

It should be noted that the introduction to MATLAB is relatively brief and is meant only as an aid to the reader. It can in no way be expected to replace the standard MATLAB manual. A particular feature of the book is that it provides supplemental functions and scripts developed with MATLAB, v4. Thus we are able to use the new facilities provided for sparse matrices and the greatly enhanced graphics to introduce the reader to the more difficult topics. However, apart from those functions and scripts which use the most recent features of MATLAB v4 the rest are compatible with earlier versions of MATLAB. The scripts and functions we provide are, as far as we are aware, platform independent. They have been kept as simple as possible for reasons of clarity. They could be improved and we urge readers to develop the ones that are of particular interest to them. These supplemental functions and scripts are available from The MathWorks, Inc. free of charge on PC or Macintosh diskette. To request a copy please complete and return the card bound in the back of this book.

The text provides a general introduction to MATLAB; the solution of linear equations and eigenvalue problems; methods for solving non-linear equations; numerical integration and differentiation; the solution of initial value and boundary value problems; curve fitting including splines, least squares and Fourier analysis. The last chapter presents topics in optimisation such as interior point methods, non-linear programming and genetic algorithms which we hope will attract the interest of readers. We provide a broad introduction to the topics and encourage the reader to experiment with the MATLAB functions provided. The text also provides some MATLAB execution times which, in general, have been produced using a Macintosh LC II, but a table providing a comparison with other machines is given in Chapter 1.

The text contains many worked examples, practice problems and solutions and we hope we have provided an interesting range of problems. The text is suitable for undergraduate and postgraduate students and for those working in industry and education. We hope readers will share our enthusiasm for this area of study. For those who do not currently have access to MATLAB, this text still provides a general introduction to a wide range of numerical algorithms and many useful examples.

We would be pleased to hear from readers who note errors or have suggestions for improvements. We would also like to thank Karen Rose, Jacqueline Harbor of Prentice Hall and Cristina Palumbo of The Mathworks, Inc. for their help and encouragement in producing this text.

George Lindfield and John Penny
Aston University
Birmingham
July 1994

1

An introduction to MATLAB

1.1 THE SOFTWARE PACKAGE MATLAB

MATLAB® is a software package produced by The MathWorks, Inc. and is available on systems ranging from personal computers to the Cray super-computer. The name MATLAB is derived from the term "matrix laboratory". It provides an interactive development tool for scientific and engineering problems and more generally for those areas where significant numeric computations have to be performed. The package can be used to evaluate single statements directly or a list of statements called a script can be prepared. Once named and saved, a script can be executed as an entity in much the same way as a program. The package is based on software produced by the LINPACK and EISPACK projects which represent the "state-of-the-art" numerical software for matrix computations. MATLAB provides the user with:

- (1) easy manipulation of matrix structures,
- (2) a vast number of powerful inbuilt routines which is constantly growing and developing,
- (3) powerful two- and three-dimensional graphing facilities,
- (4) a scripting system which allows users to develop and modify the software for their own needs.

It is not difficult to use MATLAB, although to use it with maximum efficiency requires experience. Essentially MATLAB works with rectangular or square arrays of data (matrices), the elements of which may be real or complex. A scalar quantity is thus a matrix containing a single element. This is an elegant and powerful notion but it can

present the user with an initial conceptual difficulty. A user schooled in such languages as FORTRAN, BASIC or Pascal is familiar with a pseudo-statement of the form $A = \text{abs}(B)$ and will immediately interpret it as an instruction that A is assigned the absolute value of the number stored in B . In MATLAB the variable B may represent a matrix so that *each element* of the matrix A will become the absolute value of the corresponding element in B .

There are several software packages that have some similarities to MATLAB. These packages include:

The NAG Library. This is a very extensive, high-quality collection of subroutines for numerical analysis but some users may find it more difficult to use than MATLAB since the subroutines must be called from a programming language such as FORTRAN, BASIC or Pascal.

Mathematica, Maple and Derive. These packages place a major emphasis on symbolic mathematical manipulation. There is now a MATLAB symbolic manipulation toolbox. It consists of the embedded Maple V library and over 50 M-files which provide MATLAB versions of Maple functions for such tasks as calculus, symbolic equation solving and variable precision arithmetic. In addition there is an extended symbolic manipulation toolbox with the complete Maple V library of functions.

APL. The letters stand for **A** Programming Language. This is a language designed mainly for manipulating matrices. It contains many powerful facilities but the symbols it uses are non-standard and the syntax unusual. The keyboard must be remapped for these special characters.

Control-C and Octave. These packages are also developed from the LINPACK and EISPACK projects. Some of the commands and functions are very similar to those of MATLAB.

1.2 MATLAB ON PERSONAL COMPUTERS AND WORKSTATIONS

The current MATLAB release, version 4, is available on a wide variety of platforms and functions under a variety of operating systems. In particular it is implemented on the Sun Sparc workstation under X Windows, on IBM PC compatibles under MS Windows and on Macintosh computers. When MATLAB is invoked it opens a command window, and if required graphics, editing and help windows may also be opened. MATLAB scripts and function are in general platform independent and they can be readily ported from one system to another. To install and start MATLAB readers should consult the manual appropriate to their particular working environment.

The scripts and functions given in this book have been tested under MATLAB version 4. However, most of them will work directly using earlier versions of MATLAB but some will require modification, particularly those involving graphics and sparse matrices.

The remainder of this chapter is devoted to introducing some of the statements and syntax of MATLAB. The intention is to give the reader a sound but brief introduction to the power of MATLAB. Some details of structure and syntax will be omitted and must be obtained from the MATLAB manual.

1.3 MATRICES AND MATRIX OPERATIONS IN MATLAB

The matrix is fundamental to MATLAB and we have provided a broad and simple introduction to matrices in Appendix 1. In MATLAB the names used for matrices must start with a letter and may be followed by any combination of letters or digits. The letters may be upper or lower case. Note that throughout this text a distinctive font is used to denote MATLAB statements and output.

