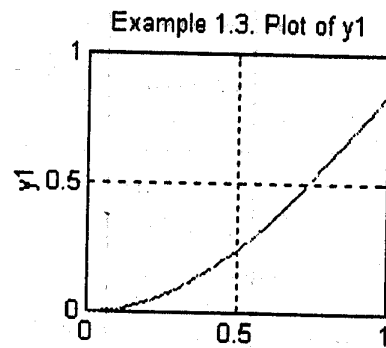


The graph display can be divided into two or four smaller windows using the subplot (mnp) function. The effect is to divide the graph display into an m x n grid of smaller windows, where m,n = 1 or 2.

This facility is illustrated in the next example.

Example 1.3

```
» x= [0: 0.1: 1.0];
» y1 = x.*sin(x);
» y2=sin(x);
» y3 = x.*cos(x);
» y4 = cos (x) ;
» subplot(221), plot(x,y1,'-')
» title('Example 1.3. Plot of y1')
» xlabel ('x')
» ylabel ('y1')
» grid
» subplot(222), plot(x,y2,'--') ..
» title ('Example 1.3. Plot of y2')
» xlabel ('x')
» ylabel ('y2')
» grid
» subplot(223), plot(x,y3,':')
» title('Example 1.3. Plot of y3')
» xlabel ('x')
» ylabel ('y3 ')
» grid
» subplot(224), plot(x,y4, '-.')
» title ('Example 1.3. Plot of y4')
» xlabel ('x')
» ylabel ('y4 ')
» grid
```



1.7 Command-line editor

To simplify the process of entering commands to MATLAB, use can be made of the arrows keys on the keypad to alter mistyped commands and to recall previous command lines. In the following example, the function `sqrt` was misspelt. This could have been corrected by retyping the entire line. Instead the up-arrow key was pressed to recall the previous line, thereby displaying the misspelled command, and the cursor was moved to the appropriate place for the insertion of the missing `r`. After pressing `<enter>` the command returns the answer.

```
» log (sqrt (atan2 3, 4))
?? Undefined function or variable sqrt,
» log(sqrt(atan2(3,4)))

ans =
    -0.2204
```

1.8 Word processing

This document was created using the word processing program "Word for Windows". The short sections of MATLAB were copied to the Windows clipboard using the MATLAB Command Window Edit Menu item **Copy**, and entered into the Word document using the Word Window Menu Edit item **Paste**. The MATLAB graphs were copied to the Windows clipboard using the MATLAB Figure Window Edit Menu item **Copy to Bitmap**, and entered into the Word document using the Word Window. Edit Menu item **Paste**.

Problems 1

1.1 Use MATLAB to evaluate the following:

(i) $100 + 5^3$, (ii) $\frac{100+5}{2+10}$ (iii) $3 \times 100 + \frac{97 \times 5}{2+20}$

1.2. Evaluate the following expressions taking the values:

$A = 100$, $B = 5$, $C = 2$, $D = 10$,

i) $\frac{A+B+C}{2D}$ ii) $A+B^C$, iii) $\frac{-1}{B(A-B)}$, iv) $\frac{AD}{BC}$

1.3. Taxi fares are calculated as the sum of a standing charge of 150p plus 30p for every half-mile over two miles. Calculate the charges for journeys of 5, 10, 3 and 2 miles.

1.4. Calculate the areas of circles of radius 0.1, 0.2, ...0.5 metres.

1.5. Calculate the areas of the rectangles whose lengths and breadths are given in the table of values:

length	5	10	3	2
breadth	1	5	2	0.5

1.6 Plot the graph which relates temperature in degrees centigrade to degrees Fahrenheit. Plot the graph of the inverse function which relates degrees Fahrenheit to degrees centigrade.

An introduction to MATLAB

Chapter 2 MATLAB Scripts and menus

2.1 Introduction

So far, all interaction with MATLAB has been at the MATLAB command prompt `»`. At the prompt a statement is entered, and executed when the enter key is pressed. This is the preferred way of working for short and non repetitive jobs. However, the real power of MATLAB for engineering calculations derives from its ability to execute a long sequence of commands stored in a file. These files are called m-files since the filename has the form `filename.m`. A script is one form of m-file. In fact MATLAB comes with a large collection of such files and a number of collections of such files called MATLAB Toolboxes are also available for particular applications.

Scripts are ordinary ASCII text files which can be created and edited using any text editor or word processor. Windows notepad is particularly convenient and simple to use. A script is just a sequence of statements and functions that could be used at the MATLAB command prompt. It is invoked at the command prompt by typing the filename (without the `.m` extension), and works through the sequence of statements in the script without waiting for the command prompt after each.

Suppose that it is required to plot the function $y = \sin(\omega t)$ for different values of the variable ω (the frequency). A script called `plotsine.m` is created. This is shown below.

```
%This is a script to plot the function y = sin(omega*t).
%
%The value of omega (rad/s) must be in the workspace
%before invoking the script
%
t = [0: 0.01: 1];
y = sin(omega*t);
plot(t,y)
xlabel('Time (s) ')
ylabel('y')
title (' Plot of Y = sin (omega*t)')
grid
```

This script can be used by entering a value for the variable `omega` at the command prompt and then by typing `plotsine` at the command prompt. After executing the script another value for ω could be entered and another graph plotted. This process can be related any number of times.

Scripts should be well documented with comments, so that their purpose and functionality can be readily understood sometime after their creation. A comment begins with a `%`. Comments at the beginning of a script form a header which can be displayed using the `help` function. This is illustrated by the following example.

```
Help plotsine
```

```
This is a script to plot the function y = sin (omega*t).
```

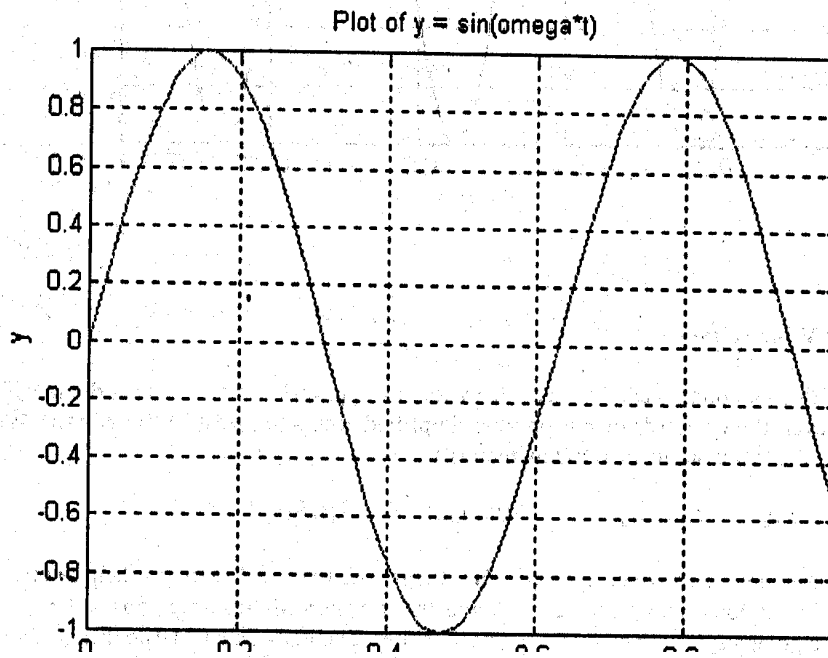
```
The value of omega (rad/s) must be in the workspace before invoking the script
```

```
» omega=10
```

```
omega =
```

```
10
```