In MATLAB the arithmetic operations of addition, subtraction, multiplication and division can be performed directly with matrices and we will now examine these commands. First of all the matrices must be created. There are several ways of doing this in MATLAB and the simplest method, which is suitable for small matrices, is as follows. We will assign an array of values to A by opening the **command** window and then typing

```
A=[1 3 5;1 0 1;5 0 9]
```

When the return key is pressed the matrix will be displayed thus:

```
A =
     1     3     5
     1     0     1
     5     0     9
```

By typing, for example, $B=[1\ 3\ 51;2\ 6\ 12;10\ 7\ 28]$ and pressing the return key we assign values to B . All statements are terminated by pressing return. To add the matrices in the **command** window and assign the result to C we type $C=A+B$ and similarly if we type $C=A-B$ the matrices will be subtracted. In both cases the result will be printed row by row. Note that terminating a MATLAB statement with a semicolon suppresses the output. If these statements are typed in an **edit** window as part of a script (i.e. a program) then there will be no execution or output until the script is run by typing its name in the **command** window and pressing return.

The implementation of vector and matrix multiplication in MATLAB is straightforward. Beginning with vector multiplication, we will assume that row vectors having the same number of elements have been assigned to d and p . To multiply them together we write $x=d*p'$. Note that the symbol $'$ transposes the row p into a column so that the multiplication is valid. The result, x , is a scalar. For matrix multiplication, assuming the two matrices A and B have been assigned, the user simply types $C=A*B$. This will compute A postmultiplied by B , assign the result to C and display it if the multiplication is valid; otherwise MATLAB will give an appropriate error indication. The conditions for matrix multiplication to be valid are given in Appendix 1. Notice that the symbol $*$ must be used for multiplication because in MATLAB multiplication is not implied.

1.4 USING THE MATLAB OPERATOR \ FOR MATRIX DIVISION

It is easy to solve the problem $ax = b$ where a and b are simple scalar constants and x is the unknown. Given a and b then $x = b/a$. However, consider the corresponding matrix equation

$$Ax = b \quad (1.4.1)$$

where A is a square matrix. We now wish to find x where x and b are vectors. Computationally this is a much more difficult problem and in MATLAB it is solved by executing the statement

$$x = A \backslash b$$

This statement uses the important MATLAB division operator \backslash and solves the linear equation system (1.4.1).

Solving linear equation systems is an important problem and the computational efficiency and other aspects of this type of problem are discussed in considerable detail in Chapter 2.

1.5 MANIPULATING THE ELEMENTS OF A MATRIX

In MATLAB matrix elements can be manipulated individually or in blocks. For example, $X(1,3)=C(4,5)+V(9,1)$, $a(1)=b(1)+d(1)$ or $C(i,j+1)=D(i,j+1)+E(i,j+1)$ are valid statements relating elements of matrices. Rows and columns can be manipulated as complete entities. Thus $A(:,3)$, $A(5,:)$ refer respectively to the third column and fifth row of A . If B is a 10 by 10 matrix then $B(:,4:9)$ refers to columns 4 to 9 of the matrix. Note that in MATLAB, by default, the *lowest matrix index starts at 1*.

The following examples illustrate some of the ways subscripts can be used in MATLAB. First we assign a matrix

```
»a=[2 3 4 5 6;-4 -5 -6 -7 -8;3 5 7 9 1;4 6 8 10 12;-2 -3 -4 -5 -6]
```

```
a =
     2     3     4     5     6
    -4    -5    -6    -7    -8
     3     5     7     9     1
     4     6     8    10    12
    -2    -3    -4    -5    -6
```

Executing the following statements

```
»v=[1 3 5];
»b=a(v,2)
```

gives

```
b =
     3
     5
    -3
```

Thus b is composed of the elements of the first, third and fifth rows in the second column of a . Executing

```
»c=a(v,:)
```

gives

```
c =
     2     3     4     5     6
     3     5     7     9     1
    -2    -3    -4    -5    -6
```

Thus c is composed of the first, third and fifth rows of a . Executing

```
»d=zeros(3)
»d(:,1)=a(v,2)
```

gives

```
d =
     3     0     0
     5     0     0
    -3     0     0
```

Here d is a 3 x 3 matrix of zeros with column 1 replaced by the first, third and fifth elements of column 2 of a . Executing

```
»e=a(1:2,4:5)
```

gives

```
e =
     5     6
    -7    -8
```

1.6 TRANSPOSING MATRICES

A simple operation that may be performed on a matrix is transposition which interchanges rows and columns. In MATLAB this is denoted by the symbol $'$. For example:

```
»a=[1 2 3;4 5 6;7 8 9]
```

```
a =
     1     2     3
     4     5     6
     7     8     9
```


To assign the transpose of *a* to *b* we write

```
»b=a'

b =
     1     4     7
     2     5     8
     3     6     9
```

However, if *a* is complex then the MATLAB operator ' gives the complex conjugate transpose. For example:

```
»a=[1+2*i 3+5*i;4+2*i 3+4*i]

a =
 1.0000 + 2.0000i  3.0000 + 5.0000i
 4.0000 + 2.0000i  3.0000 + 4.0000i

»b=a'

b =
 1.0000 - 2.0000i  4.0000 - 2.0000i
 3.0000 - 5.0000i  3.0000 - 4.0000i
```

To provide the transpose without conjugation, we execute

```
»b=a.'

b =
 1.0000 + 2.0000i  4.0000 + 2.0000i
 3.0000 + 5.0000i  3.0000 + 4.0000i
```

1.7 SPECIAL MATRICES

Certain matrices occur frequently in matrix manipulations and MATLAB ensures that these are generated easily. Some of the most common are `ones(m,n)`, `zeros(m,n)`, `rand(m,n)` and `randn(m,n)`. These MATLAB statements generate *m* × *n* matrices composed of ones, zeros, randomly generated and normally randomly generated elements respectively. The MATLAB statement `eye(n)` generates the *n* × *n* unit matrix. If only a single scalar parameter is given, then these statements generate a square matrix of the size given by the parameter. If we wish to generate an identity matrix *B* of the same size as an already existing matrix *A*, then the statement `B=eye(size(A))` can be used. Similarly `C=zeros(size(A))` and `D=ones(size(A))` will generate a matrix *C* of zeros and a matrix *D* of ones, both of which are the same size as matrix *A*.

1.8 GENERATING MATRICES WITH SPECIFIED ELEMENT VALUES

Here we will confine ourselves to some relatively simple examples thus:

```
x=-10:1:10 sets x to a vector having elements -10, -9, -8, ..., 8, 9, 10.
y=-2:.2:2 sets y to a vector having elements -2, -1.8, -1.6, ..., 1.8, 2.
z=[1:3 4:2:8 10:0.5:11] sets z to a vector having the elements
```

```
[1 2 3 4 6 8 10 10.5 11]
```

More complex matrices can be generated from others. For example, consider the two statements

```
C=[2.3 4.9; 0.9 3.1];
D=[C ones(size(C)); eye(size(C)) zeros(size(C))]
```

These two statements generate a new matrix *D* the size of which is double that of the original *C*. The matrix will have the form

$$D = \begin{bmatrix} 2.3 & 4.9 & 1 & 1 \\ 0.9 & 3.1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

1.9 SOME SPECIAL MATRIX OPERATIONS

Some arithmetic operations are simple to execute for single scalar values but involve a great deal of computation for matrices. For large matrices such operations may take a significant amount of time. An example of this is where a matrix is raised to a power. We can write this in MATLAB as `A^p` where *p* is a positive scalar value and *A* is a matrix. This produces the power of the matrix and may be obtained in MATLAB for any value of *p*. For the case where the power equals 0.5 it is better to use `sqrtm(A)` which gives the square root of the matrix *A*. Another special operation directly available in MATLAB is `expm(A)` which gives the exponential of the matrix *A*. Other special operators are available in MATLAB which involve operations between the individual elements of a matrix. These are discussed in the following section.

1.10 ELEMENT-BY-ELEMENT OPERATIONS

Element-by-element operations differ from the standard matrix operations but they can be very useful. They are achieved by using a period or dot (.) to precede the operator. For example, `X.^Y`, `X.*Y` and `Y.\X`. If, in these statements, *X* and *Y* are matrices (or vectors) the *elements* of *X* are raised to the power, multiplied or divided by the *corresponding element* of *Y* depending on the operator used. The form `X./Y` gives the same result as the division operation specified above. For these operations to be executed the matrices and vectors used must be the same size. Note that a period is *not* used in the operations + and - because ordinary matrix addition and subtraction

are element-by-element operations. Examples of element-by-element operations are given below:

```
»a=[1 2;3 4]
```

```
a =
     1     2
     3     4
```

```
»b=[5 6;7 8]
```

```
b =
     5     6
     7     8
```

```
»a*b
```

```
ans =
    19    22
    43    50
```

This gives normal matrix multiplication. However, using the `.` operator we have

```
»a.*b
```

```
ans =
     5    12
    21    32
```

which is element-by-element multiplication.

```
»a.^b
```

```
ans =
         1         64
    2187    65536
```

In the above, each element of `a` is raised to the corresponding power in `b`.

1.11 INPUT AND OUTPUT IN MATLAB

To output the names and values of variables the semicolon can be omitted from assignment statements. However, this does not produce clear scripts or well-organised and tidy output. It is often better practice to use the function `disp` since this leads to clearer scripts. The `disp` function allows the display of text and values on the screen. To output the contents of the matrix `A` on the screen we write `disp(A)`; Text output must be placed in single quotes, for example:

```
disp('this will display this text');
```

Combinations of strings can be printed using square brackets `[]` and numerical values can be placed in text strings if they are converted to strings using the `num2str` function. For example,

```
x=2.678;
disp(['Value of iterate is ', num2str(x), ' at this stage']);
```

will place on the screen

```
Value of iterate is 2.678 at this stage
```

The more flexible `fprintf` function allows formatted output to the screen or to a file. It takes the form

```
fprintf('filename','format string',list);
```

where `list` is a list of variable names separated by commas. The filename parameter is optional; if not present output is to the screen. The format string formats the output. The elements that may be used in the format string are

```
%P.Qe for exponential notation
%P.Qf fixed point
%P.Qg becomes %P.Qe or %P.Qf whichever is shorter
/n gives new line
```

where `P` and `Q`, above, are integers. The integer string characters, including a period (`.`), must follow the `%` symbol and precede the letter. The integer before the period sets the field width; the integer after the period sets the number of decimal places after the decimal point. For example, `%8.4f` and `%10.3f` give field width 8 with four decimal places and 10 with three decimal places respectively. Note that one space is allocated to the decimal point. For example,

```
x=1007.46; y=2.1278; k=17;
fprintf('/nx= %8.2f y= %8.6f k= %2.0f/n',x,y,k);
```

outputs

```
x=1007.46 y= 2.127800 k=17
```

We now consider the input of text and data from the keyboard. An interactive way of obtaining input is to use the function `input`. This takes the forms

```
Variable=input('text'); or Variable=input('text','s');
```

The first form displays a prompt text and allows the input of *numerical values*. Single values or matrices can be entered in this way. The `input` function displays the text as a prompt and then waits for an entry. This is assigned to the variable when return is

pressed. The second form allows for the entry of *strings*.

For large amounts of data, perhaps saved in a previous MATLAB session, the function `load` allows the loading of files from disk using

```
load filename
```

1.12 MATLAB GRAPHICS

MATLAB provides a wide range of graphics facilities which may be called from within a script or used simply in command mode for direct execution. We will begin by considering the `plot` function. This function takes several forms. For example:

`plot(x,y)` plots the vector `x` against `y`. If `x` and `y` are matrices the first column of `x` is plotted against the first column of `y`. This is then repeated for each pair of columns of `x` and `y`.

`plot(x1,y1,'linetype_or_pointtype1',x2,y2,'linetype_or_pointtype2')` plots the vector `x1` against `y1` using the `linetype_or_pointtype1`; then the vector `x2` against `y2` using the `linetype_or_pointtype2`.

The `linetype_or_pointtype` is selected by using the required symbol from the following table:

Lines	Symbol	Points	Symbol
solid	-	point	.
dashed	-	plus	+
dotted	:	star	*
dashdot	-.	circle	o
		x mark	x

Semilog and loglog graphs can be obtained by replacing `plot` by `semilog` or `loglog` and various other replacements for `plot` are available to give special plots.

Titles, axis labels and other features can be added to a given graph using the functions `xlabel`, `ylabel`, `title`, `grid` and `text`. These functions have the following form:

`title('title')` displays title at top of graph.

`xlabel('x_axis_name')` displays name chosen for `x_axis`.

`ylabel('y_axis_name')` displays name chosen for `y_axis`.

`grid` superimposes a grid on the graph.

`text(x,y,'text-at-x,y')` displays `text-at-x,y` at position `(x,y)` in the graph window where `x` and `y` are measured in the units of the current plotting axes. There may be one point or many at which text is placed depending on whether or not `x` and `y` are vectors.