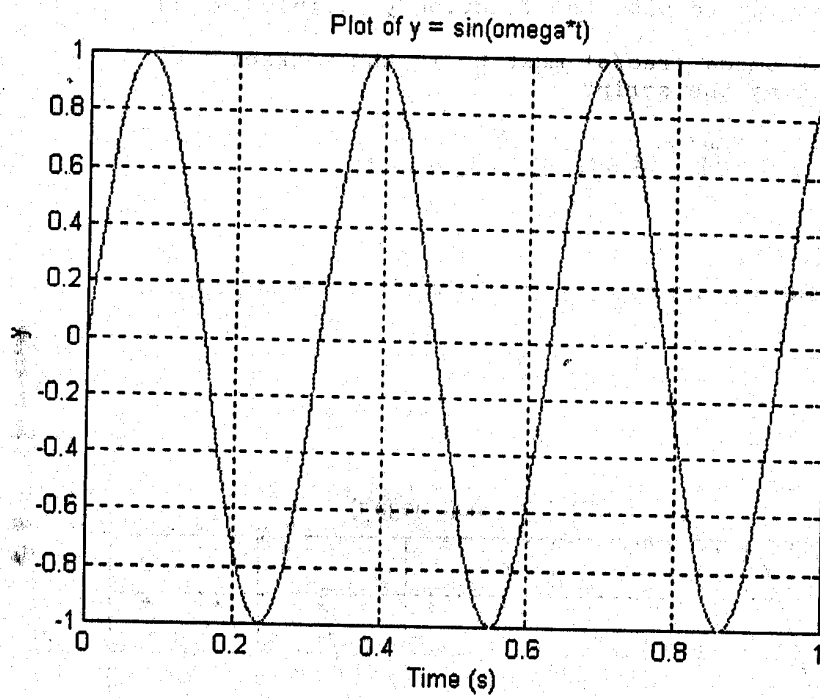
```
» plotsine
```



```

» omega=20;
» plotsine

```



2.2 Command Window menus

All of MATLAB's functionality can be accessed by issuing appropriate statements at the MATLAB prompt ». However, the window's menus provide simplified access to useful MATLAB and Windows features. The MATLAB Command Window menus are:

File Edit Options Windows Help

The **File** menu contains items **New**, **Open M-file**, **Print**, and **Printer Setup** that help to create files and set up printing options. The item **New** contains the subentry **M-file** which opens the default editor (Windows notepad) for the creation of a new m-file. The item **Open M-file** displays a file selection dialog box that asks for the name of an m-file and opens the default editor, so that amendments may be made. The item **Print** causes any selected text in the Command Window to be printed, if no text is selected, the entire session is printed. The item **Printer Setup** brings up a dialog box to enable the printer options to be set.

The **Edit** menu contains the usual Windows items **Cut**, **Copy** and **Paste**. In addition, the **Edit** menu item **Clear Session** empties the buffer containing the history of the MATLAB session.

The **Options** menu contains items **Numeric Format**, and **Editor Preference**. The item **Numeric Format** contains subentries which select between different display output formats such as described in section 1.1. The item **Editor Preference** allows the default editor to be selected. Windows notepad is convenient.

2.3 Figure Window menus

MATLAB displays all graphics output in separate windows called Figure Windows. The Figure Windows menu items are:

File Edit Windows Help

The menu items are consistent with the usual Windows items. for further details refer to the MATLAB,User's Guide.

2.4 Creating script files

Script files can be conveniently created in two ways. The obvious way is to prepare the program on a Word processor such as **Windows notepad** and to use the Windows facilities described above to run and to edit the program until it runs satisfactorily. Alternatively, use MATLAB's *diary* function. Enter MATLAB, at the command prompt type `diary on` and your interactive MATLAB session will be saved to a disk file with the pathname `matlab\bin\diary`. Note that `diary off` suspends the diary function. You can copy the diary file to your Word processor and edit it, so as to create either a report or a script file.

MATLAB program outputs can be made more informative by using the `disp` function. The purpose of this function is to display text or variable values on screen. Another useful function is `format compact`, which reduces the number of spaces that MATLAB inserts in its output. The normal display is obtained by using the function `format loose`. Use of these functions is illustrated in the following example :

Example 2.1

The following program is created and stored as script file `Ex2_1.m`

```
% Example 2.1
format compact
length = [5 10 3 2];
breadth = [1 5 2 0.5];
area = length.*breadth;
disp('    length breadth area')
disp(,    m            m        sq m')
disp([length' breadth' area'])
```

The program is then run by typing its filename at the MA TLAB command prompt

```
» ex2 1
```

```
      length      breadth      area
      m           m           sq m
5.0000      1.0000      5.0000
10.0000     5.0000     50.0000
3.0000      2.0000      6.0000
2.0000      0.5000     10.0000
```

```
»
```

Notice, that the elements of the array variables `length`, `breadth` and `area` have been printed out as columns rather than rows. The apostrophe' after the name changes the array from a row to a column. Infact, the expression `[length' breadth' area']` creates a two dimensional array comprising the three columns.

2.5 Function files

A function file differs from a script file in that arguments may be passed from the workspace to the file. and variables defined inside the file are local to the function and do not operate globally on the workspace. Function files are useful for extending MATLAB by creating new MATLAB functions using the MATLAB language. Indeed, most users develop their own 'toolbox' of function files.

The process is illustrated by the following example. The file `c2f.m` is created and stored containing the statements:

```
function f = c2f(c)
%Converts degrees Fahrenheit to degrees Centigrade
f = c*9/5 + 32;
```

The conversion is obtained by invoking the function at the MA TLAB command prompt, as for example ..

```
» c2f(14)
ans =
    57.2000
```