`gtext('text')` allows the placement of text using the mouse by positioning it where the text is required and then pressing the button.

In addition, the function `axis` allows the user to set the limits of the axes for a particular plot. This takes the form `axis(p)` where `p` is a four-element row vector specifying the lower and upper limits of the axes in the `x` and `y` directions. The axis statement must be placed after the `plot` statement to which it refers. Note that the functions `title`, `xlabel`, `ylabel`, `grid`, `text`, `gtext` and `axis` must follow the plot to which they refer.

The following script gives a plot which is output as Fig. 1.12.1. The function `hold` is used to ensure that the two graphs are superimposed.

```
x=-4:0.05:4; y=exp(-0.5*x).*sin(5*x);
figure(1);
plot(x,y);
xlabel('x-axis'); ylabel('y-axis');
hold on;
y=exp(-0.5*x).*cos(5*x);
plot(x,y);
grid; gtext('Two tails...');
hold off
```

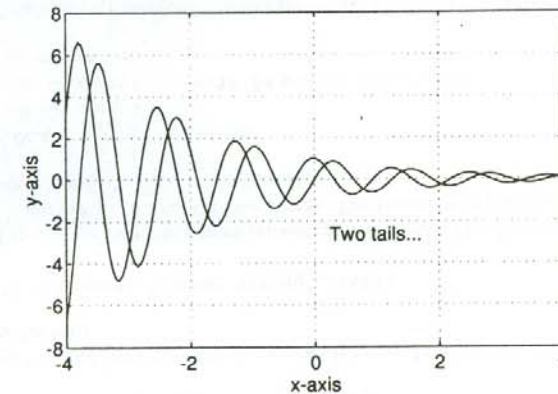


Fig. 1.12.1. Superimposed graphs obtained using `plot(x,y)` and `hold` statements.

The function `fplot` allows the user to plot a previously defined function between given limits. The important difference between `fplot` and `plot` is that `fplot` chooses the plotting points in the given range adaptively. Thus more points are chosen when the function is changing more rapidly. This is illustrated by executing the following MATLAB script:

```

x=2:.04:4;
y=f101(x);
plot(x,y);
xlabel('x'); ylabel('y');
figure(2)
fplot('f101',[2 4],10)
xlabel('x'); ylabel('y');

```

The function f101 is given by

```

function v=f101(x)
v=sin(x.^3);

```

Running the above script produces Fig. 1.12.2 and Fig. 1.12.3. In this example we have deliberately chosen an inadequate number of plotting points but even so function fplot has produced a smoother and more accurate curve.

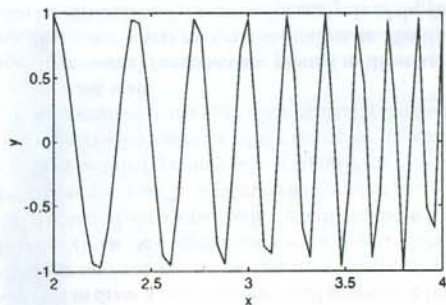


Fig. 1.12.2. Plot of $y = \sin(x^3)$ using 75 equispaced plotting points.

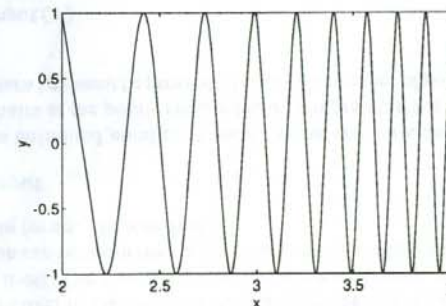


Fig. 1.12.3. Plot of $y = \sin(x^3)$ using the function fplot to choose plotting points adaptively.

There are a number of special features available in MATLAB for the presentation and manipulation of graphs and some of these will now be discussed. The subplot function takes the form subplot(p,q,r) where p,q splits the figure window into a p by q grid of cells and places the plot in the rth cell of the grid, numbered consecutively along the rows. This is illustrated by running the following script which generates six different plots, one in each of the six cells. These plots are given in Fig. 1.12.4.

```

x=0.1:1:5;
subplot(2,3,1);plot(x,x);title('plot of x');
xlabel('x'); ylabel('y');
subplot(2,3,2);plot(x,x.^2);title('plot of x^2');
xlabel('x'); ylabel('y');
subplot(2,3,3);plot(x,x.^3);title('plot of x^3');
xlabel('x'); ylabel('y');
subplot(2,3,4);plot(x,cos(x));title('plot of cos(x)');
xlabel('x'); ylabel('y');
subplot(2,3,5);plot(x,cos(2*x));title('plot of cos(2x)');
xlabel('x'); ylabel('y');
subplot(2,3,6);plot(x,cos(3*x));title('plot of cos(3x)');
xlabel('x'); ylabel('y');

```

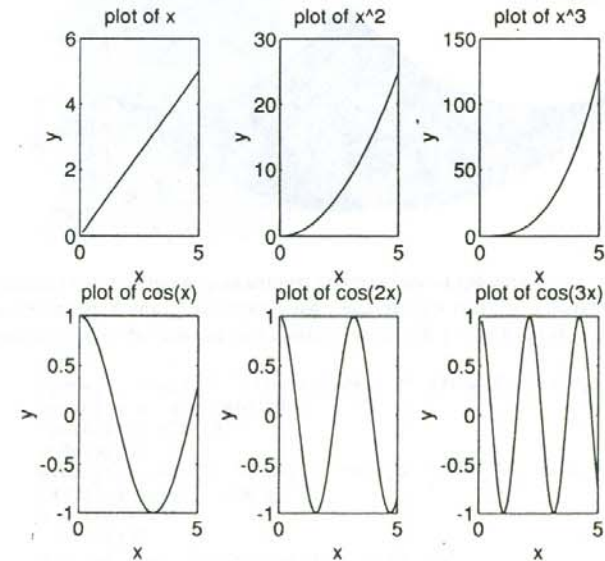


Fig. 1.12.4. Example of the use of subplot.

The current plot can be held on screen by using the function `hold` and subsequent plots are drawn over it. The function `hold on` switches the hold facility on while `hold off` switches it off. The figure window can be cleared using the function `clf`.

Information can be taken from a graphics window using the function `ginput` which takes two main forms. The simplest is

```
[x,y]=ginput
```

This inputs an unlimited number of points to the vectors `x` and `y` by positioning the mouse cross-hairs at the points required and then pressing the mouse button. To exit `ginput` the return key must be pressed. If a specific number of points `n` are required then we write

```
[x,y]=ginput(n)
```

1.13 THREE-DIMENSIONAL GRAPHICS

It is often convenient to draw a three-dimensional graph of a function or set of data to gain a deeper insight into the nature of that data. MATLAB provides powerful and extensive facilities to allow the user to draw a wide range of three-dimensional graphs. Here we only briefly introduce a small selection of these functions. These are the functions `meshgrid`, `mesh`, `surf`, `surf1`, `contour` and `contour3`. It should be noted that the more complex graphs of this type may take a significant time to draw on the screen, depending on the algebraic complexity of the function, the amount of detail required and the power of the computer being used. Consequently the user should sometimes be prepared for a significant wait.

Usually three-dimensional functions are plotted to illustrate particular features of the function such as regions where maxima or minima lie. Plotting surfaces to illustrate these features can be difficult and some careful analysis of the function may be needed before the graph is drawn successfully. In addition, even when the region of interest is successfully located and plotted the feature of interest may be hidden and it will then be necessary to choose a different viewpoint. Discontinuities may also be present and cause plotting problems.

For the function $z = f(x, y)$ the MATLAB function `meshgrid` is used to generate a complete set of points in the x - y plane for the three-dimensional plotting functions. We can then compute the values of z and these are finally plotted by using one of the functions `mesh`, `surf`, `surf1` or `surfz`. For example, to plot the function

$$z = (-20x^2 + x)/2 + (-15y^2 + 5y)/2 \text{ for } x = -4:0.2:4 \text{ and } y = -4:0.2:4$$

we first set up the values of the x - y domain and then compute z corresponding to these x and y values using the given function. Finally we plot the three-dimensional graph using the function `surf1`. This is achieved by using the following script. Note how the function `figure` is used to direct the output to a graphics window so that the first plot is not over-written by the second.

```
clf
[x,y]=meshgrid(-4.0:0.2:4.0,-4.0:0.2:4.0);
z=-0.5*(-20*x.^2+x)+0.5*(-15*y.^2+5.*y);
figure(1);
surf1(x,y,z);
axis([-4 4 -4 4 -400 0])
xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis');
figure(2);
contour3(x,y,z,15);
axis([-4 4 -4 4 -400 0])
xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis');
```

Running this script generates the plots shown in Fig. 1.13.1 and Fig. 1.13.2. The former is created using `surf1` and shows the function as a surface. The latter is created by `contour3` and is a three-dimensional contour plot of the surface.

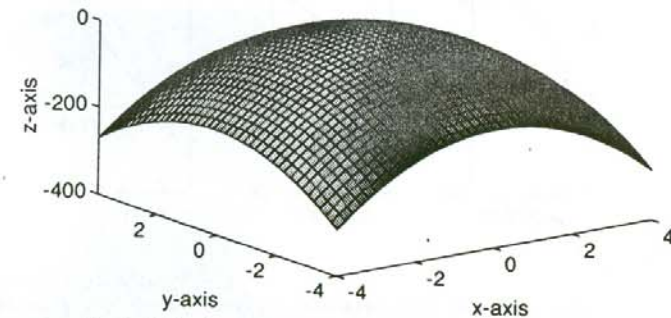


Fig. 1.13.1. Three-dimensional surface using default view.

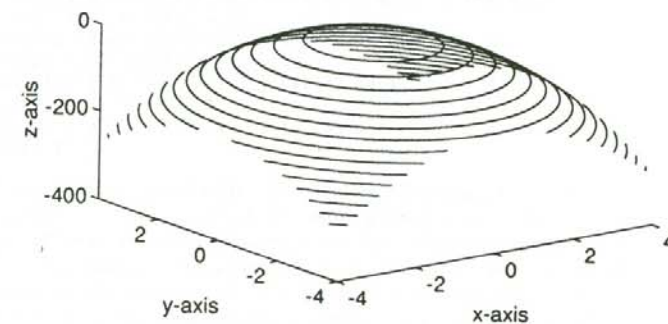


Fig. 1.13.2. Three-dimensional contour plot.

When plotting surfaces a very useful function is `view`. This function allows the surface or mesh to be viewed from different positions. The function has the form `view(az, el)` where `az` is the azimuth and `el` the elevation of the viewpoint required. Azimuth may be interpreted as the viewpoint rotation about the z -axis and elevation as the rotation of the viewpoint about the x - y plane. A positive value of the elevation gives a view from above the object and a negative value a view from below. Similarly a positive value of azimuth gives a counter-clockwise rotation of the viewpoint about the z -axis while a negative value gives a clockwise rotation. If the `view` function is not used the default values are -37.5° for the azimuth and 30° for the elevation.

There are many other plotting facilities but they are not described here.

1.14 SCRIPTING IN MATLAB

In some of the previous sections we have created some simple MATLAB scripts that have allowed a series of commands to be executed sequentially. However, many of the features usually found in programming languages are also provided in MATLAB to allow the user to create versatile scripts. The more important of these features will be described in this section. It must be noted that scripting is done in the `edit` window using a text editor appropriate to the system, not in the `command` window which only allows the execution of statements one at a time or several statements provided that they are on the same line.

MATLAB *does not require the declaration of variable types* but for the sake of clarity the role and nature of key variables may be indicated by using comments. Any text following the symbol `%` is considered as comment. In addition there are certain variable names which have predefined special values for the convenience of the user. They can, however, be redefined if required. These are

<code>eps</code>	which is set to the particular machine accuracy
<code>pi</code>	which equals π
<code>inf</code>	the result of dividing by zero
<code>NaN</code>	which is not a number produced on dividing zero by zero
<code>i, j</code>	both of which equal $\sqrt{-1}$.

Assignment statements in a MATLAB script take the form

`variable=expression;`

The expression is calculated and the value assigned to the variable on the left-hand side. If the semicolon is omitted from the end of these statements the name of the variable(s) and the assigned value(s) is displayed on the screen. A further form is not to assign the expression explicitly. In this case the value of the expression is calculated, assigned to the variable `ans` and displayed.

A variable in MATLAB is assumed to be a matrix of some kind; its name must start with a letter and may be followed by any combination of digits and letters; a maximum of 19 characters is recognised. It is good practice to use a meaningful variable name.