2.6 Further Examples

Example 2.2

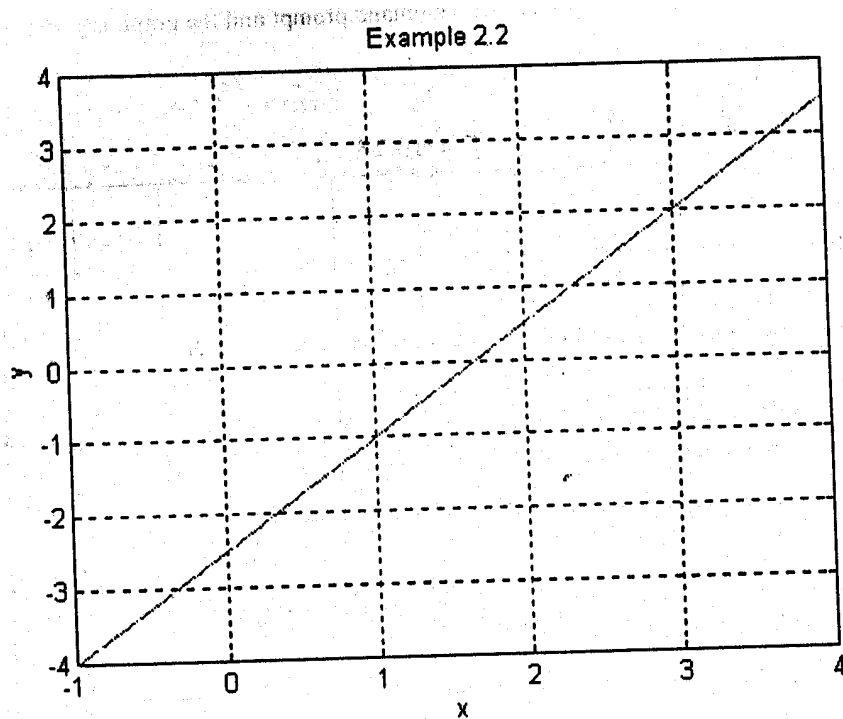
Find and plot the equation of the straight line passing through the point (3,2) and parallel to the line $2y = 3x + 4$. Determine the intercepts on the x and y axes. (see Example 1.4 of Ref I).

Create the script file `Ex2_2.m`, which is listed below

```
% This script file solves example 2.2 by creating the equation
% of the straight line y = m*x + c
%
format compact m=3/2
c=2-m*3
yintercept = c
xintercept = -c/m
x = [-1 4];
y = m*x + c;
plot(x,y)
xlabel('x')
ylabel('y')
title('Example 2.2')
grid
```

at the MA TLAB command prompt enter the filename to obtain the results

» ex2_2
m = -1.5000
c = -2.5000
yintercept = -2.5000
xintercept = 1.6667
»



Example 2.3

Draw a circle with centre (4,3) and radius 2.

The first step is to create a function file which will return the coordinates of a circle.

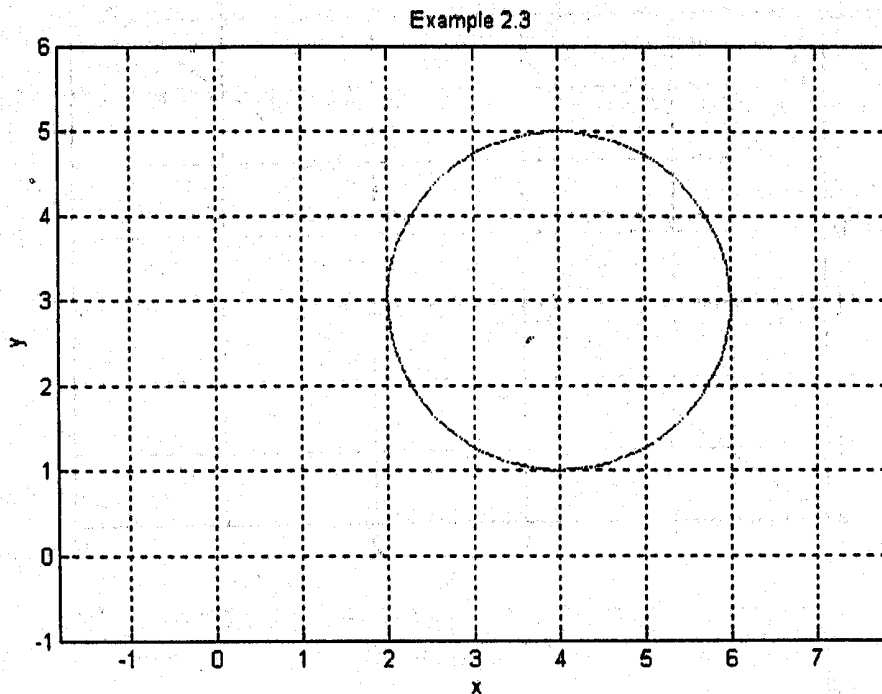
```
function [x,y] = circle(theta,a,b,r)
% Returns the x,y coordinates corresponding to the angles theta
% on the circle with centre a,b and radius r
x = a + r*cos(theta);
y = b + r*sin(theta);
```

This function can be used to return the x,y coordinates corresponding to the angles theta on the circle with centre a,b and radius r. In the following script file an array theta or angles is set up for this function so as to compute the x,y coordinates of the corresponding points on the circle.

```
% Example 2.3
theta = [0:0.1:2*pi+0.1];
[x,y] = circle(theta,4,3,2);
plot(x,y)
axis([-1 7 -1 6]);
axis('equal');
title('Example 2.3')
xlabel('x')
ylabel('y')
grid
```

This script file is then involved at the MATLAB command prompt and the graph is plotted

```
» ex2_3_
```



Example 2.4

Given the functions:

$$y = f(x) = x^2 + 2x \text{ and } y = g(x) = x - 1$$

plot the composite functions:

$$y = f(g(x)) \text{ and } y = g(f(x))$$

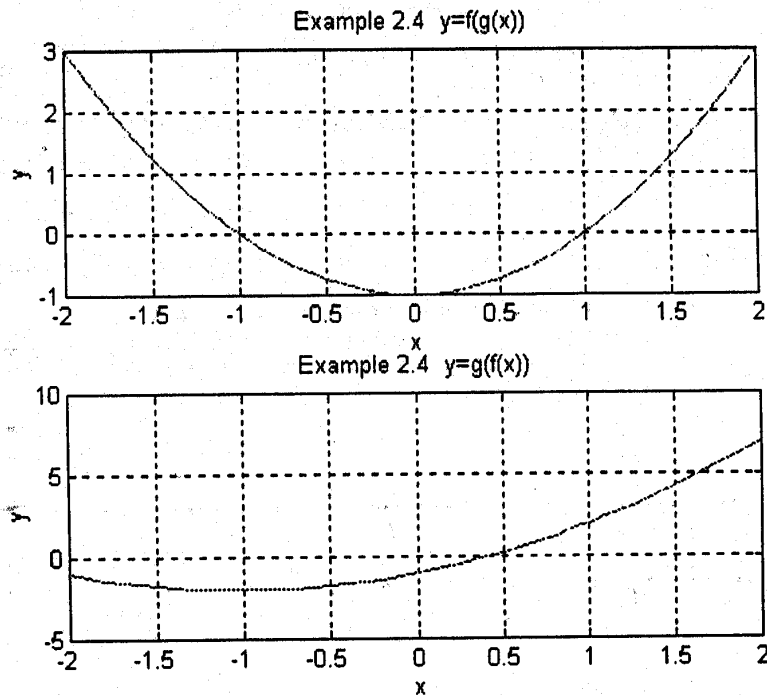
(see example 1.13 of Ref I)

A script file to plot the two composite functions is shown below :

```
% Example 2.4
x = [-2:0.1:2];
g = x - 1;
fg = g.*g + 2*g;
f = x.*x + 2*x; gf = f - 1;
subplot(211), plot(x,fg)
title('Example 2.4 y=f(g(x))')
xlabel('x')
ylabel('y')
grid
subplot(212), plot(x,gf)
title('Example 2.4 y=f(g(x))')
xlabel('x')
ylabel('y')
grid
```

To obtain the two graphs script file is invoked at the MATLAB command prompt.

» ex2_4



Problems 2

2.1 The following program is stored as script file `pr2_1.m`. It generates a signal stored as an array `s` of 101 numbers, which represent the signal starting at time $t = 0$ and increasing in steps of 0.1 seconds up to 10 seconds. The signal starts with eight values, and all the signal values after these are zero. The program then generates echoes of this signal and adds the echoes to the original signal. A rolled echo has values which are the negatives of the original signal. This replicates distortions which might occur in a two-way communication system and would complicate signal analysis.

The first echo is scaled by 0.5 and delayed by 2 seconds, the second rolled echo is scaled by -0.3 and delayed by 4 seconds, and the third echo is scaled by 0.1 and delayed by 7.5 seconds. The program then adds the echoes to the original signal and plots the result.

```
%
% This program generates three echoes from an % original signal. The sum of the original
% signal and the three echoes is then plotted.
%
%... ..program adapted from "Engineering problem solving with MATLAB" % by D M Etter, Prentice Hall, 1993
%   A. Bradshaw, 2 september, 1994
%
% = 0.0 : 0.1 : 10; % time axis stored as an array
s = zeros (size(t)); % creates an array of zeros the same size as
%           the time axis array
s (1:8) = [0 0.5 1 -0.5 0.75 0 0.2 0.1];
%           signal values
%           the first element corresponds to t = 0
%           the second element corresponds to t = 0.1
%           and so on
%
%           Generate three echoes by scaling and
%           delaying original signal.
%
echo_1 = zeros(size(t));
echo_1 (21:101) = 0.5*s(1:81);
% 2 secs delay = 20 samples
echo_2 = zeros(size(t));
echo_2 (41:101) = -0.3*s(1:61);
% 4 secs delay = 40 samples
echo_3 = zeros(size(t));
echo_3 (76:101) = 0.1*s(1:26);
% 7.5 secs delay = 75 samples
%
%           Add echoes to original signal giving new signal.
%
g = s + echo_1 + echo_2 + echo_3;
%
%           Plot new signal.
%
plot(t,g),...
title('Signal with Echoes'),...
xlabel ('t, seconds'),...
ylabel ('[g(k)]'),...
grid
```

Run this program by typing `pr2_1` at the MATLAB prompt and obtain the graph of the signal with the echoes.

2.2. Modify the program of problem 2.1 by adding an echo scaled by 0.3 and delayed by 3 seconds, and a rolled echo scaled by 0.6 and delayed by 5.5 seconds.

2.3. Modify the program of problem 2.1 to generate a signal that contains the absolute value of the signal with echoes.

2.4. Modify the program of problem 2.1 to generate a signal that contains the square of the signal with echoes.

3.1 Introduction

In this section some further work with MATLAB statements is presented. The first example plots the function in Example 1.1 of 'Modern Engineering Mathematics', by James G, Addison-Wesley, 1993.

Example 3.1

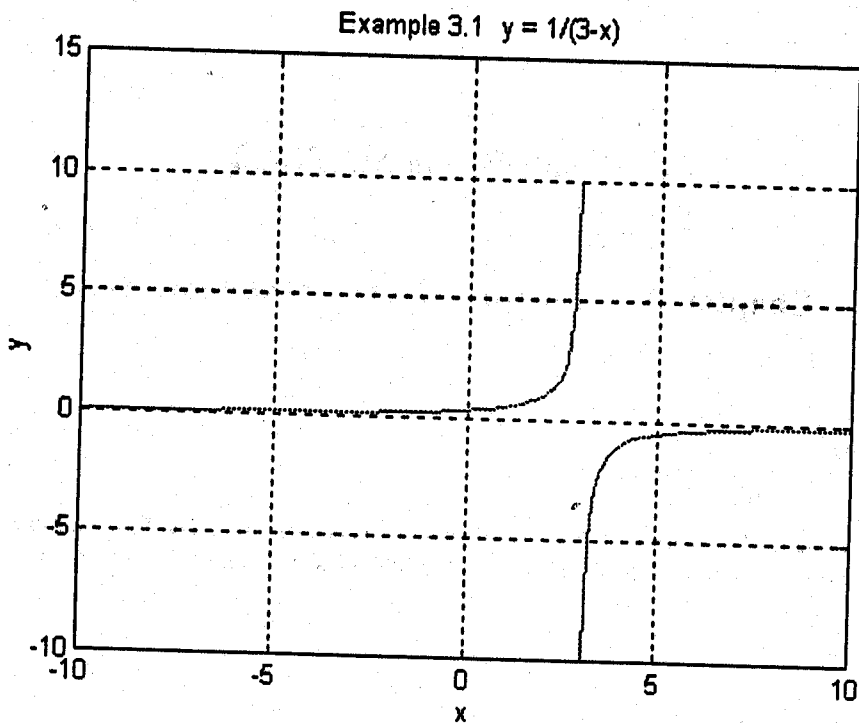
```
% This is a script to plot the function y = 1/(3-x)
%
% The filename is ex3_1.m
%
x = [-10: 0.1:10]; y = 1./(3-x);
plot (x,y)
xlabel ('x')
ylabel ('y')
title (' Example 3.1 y = 1/(3-x) ')
grid
```

This m-file is run by typing the filename at the MATLAB prompt:

```
»ex3_1
```

Warning: Divide by zero

MATLAB gives a warning, the function obviously has a pole at $x = 3$, but the graph of the function is drawn satisfactorily.



3.2 For loops

A "for" loop allows a statement, or group of statements, to be repeated a fixed predetermined number of times. For example

```
» for i = 1:5,      x(i) = 0; end, x
x =
    0  0  0  0  0
```

In the above example, 0 is assigned to the elements of the array x . Notice that four statements have been written on one line. Statements can be terminated by a comma. The inner statement is terminated by a semicolon to suppress repeated printing, while the x after the loop displays the final result. Each `for` must be matched with an `end` to close the loop.

3.3 Conditional statements

The MATLAB function `if` conditionally executes statements. The simple form is

```
if      expression
      statements
else
      statements
end
```

Its use is illustrated in the following example (see Example 1.2 of 'Modern Engineering Mathematics', by James G, Addison-Wesley, 1993).