However, it is very important to avoid the use of existing MATLAB commands, function names or even the word MATLAB itself! MATLAB does not prevent their use but using them can lead to problems and inconsistencies. The expression is a valid combination of variables, constants, operators and functions. Brackets can be used as convenient to alter or clarify the precedence of operations. The precedence of operation for simple operators is first \wedge , second $*$, third $/$ and finally $+$ and $-$ where

\wedge	raises to a power
$*$	multiplies
$/$	divides
$+$	adds
$-$	subtracts

The effects of these operators in MATLAB have already been discussed.

Unless there are instructions to the contrary, a set of MATLAB statements in a script will be executed in sequence. This is the case in the following example.

```
% Matrix calculations for two matrices A and B
A=[1 2 3; 4 5 6; 7 8 9];
B=[5 -6 -9; 1 1 0; 24 1 0];
% Addition result assigned to C
C=A+B;
disp(C);
% Multiplication result assigned to D
D=A*B;
disp(D);
% Division result assigned to E
E=A\B;
disp(E);
```

In order to allow the repeated execution of one or more statements a `for` loop is used. This takes the form

```
for loopvariable = loopexpression
    Statements
end
```

The `loopvariable` is a suitably named variable and `loopexpression` is usually of the form `n:m` or `m:i:n` where `n`, `i` and `m` are the initial, incremental and final values of `loopvariable`. They may be constants, variables or expressions; they can be negative or positive but clearly they should be chosen to take values consistent with the logic of the script. This structure should be used when the loop is to be repeated a predetermined number of times.

Example:

```
for i=1:n
    for j=1:m
        C(i,j)=A(i,j)+cos((i+j)*pi/(n+m))*B(i,j);
    end
end

for k=n+2:-1:n/2
    a(k)=sin(pi*k);
    b(k)=cos(pi*k);
end
```

When the repetition is continued subject to a condition being satisfied which may be dependent on values generated within the loop the while statement is used. This has the form

```
while while_expression
    statements
end
```

The while_expression is a *relational expression* of the form $e1 \cdot e2$ where $e1$ and $e2$ are ordinary arithmetic expressions as described before and \cdot is a relational operator defined as follows:

```
== equal
<= less than or equal
>= greater than or equal
~= not equal
< less than
> greater than
```

Relational expressions may be combined using the following *logical operators*:

```
& provides the and operator;
| provides the or operator;
~ provides the not operator.
```

Note that false is 0 and true is non-zero. Relational operators have a higher order of precedence than logical operators.

Examples of while loops:

```
dif=1;
while dif>0.0005
    x1=x2-cos(x2)/(1+x2);
    dif=abs(x2-x1);
    x2=x1;
end
```

```
while sum(x) ~= max(y)
    x=x.^2;
    y=y+x;
end
```

Note also that break allows exit from while or for loops.

A vital feature of all programming languages is the ability to change the sequence in which instructions are executed, subject to some condition being satisfied that may be dependent on values generated within the program. In MATLAB the if statement is used to achieve this and has the general form

```
if if_expression
    statements
elseif if_expression
    statements
elseif if_expression
    statements
...
else
    statements
end
```

Here the if_expression is a relational expression of the form $e1 \cdot e2$ where $e1$ and $e2$ are ordinary arithmetic expressions and \cdot is a relational operator as described before. Relational expressions may be combined using logical operators.

Examples:

```
for k=1:n
    for p=1:m
        if k==p
            z(k,p)=1;
            total=total+z(k,p);
        elseif k<p
            z(k,p)=-1;
            total=total+z(k,p);
        else
            z(k,p)=0;
        end
    end
end

if (x~=0)&(x<y)
    b=sqrt(y-x)/x;
    disp(b);
end
```

It is important to note that if any statement in a MATLAB script is longer than one line then it must be continued by using an ellipsis (...) at the end of a line to denote a

continuation onto the next line.

1.15 FUNCTIONS IN MATLAB

A very large number of functions are directly built into MATLAB. They may be called by using the function name together with the parameters that define the function. These functions may return one or more values. A small selection of MATLAB functions are given in the following table which gives the function name, the function use and an example function call. Note that all function names must be in lower case letters.

Function	Function gives	Example
<code>sqrt(x)</code>	square root of x	<code>y=sqrt(x+2.5);</code>
<code>abs(x)</code>	positive value of x	<code>d=abs(x)*y;</code>
<code>conj(x)</code>	the complex conjugate of x	<code>x=conj(y);</code>
<code>sin(x)</code>	sine of x	<code>t=x+sin(x);</code>
<code>log(x)</code>	log to base e of x	<code>z=log(1+x);</code>
<code>log10(x)</code>	log to base 10 of x	<code>z=log10(1-2*x);</code>
<code>cosh(x)</code>	hyperbolic cosine of x	<code>u=cosh(pi*x);</code>
<code>exp(x)</code>	exponential of x , i.e. e^x	<code>p=.7*exp(x);</code>
<code>gamma(x)</code>	gamma function of x	<code>f=gamma(y);</code>
<code>expm(A)</code>	exponential of matrix A	<code>p=expm(A+B);</code>
<code>sqrtm(A)</code>	square root of matrix A	<code>y=sqrtm(A)</code>

In the case of the functions `expm(A)` and `sqrtm(A)` the matrix is treated as a whole. For example, `B=sqrtm(A)` assigns the square root of the matrix A to B so that the matrix B multiplied by itself is equal to A . In contrast, `C=sqrt(A)` takes the square root of each element of A and assigns them to C .

Some functions perform special calculations for an important and general mathematical process. These functions often require more than one input parameter and may provide several outputs. For example, `bessel(n,x)` gives the n th order Bessel function of x . The statement `y=fzero('fun',x0)` determines the root of the function `fun` near to `x0` where `fun` is a function given by the user that defines the equation for which we are finding the root. The statement `[Y,I]=sort(X)` is an example of a function that can return two output values. Y is the sorted matrix and I is a matrix containing the indices of the sort.

In addition to a large number of mathematical functions, MATLAB provides several useful utility functions that may be used for examining the operation of scripts. These are:

<code>clock</code>	Returns the current date and time in the form: [year month day hour min sec].
<code>etime(t2,t1)</code>	Calculates elapsed time between <code>t1</code> and <code>t2</code> . This is very useful for timing segments of a script.