Example 3.2

```
% This is a script to identify membership of the open set
%      (x:abs(x-3)<5}
% and the closed set
%      (x:abs(x-3)<=5}
% The filename is ex3_2.m
%
x = [-10:1:10];
[m, n] = size (x) ;
for i=1:n
    if abs(x(i) -3) < 5
        yopen(i)=1;
    else
        yopen(i)=0;
    end
    if abs (x (i) -3) <= 5
        yclosed(i)=1;
    else
        yclosed(i)=0;
    end
end
disp(' x yopen yclosed')
disp([x' yopen' yclosed'])
```

Notice that in this program use has been made of the MATLAB function `size (x)` which returns the number of rows and columns in the two dimensional array x . Note that in this program $m = 1$ since x is a single row. This saves the trouble of manually counting the number of elements in x which is required to define the `for` loop. It also means that if the variable array x is changed it is not necessary to change the `for` loop.

» ex3_2 x

x	yopen	yclosed
-10	0	0
-9	0	0
-8	0	0
-7	0	0
-6	0	0
-5	0	0
-4	0	0
-3	0	0
-2	0	1
-1	1	1
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	0	1
9	0	0
10	0	0

Of course, -1.9 and 7.9 are members of the open set.

Example 3.3

A Gas Board charges according to the tariff (see example 1.10 of 'Modem Engineering Mathematics', by James G, Addison-Wesley, 1993.):

Quarterly usage (1000 units)	Standing charge (£)	Charge (£ per 1000 units)
0 to 20	200	60
20 to 50	400	50
50 to 100	600	46
over 100	800	44

Create a function file to calculate the charge for a given usage and plot a graph of the function.

The first step is to create a function file which will return the charge for a given usage.

```
function charge = gas (usage)
% Returns the charge for the given usage of gas
% The one dimensional array 'charge' is the same size as 'usage'
[m, n] = size (usage) ;
charge = zeros(m,n);
for i = 1:n
    if usage(i)<20
        charge(i)=200+80*usage(i);
    elseif usage(i)<50
        charge(i)=600+60*usage(i);
    elseif usage(i)<100
        charge(i)=1600+40*usage(i);
    else
        charge(i)=3600+20*usage(i);
    end;
end;
```

This function can be used to return the set of charges corresponding to the set of usages. The function is piecewise linear. In the following script file an array charge is set up for this function so as to compute the charges for the corresponding usages at the points of discontinuity.

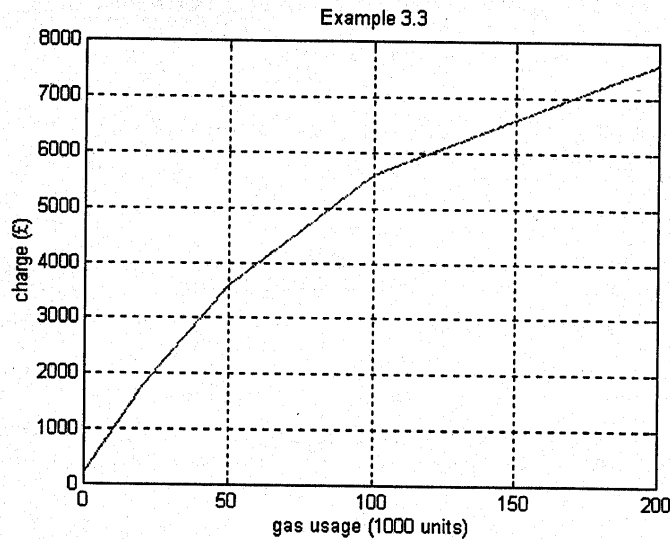
```

% Example 3.3
usage = [20 50 100 200];
charge = gas (usage);
plot(charge,usage)
title('Example 3.3')
xlabel('gas usage (1000 units) ')
ylabel('charge (£) ')
grid

```

This script is the invoked at the MATLAB command prompt and the graph is plotted.

```
»ex3_3
```



3.4 Complex numbers

MATLAB allows complex numbers, and uses *i* and *j* to represent the square root of minus one. Complex numbers can arise in the solution of quadratic equations, as illustrated in the following example (see Example 1.3 of 'Modern Engineering Mathematics', by James G, Addison-Wesley, 1993.):

Example 3.4

The first step is to create a MATLAB function which will return the roots of a quadratic equation.

```

function [x1,x2] = quadeq(a,b,c)
% Solves ax^2 + bx + c = 0
y = sqrt(b^2 -4*a*c);
x1 = (-b + y)/(2*a);
x2 = (-b -y)/(2*a);

```

This function can then be used at the MATLAB command prompt.

```

» [x1,x2]=quadeq(1,2,3)
x1 = -1.0000 + 1.4142i
x2 = -1.0000 -1.4142i

```

Notice that *y* in function `quadeq` may be either real or imaginary, or possibly zero. If *y* is imaginary then *x1* and *x2* will be complex as in the case above, otherwise *x1* and *x2* will be real as in the case below :

```

» [x1,x2]=quadeq(1,-3,2)
x1 = 2
x2 = 1

```

Example 3.5

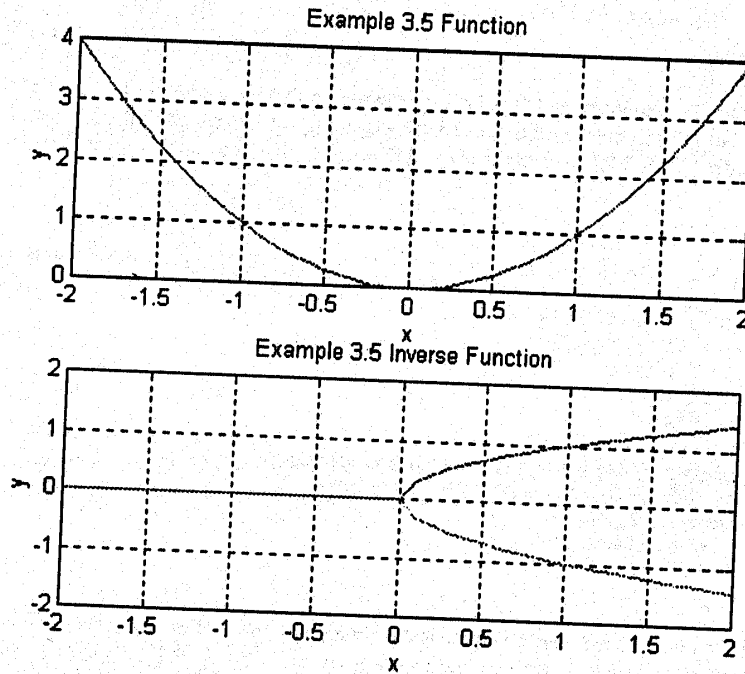
Plot the graph of the function $y=x^2$, and plot the graph of its inverse function. (see example 1.12 of "Modern Engineering Mathematics" by James F. Addison-Wesley, 1993)

A script file to plot the function and its inverse is shown below

```
% Example 3.5
x = [-2:0.1:2];           % set range of x
y= x.*x; %               calculate function
subplot(211), plot(x,y)
title('Example 3.5 Function')
xlabel('x')
ylabel('y')
grid
y1= real(sqrt(x)) % calculate inverse function
y2= real(-sqrt(x)) % calculate inverse function
subplot(212), plot(x,y1,x,y2)
title('Example 3.5 Inverse Function')
xlabel('x')
ylabel('y')
grid
```

Notice the allowance for the positive and negative square roots, and the elimination of the imaginary roots by the use of the MATLAB function `real` which returns the real part of the argument. The graphs are plotted by invoking the script file at the MATLAB command prompt:

» ex3 5



3.5 Polynomials

The roots of a polynomial function can be obtained by using the MATLAB function `roots c`, the argument `c` is a one dimensional array containing the coefficients of the polynomial.

$$c_1x^n + c_2x^{n-1} + \dots + c_nx + c_{n+1}$$

Conversely, the coefficients of a polynomial can be obtained from the roots by using the MATLAB function `poly r`, the argument `r` is a one dimensional array containing the roots of the polynomial.

Example 3.6

Find the roots of the polynomial $x^3 - 3x^2 + 6x - 4$

(see example 1.15 of 'Modern Engineering Mathematics', by James G, Addison-Wesley, 1993.)

At the MATLAB command prompt the coefficients of the polynomial are entered as array `c1`, the function `roots` is invoked and the roots are returned in the array `r`.

```
> c1 = [1 -3 6 -4];
> r = roots (c1)
r =
    1.0000 + 1.7321i
    1.0000 - 1.7321i
    1.0000
```

If the function `poly` is now invoked with the array `r` as the argument the coefficients of the original polynomial are obtained in array `c2`.

```
> c2 = poly (r)
c2 =
    1.0000   -3.0000    6.0000   -4.0000
```

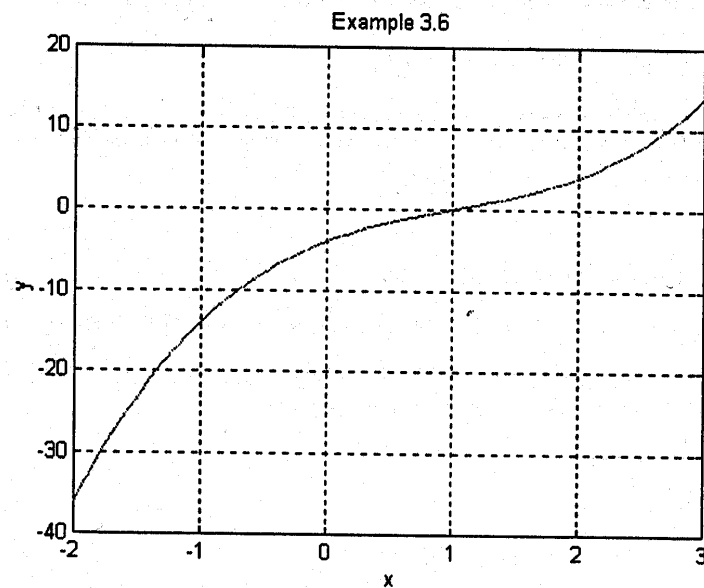
Given the coefficients in an array `c` as above the polynomial function can be plotted over the range `x` using the MATLAB function `polyval (c, x)`. Below is a script file for this purpose

```
% Example 3.6
c = [1 -3 6 -4];
x = [-2:0.1:3];
y = polyval(c,x);
plot(x,y)
title('Example 3.6')
xlabel('x')
ylabel('y')
grid
```

Invoking this script at the MATLAB command prompt will cause the graph to be plotted.

```
> Ex3_6
```

Example 3.6



Problems 3

3.1. An active sonar system transmits pulses of a sinusoidal signal of a specified frequency according to the equation.

$$s(t) = \begin{cases} \sqrt{\left(\frac{2E}{P_d}\right)} \cos(2\pi f_c t) & 0 \leq t \leq P_d \\ 0 & P_d < t \end{cases}$$

where E is the transmitted energy in joules, P_d is the pulse duration in seconds, and f_c is the frequency in Hz. The following program allows the user to enter values for E , P_d and f_c ; and to sample the signal in such a way that there are 10 samples for each period of the sine wave. The signal is then completed by adding 200 points of silence and the result is plotted out.

```
%
% ..... Program adapted from "Engineering Problem Solving with MATLAB" % by D M Etter,
%           Prentice Hall, 1993
%
% ..... A Bradshaw, 2 September 1994
%
% This program generates a sonar signal
% using values obtained from the user for
% energy, pulse duration, and frequency.
% An additional 200 points of silence are
% added to the signal before it is plotted.
%
energy = input ('Enter energy in joules");
duration = input ('Enter pulse duration in seconds');
fc = input ('Enter frequency in Hz");
%
% Generate sonar signal.
%
A = sqrt(2*energy/duration);
period = 1/fc;
t_incr = period/10;
t = [0:t_incr:duration];
s = A*cos(2*pi*fc*t);
%
% Generate and add 200 points of silence.
%
last = length(t);
silence_start = t(last) + t_incr;
silence_end = t(last) + 200*t_incr;
silence = [silence_start:t_incr:silence_end];
t = [t silence];
s = [s zeros(size(silence))];
%
% Plot sonar signal.
%
plot(t,s), ...
title('Sonar Signal'),...
xlabel('t, seconds'),...
ylabel('[s(k)]'),...
grid
```

The program is stored as a script file with the name pr3_1.m. Run the program for a pulse of 500 joules, duration 0.005 seconds and frequency 3500 Hz.

3.2. Modify the program of problem 3.1 by adding 2 milliseconds of the sine wave after the 200 points of silence.

An introduction to MA TLAB

Chapter 4 Engineering Problem Solving

4.1 Engineering Challenges

Engineers use scientific principles to make things work. Major achievements of engineers over the last 25 years include:

1. Moon landing, probably the most complex engineering project ever attempted;
2. Application satellites, used for weather reporting, communications, earth resource mapping, navigation;
3. Microprocessors, miniature computers used in almost every modern product;
4. Computer Aided Design and Manufacture, increasing the speed and efficiency of a wide range of manufacturing processes;
5. Concorde, still the only supersonic transport aircraft 25 years after its first flight;
6. Advanced Composite Materials, provide lighter, stronger, and more temperature-resistant materials;
7. Modern Jet Airliners, have made mass international travel available;
8. Lasers, light waves that travel in phase forming a narrow beam that can easily be directed and focused;
9. Fibre-Optic Communications, glass threads that can carry more information than radio waves or electric waves in a copper telephone wires;
10. Space shuttle, probably the most complex machine ever built.

Major challenges remaining:

1. Mars landing, even more complex than moon landing;
2. Aero-space plane, to carry passengers on sub-orbital and orbital flights;
3. Robots, able to speak and understand speech, able to see and understand images, able to walk and to work in an environment built for humans;
4. Exploitation of undersea resources, there is vast store of mineral wealth under the sea, viable means of extraction need to be developed;
5. Nuclear Fusion, for power generation.