<code>tic,toc</code>	An alternative way of finding the time taken to execute a segment of script. The statement <code>tic</code> starts the timing and <code>toc</code> gives the elapsed time since the last <code>tic</code> .
<code>cputime</code>	Returns the total time in seconds since MATLAB was launched.
<code>flops</code>	Allows the user to check the number of floating point operations, that is the total number of additions, subtractions, multiplications and divisions in a section of script. Executing <code>flops(0)</code> starts the count of flops at zero; <code>flops</code> counts the number of flops cumulatively.
<code>pause</code>	Causes the execution of the script to pause until the user presses a key. Note that the cursor is turned into the symbol P, warning the script is in pause mode. This is often used when the script is operating with <code>echo on</code> .
<code>echo on</code>	Displays each line of script in the command window before execution. This is useful for demonstrations. To turn it off use the statement <code>echo off</code> .

The following script uses the timing functions described above to estimate the time taken to solve a 100 x 100 system of linear equations:

```
%Solve 100x100 system
A=rand(100);b=rand(100,1);
Tbefore=clock; tic; t0=cputime; flops(0);y=A\b;
timetaken=etime(clock,Tbefore); tend=toc; t1=cputime-t0;
disp('etime    tic-toc    cputime ')
fprintf('%5.2f%10.2f%10.2f\n\n', timetaken,tend,t1);
count=flops;
fprintf('flops = %6.0f\n',count);
```

Running this script gives

```
»etime    tic-toc    cputime
  6.92      6.95      7.00

flops = 732005
```

The output shows that the three alternative methods of timing give essentially the same value. The flops are also counted and displayed.

1.16 USER-DEFINED FUNCTIONS

In our description of functions we have mentioned user-defined functions. MATLAB allows users to define their own functions but a specific form of definition must be followed. This is very simple and may be described as follows:

```
function output_params = function_name(input_params)
function body
```

Once the function is defined it *must be saved as an M-file* so that it can then be used. It is good practice to put some comment describing the nature of the function after the function heading. This comment may then be accessed via the help command or menu.

The function call takes the form

```
specific_output_params = function_name(specific_input_params);
```

where `specific_output_params` is either one parameter name or, if there is more than one output parameter, a list of these parameters placed in square brackets, separated by commas. The `specific_input_params` term is either a single parameter or a list of parameters separated by commas. The function body consists of the statements defining the user's function. These statements *should include statements assigning values to the output parameters*. For example, suppose we wish to define the function

$$\left(\frac{x}{2.4}\right)^3 - \frac{2x}{2.4} + \cos\left(\frac{\pi x}{2.4}\right)$$

The MATLAB function definition will be as follows:

```
function p=fun1(x)
% A simple function definition
x=x/2.4;
p=x^3-2*x+cos(pi*x);
```

A call of this function is `y=fun1(2.5)`; or `z=fun1(x-2.7*y)`; assuming `x` and `y` have preset values. Another way of calling the function is as an input parameter to another function. For example:

```
solution=fzero('fun1',2.9);
```

This will give the zero of `fun1` closest to 2.9.

The following example illustrates the definition of a function with more than one input and one output parameter:

```
function [x1,x2]=rootquad(a,b,c)
% This function solves a simple quadratic
% equation ax^2+bx+c=0 given the
% coefficients a,b,c. The solutions are
% assigned to x1 and x2
d=b*b-4*a*c;
x1=(-b+sqrt(d))/(2*a);
x2=(-b-sqrt(d))/(2*a);
```

This function solves quadratic equations and, assuming `x`, `y` and `z` have preset values, valid function calls are

```
[r1,r2]=rootquad(4.2, 6.1, 4);
or
[p,q]=rootquad(x+2*y,x-y,2*z);
```

or using the function `feval`

```
[r1,r2]=feval('rootquad',1,2,3);
```

A more important application of `feval`, which is widely used in this text, is in the process of defining functions which themselves have functions as parameters. These functions can be evaluated internally in the body of the calling function by using `feval`.

1.17 SOME PITFALLS IN MATLAB

We now list five important points which if observed will enable the MATLAB user to avoid some significant difficulties. This list is not exhaustive.

- It is important to take care when naming files and functions. File and function names follow the rules for variable names, i.e. they must start with a letter followed by a combination of letters or digits and names of existing functions must not be used.
- Do not use MATLAB function names or commands for variable names.
- Matrix sizes are set by assignment so it is vital to ensure that matrix sizes are compatible. Often it is a good idea initially to assign a matrix to an appropriate sized zero matrix; this also makes execution more efficient. For example, consider the following simple script:

```
for i=1:2
    b(i)=i*i;
end
a=[4 5; 6 7];
a*b'
```

We assign two elements to `b` in the `for` loop and make `a` a 2×2 array so that we would expect this script to succeed. However, if `b` had in the same session been previously set to be a different size matrix then this script will fail. To ensure that it works correctly we must either assign `b` to be a null matrix using `b = []`, or make `b` a column vector of two elements by using `b = zeros(2,1)` or by using the `clear` statement to clear all variables from the system.

- To create a matrix of zeros that is the same size as a given matrix `P` the recommended form in MATLAB version 4 is `B=zeros(size(P))`. To create a $P \times P$ matrix, where `P` is a scalar, the user writes `B=zeros(P)`. Earlier versions of MATLAB allowed the user to write an ambiguous statement of the form `B=zeros(P)` where `P` could be a scalar or a matrix.

- Take care with dot products. For example, when defining functions where any of the variables may be vectors then dot products must be used. Note also that $2.^x$ and $2.^x$ are different because the space is important. The first example gives the dot power whilst the second gives 2.0 to the power x , not the dot power. Similar care with spaces must be taken when using complex numbers. For example, $a=[1\ 2-i*4]$ assigns two elements: 1 and the complex number $2-i4$. In contrast $b=[1\ 2\ -i*4]$ assigns three elements: 1, 2 and the imaginary number $-i4$.

1.18 SPEEDING UP CALCULATIONS IN MATLAB

Calculations can be greatly speeded up by using vector operations rather than using a loop to repeat a calculation. To illustrate this consider the following simple examples:

Example 1. This script fills the vector **b** using a for loop.

```
% Fill b with square roots of 1 to 1000 using a for loop
clear;
tic;
for i =1:1000
    b(i)=sqrt(i);
end
t=toc;
disp(['Time taken for loop method is ', num2str(t)]);
```

Example 2. This script fills the vector **b** using a vector operation.

```
% Fill b with square roots of 1 to 1000 using a vector
clear;
tic;
a=1:1:1000;
b=sqrt(a);
t=toc;
disp(['Time taken for vector method is ', num2str(t)]);
```

The times taken (in seconds) using various computers were as follows:

	by loop	by vector	ratio
Macintosh LC II	10.92	0.333	33
Macintosh Quadra 610	1.82	0.083	22
PC 486-66MHz	0.74	0.006	123
Sun Sparc LX	0.81	0.005	162

These results illustrate the need to think very carefully about the way algorithms are implemented in MATLAB, particularly with regard to the use of vectors and arrays.