4.2 Engineering Problem Solving

A systematic approach to engineering problem solving using MA TLAB and involving the following five steps is recommended:

1. Problem Statement, a clear, concise statement of the problem;
2. Input/Output Description, a description of the given information and the information to be computed;
3. Hand Example, work the problem by hand using a simple set of data;
4. MATLAB solution, use the appropriate MATLAB commands to obtain the solution;
5. Test the solution, using a variety of data sets.

4.3 Storing Data

Arrays of data can be stored as ASCII files for use by another program using the save command, such as in the example:

```
save data1.dat z /ascii
```

each row of the array z will be written to a separate line in the data file. Alternatively, the data file may be created using a word processor or an editor.

The information contained in a data file can be read by a MATLAB program using the load command, for example

```
load data1.dat
```

would store the data values in the array data_1 with the same number of rows and columns as in the data file ..

4.4 Formatted output

The fprintf command gives more control over printed output than the disp command used in example 2.1. The general form of this command is as follows:

```
fprintf(string, variables)
```

where string contains text and format specifications enclosed in single quotation marks and variables is a list of arrays to be printed. The format specifiers % e, % f, and % g are used to place the variables in the text and to define the number format as follows:

- % e exponential notation,
- % f fixed decimal point notation,
- % g either of the above depending upon which is shorter.

A newline is specified by the string \n.

A simple example of the `fprintf` command is as follows:

```
> temp = 78;
> fprintf('The temperature is %f degrees F \n',temp)
The temperature is 78.000000 degrees F
>
```

Use of the newline specifier is illustrated by the following example:

```
> temp = 78;
> fprintf('The temperature is \n %f degrees F \n',temp)
The temperature is
78.000000 degrees F
>
```

The format specifiers can also be used to control the number of decimal places and the number of characters for printing numbers as shown in the following example:

```
> temp = 78;
> fprintf('The temperature is %4.1f degrees F \n',temp)
The temperature is 78.0 degrees F
>
```

In this example the value of `temp` is printed with 4 characters, one of which is the decimal point, and with one decimal place.

Example 4.1 Optical Fibres

If light is directed into one end of a long rod of glass or plastic, the light is totally reflected by the walls, even when the rod is bent, until it emerges at the other end. Such a "light pipe" is actually composed of two materials, the material of the rod and the material of the surrounding medium. When light passes from one material to another material with a different density it bent, or refracted, at the interface of the two materials. The angle of refraction depends upon the indices of refraction of the materials and the angle of incidence of the light. If the light striking the interface comes from the within the denser material, it may reflect off the interface rather than pass through it. The angle of incidence where the light will be reflected from the surface rather than pass through it is called the critical angle θ_c . This critical angle depends on the index of refraction n_2 of the surrounding medium and the index of refraction n_1 of the rod according to the equation

$$\sin(\theta_c) = \frac{n_2}{n_1}$$

and if $n_2 > n_1$ the pipe will not transmit the light.

The problem is to write a MATLAB program that will read a data file containing indices of refraction of possible light pipe materials and determine whether light will be transmitted and, if so, for what angles of light entering the pipe.

1. Problem Statement.

Determine whether or not specified materials will form a light pipe. If they do form a light pipe, compute the angles at which light can enter the pipe and be transmitted.

2. Input/Output Description.

The input is a data file containing the indices of refraction of the pairs of potential light pipe materials. The output is a message indicating whether or not light is transmitted and the angles at which it can be transmitted.

3. Hand Example.

Consider a glass rod in air. The index of refraction for air is 1.0003 and the index for glass is 1.5. Thus for a light pipe consisting of glass in air, the critical angle is calculated as follows:

$$\theta_c = \sin^{-1}\left(\frac{n_2}{n_1}\right) = \sin^{-1}\left(\frac{1.0003}{1.5}\right) = \sin^{-1}(0.66687) = 41.82$$

This light pipe will transmit light for all angles of incidence greater than 41.82 degrees.

4. MATLAB SOLUTION

The following program is written and stored as script file ex4_1.m

```
%
%   This program reads the indices of refraction
%   for materials forming a light pipe. For each
%   pair of materials, we determine if the pipe
%   will transmit light, and at what angles of
%   incidence on degrees.
%
%   Program adapted from "Engineering Problem Solving with MATLAB
%   by D M Etter, Prentice Hall, 1993
r2d = 189/pi           % convert radians to degrees
load indices.dat      % read data into indices
n1 = indices (:,1);   % first column of data array
n2 = k=1:length (n1) % second column of data array
for k=1:length(n1)
    if n2(k) > n1(k)
        fprintf('Light is not transmitted for \n')
        fprintf('rod index %g and medium index %g \n\nl,...
n1(k),n2(k))
    else
        critical angle = asin(n2(k)/n1(k))*r2d;
        fprintf('Light is transmitted for \n')
        fprintf('rod index %g and medium index %g \nl,...
n1(k),n2(k))
        fprintf('for angles greater than %g degrees \n\nl,...
critical angle)
    end
end
```

5. Testing

The program is tested with indices. dat a data file containing 3 pairs of indices as follows:

```
1.31    1.473
1.5     1.0003
1.49    1.33
```

At the MA TLAB prompt type the name of the script file to test the program as follows:

```
» ex4 1
Light-is not transmitted for
rod index 1.31 and medium index 1.473

Light is transmitted for
rod index 1.5 and medium index 1.0003
for angles greater than 41.8257 degrees

Light is transmitted for
rod index 1.49 and medium index 1.33
for angles greater than 63.204 degrees
```

4.5 While loops

The general format for a while loop is as follows:

```
while expression
    statements
end
```

If the expression is true then the statements are executed. After the statements are executed the condition is retested. If the condition is still true the statements are executed again. When the condition is false the program skips to the statements following the end of the while loop.

As an illustration of the use of a while loop consider a program which prints out a number (10) of messages:

```
» k = 1;
» while k <=10
    fprintf ('Time #%2.0f: Hello world \n',k);
    k = k + 1;
end
Time # 1: Hello world
Time # 2: Hello world
Time # 3: Hello world
Time # 4: Hello world
Time # 5: Hello world
Time # 6: Hello world
Time # 7: Hello world
Time # 8: Hello world
Time # 9: Hello world
Time #10: Hello world
```

4.6 Logical expressions

MATLAB has six relational operators which can be used to compare values in logical expressions, they are :

Relational Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

If the variables to be compared are scalars the result is 1 if true or 0 if false. If the variables to be compared are arrays, they must be of the same dimension, the corresponding array elements are compared and the result is an array of the same dimension with 0's as its elements according to the result of the element by element comparison. This is illustrated in the following :

```
» a = [2 4 6];
» b = [3 5 1]
» a<b

ans =

     1     1     0
» a~=b

ans =

     1     1     1
»
```

Logical expressions can be preceded by the logical operator *not* ~ to change the value of the logical expression to the opposite value. This is illustrated in the following :

```
» a = [2 4 6]
» b = [3 5 1]
» ~(a,b)

ans =

     0     0     1
»
```

MATLAB has some useful logical functions. Two of these are :

any(x) for each column of the array x this function returns 1(true) if any elements of the array are nonzero or 0 (false) otherwise

all(x) for each column of the array x this function returns 1(true) if all the elements of the array are nonzero or 0 (false) otherwise

These functions are illustrated in the following

```
> b = [1 0 4
       0 0 3
       8 7 0];
> any (b)

ans =

       1       1       1
> all (any (b))

ans =

       1
> any (all (b))

ans =

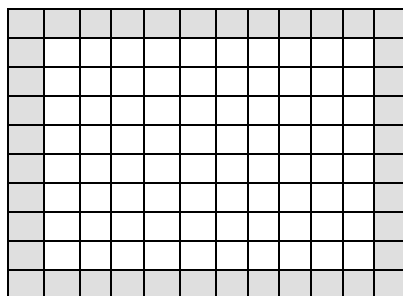
       0
>
```

A further illustration is as follows :

```
> a = [1 0 0
       1 2 0
       1 2 2];
> if all (a) == 0
    disp ('a contains all zeros')
end
>
> a = [0 0 0
       0 0 0
       0 0 0 ];
> end
a contains all zeros
>
```

Example 4.2 : Temperature distribution

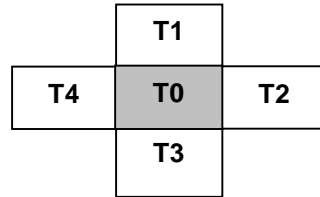
Consider the temperature distribution in a rectangular plate in which the temperatures of three sides are maintained at a constant value and the temperature of the remaining side is maintained at a different value. The problem can be solved by imagining a rectangular grid drawn over the plate as shown below :



The problem is to compute the temperature of each rectangular section of the plate, given the temperatures of the shaded sections. The numerical procedure, known as relaxation, is to assume values for the unknown temperatures of the sections within the plate and to compute a new temperature for each of the unshaded sections in turn using this formula.

$$T_0 = \frac{T^1 + T^2 + T^3 + T^4}{4}$$

Where t_0 is the new temperature of a section and T_1, T_2, T_3 and T_4 are the current temperatures of the adjacent sections as indicated below :



This process is carried on repeatedly, continually sweeping across and down the plate covering all sections until the calculated temperatures cease to alter by specified (small) amount called the tolerance. This is an example of iteration, a commonly used numerical technique of repeated calculation until a sufficiently accurate result is obtained. The particular formula to be used depends upon the particular problem to be solved.

1. Problem statement

Determine the equilibrium temperature distribution for a rectangular plate with fixed temperatures on its sides.

2. Input / output distribution

The input is the size of the grid, the temperatures of the sides, and the iteration tolerance. The output is the set of temperature values for the sections of the plate.

3. Hand example

To be sure that the process is understood carryout two iterations using a coarse grid. Assume four rows and four columns, that the top and the left and right hand sides are at 100 degrees and that the bottom is at 50 degrees. Take the initial values of the temperatures to be computed as 0 degrees. Common sense indicates the closer the initial guess is to the actual values the fewer the number of iterations required to achieve the desired accuracy. However, the ultimate success of the procedure should not be dependent on the initial values.

Initial Temperatures

100	100	100	100
100	0	0	100
100	0	0	100
50	50	50	50

First iteration

100	100	100	100
100	50	50	100
100	37.5	37.5	100
50	50	50	50

note that the biggest change is $50 - 0 = 50$

Second iteration

100	100	100	100
100	781.875	781.875	100
100	59.375	59.375	100
50	50	50	50

note that the biggest change is $71.875 - 50 = 21.875$

4. MATLAB SOLUTION

The following program is written and stored as script file ex4_2.m

```
%
% This program initializes the temperature in a
% metal plate and determines the equilibrium
% temperatures based on a tolerance value.
%
% Program adapted from "Engineering Problem Solving with MATLAB"
% by D M Etter, Prentice Hall, 1993

nrows = input ('Enter number of rows');
ncols = input ('Enter number of columns');
iso1 = input ('Enter temperature for top and sides');
iso2 = input ('Enter temperature for bottom');
tolerance = input ('Enter equilibrium tolerance');
%
% Initialize and print temperature array.
%
old = zeros (nrows, ncols);
old (1,:) = iso1 + zeros (1, ncols);
old (:,1) = iso1 + zeros (nrows,1);
old (:,ncols) = iso1 + zeros (nrows,1);
old (nrows,:) = iso2 + zeros (1,ncols);
disp ('Initial Temperature');
disp (old)
new = old;
equilibrium = 0; % logical false
%
% update temperatures and test for equilibrium.
%
while ~equilibrium % not false = true
    for m = 2:nrows-1
        new (m,n) = (old (m-1,n) + old (m,n-1) +...
            old (m,n+1) + old (m+1,n))/4;
    end
end
if all (new-old <=tolerance) % check all changes
    equilibrium =1; % logical true
    disp ('Equilibrium Temperatures');
    disp (new)
end
old = new;
end
```

5. TEST THE PROGRAM

Repeat the hand calculation, at the MATLAB prompt type ex4.2

```
>> ex4_2
```

```
Enter number of rows 4
```

```
Enter number of columns 4
```

```
Enter temperature for top and sides 100
```

```
Enter temperature for bottom 50
```

```
Enter equilibrium tolerance 40
```

```
Initial Temperatures
```

```
100    100    100    100
100     0     0     100
100     0     0     100
50     50     50     50
```

Equilibrium Temperatures			
100.0000	100.0000	100.0000	100.0000
100.0000	71.8750	71.8750	100.0000
100.0000	59.3750	59.3750	100.0000
50.0000	50.0000	50.0000	50.0000

Since the tolerance was set at 40 the second iteration is accepted. A more realistic result is obtained with a smaller tolerance and smaller mesh size (more sections)

» ex4_2

Enter number of rows 6

Enter number of columns 6

Enter temperature for top and sides 100

Enter temperature for bottom 50

Enter equilibrium tolerance 1

Initial Temperatures					
100	100	100	100	100	100
100	0	0	0	0	100
100	0	0	0	0	100
100	0	0	0	0	100
100	0	0	0	0	100
50	50	50	50	50	50

Equilibrium Temperatures					
100.0000	100.0000	100.0000	100.0000	100.0000	100.0000
100.0000	96.1837	93.9040	93.9040	96.1837	100.0000
100.0000	92.0108	87.4372	87.4372	92.0108	100.0000
100.0000	86.3297	79.4841	79.4841	86.3297	100.0000
100.0000	75.7305	67.7698	67.7698	75.7305	100.0000
50.0000	50.0000	50.0000	50.0000	50.0000	50.0000

Problem 4

1. Modify program ex4_1 so that the results are printed in a table with the headings

Light Transmitting Pipes		
Rod Index	Medium Index	Angles of Incidence (degrees)

2. Modify the program ex4_2 so that the four sides can have different given temperatures and the number of iterations is printed out.