PROBLEMS

- 1.1. (a) Start up MATLAB. In the **command** window type $x=-1:0.1:1$ and then execute each of the following statements by typing them in and pressing return:

```
sqrt(x)          cos(x)
sin(x)           x.^2
x.^3            plot(x, sin(x.^3))
plot(x, cos(x.^4))
```

Examine the effects of each statement carefully.

- (b) Execute the following and explain the results:

```
x=[2 3 4 5]
y=-1:1:2
x.^y
x.*y
x./y
```

- 1.2. (a) Set up the matrix $A=[1\ 5\ 8; 84\ 81\ 7; 12\ 34\ 71]$ in the **command** window and examine the contents of $A(1,1)$, $A(2,1)$, $A(1,2)$, $A(3,3)$, $A(1:2,:)$, $A(:,1)$, $A(3,:)$, $A(:,2:3)$.

- (b) What do the following MATLAB statements produce?

```
x=1:1:10
z=rand(10)
y=[z;x]
c=rand(4)
e=[c eye(size(c)); eye(size(c)) ones(size(c))]
d=sqrt(c)
t1=d*d
t2=d.*d
```

- 1.3. Set up a 4×4 matrix. Given that the function $\text{sum}(x)$ will give the sum of the elements of the vector x , use the function sum to find the sums of the first row and second column of the matrix.

- 1.4. Solve the following system of equations using the MATLAB function inv and also using the operators \backslash and $/$ in the **command** window:

$$\begin{aligned} 2x + y + 5z &= 5 \\ 2x + 2y + 3z &= 7 \\ x + 3y + 3z &= 6 \end{aligned}$$

Verify the solution is correct using matrix multiplication.

- 1.5. Write a simple script to input two square matrices **A** and **B**: then add, subtract and multiply them. Comment the script and use `disp` to output suitable titles.
- 1.6. Write a MATLAB script to set up a 4 x 4 random matrix **A** and a four-element column vector **b**. Calculate $\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$ and display the result. Calculate $\mathbf{A} * \mathbf{x}$ and compare it with **b**.
- 1.7. Write a simple script to plot the two functions $y_1 = x^2 \cos x$ and $y_2 = x^2 \sin x$ on the same graph. Use comments in your script and take $x = -2:0.1:2$.
- 1.8. Write a MATLAB script to produce graphs of the functions $y = \cos x$ and $y = \cos(x^3)$ in the range $x = -4:0.02:4$ using the same axes. Use the MATLAB functions `xlabel`, `ylabel` and `title` to annotate your graphs clearly.
- 1.9. Draw the function $y = \exp(-x^2) \cos(20x)$ in the range $x = -2:0.1:2$. All axes should be labelled and a title included. Compare the results of using the functions `fplot` and `plot` to plot this function.
- 1.10. Write a MATLAB script to draw the functions $y = 3 \sin(\pi x)$ and $y = \exp(-0.2x)$ on the same graph for $x = 0:0.02:4$. All axes should be labelled. Use `gtext` to label one of the several point of intersection of the graphs.
- 1.11. Use the functions `mesh` and `meshgrid` to obtain a three-dimensional plot of the function

$$z = 2xy/(x^2 + y^2) \text{ for } x = 1:0.1:3 \text{ and } y = 1:0.1:3$$

Redraw the surface using the function `surf` and `contour`.

- 1.12. An iterative equation for solving the equation $x^2 - x - 1 = 0$ is given by

$$x_{r+1} = 1 + (1/x_r) \text{ for } r = 0, 1, 2, \dots$$

Given x_0 is 2 write a MATLAB script to solve the equation. Sufficient accuracy is obtained when $\text{abs}(x_{r+1} - x_r) < 0.0005$. Include a check on the answer.

- 1.13. Given a 4 x 5 matrix **A**, write a script to find the sums of each of the columns using
1. The `for` statement.
 2. The function `sum`.

- 1.14. Given a vector **x** with n elements, write a MATLAB script to form the products

$$p_k = x_1 x_2 \dots x_{k-1} x_{k+1} \dots x_n$$

for $k = 1, 2, \dots, n$. That is, p_k will contain the products of all the vector elements except the k th. Run your script with specific values of x and n .

- 1.15. The series for $\log_e(1+x)$ is given by

$$\log_e(1+x) = x - x^2/2 + x^3/3 - \dots + (-1)^{k+1} x^k/k \dots$$

Write a MATLAB script to input a value for x and sum the series while the value of the current term is greater than or equal to the variable `tol`. Use values of $x = 0.5$ and 0.82 and `tol = 0.005` and `0.0005`. The result should be checked by using the MATLAB function `log`. The script should display the value of x and `tol` and the value of $\log_e(1+x)$ obtained. Use `input` and `disp` functions to obtain clear output and prompts.

- 1.16. Write a MATLAB script to generate a matrix which has the values d along the main diagonal and the values c on the diagonals above and below the main diagonal and zero elsewhere. Your script should allow the user to input any values for c and d and work for any size of matrix n . The script should give clear prompts for input and display the results with a suitable heading.

- 1.17. Write a MATLAB function to solve the quadratic equation

$$ax^2 + bx + c = 0$$

The function will use three input parameters a , b , c and output the values of the two roots. You should take account of the three cases:

- (1) no real roots
- (2) real and different roots
- (3) equal roots.

Hint: Develop the function `rootquad` given on page 22.

- 1.18. Adjust the function of problem 1.17 to deal with the case when $a = 0$. That is, when the equation is non-quadratic. In this case include a third output parameter which will have the value 1 if the equation is quadratic and 0 otherwise.
- 1.19. Write a simple function to define $f(x) = x^2 - \cos x - x$ and plot the graph of the function in the range 0 to 2. Use this graph to find an initial approximation to the root and then apply the function `fzero` to find the root to tolerance 0.0005.