

Chapter 2

Basic Operations with MATLAB

This book is concerned with the understanding of algorithms for problems of continuous mathematics. Part of this understanding is the ability to implement such algorithms. To avoid distracting implementation details, however, we would like to accomplish this implementation in the simplest way possible, even if it is not necessarily the most efficient. One system in which algorithm implementation is especially easy is called MATLAB [?] (short for MATrix LABoratory).

While a helpful academic and instructive tool, MATLAB is used in industry and government, as well. For instance, systems engineers at the NASA Jet Propulsion Laboratory used MATLAB to understand system behavior before launching the Mars Exploration Rover (MER) spacecraft into space.

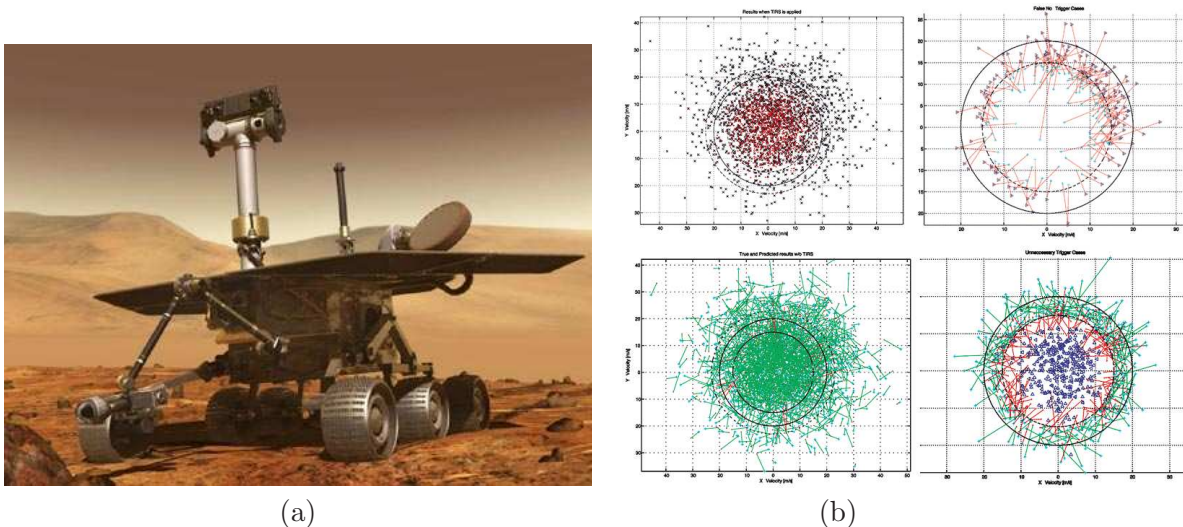


Figure 2.1: (a) An artist’s conception of the Mars rover. (b) Custom statistical MATLAB visualizations that were used to predict how the onboard systems would respond under various atmospheric conditions during descent to the Mars surface.

This chapter contains a short description of basic MATLAB commands, and more commands are described in programs throughout the book. For further information about using MATLAB see, for instance, [?].

2.1 Launching MATLAB

MATLAB is a high-level programming language that is especially well suited to linear algebra computations, but it can be used for almost any numerical problem. Following are some of the basic features of MATLAB that you will need to carry out the programming exercises in this book. Depending on what system you are using, you will start MATLAB either by double clicking on a MATLAB icon or by typing “matlab” or by some similar means. When MATLAB is ready for input from you it will give a prompt such as `>>`.

You can use MATLAB like a calculator. For instance, if you type at the prompt

```
>> 1+2*3
```

then MATLAB returns with the answer

```
ans =  
      7
```

Since you did not give a name to your result, MATLAB stores the result in a variable called **ans**. You can do further arithmetic using the result in **ans**:

```
>> ans/4
```

and MATLAB will return with the result

```
ans =  
    1.7500
```

2.2 Vectors

MATLAB can store row or column vectors. The commands

```
>> v = [1; 2; 3; 4]
```

```
v =  
     1  
     2  
     3  
     4
```

```
>> w = [5, 6, 7, 8]
```

```
w =  
     5     6     7     8
```

create a column vector \mathbf{v} of length 4 and a row vector \mathbf{w} of length 4. In general, when defining a matrix or vector, semicolons are used to separate rows, while commas or spaces are used to separate the entries within a row. You can refer to an entry in a vector by giving its index:

```
>> v(2)
ans =
     2
```

```
>> w(3)
ans =
     7
```

MATLAB can add two vectors of the same dimension, but it cannot add \mathbf{v} and \mathbf{w} because \mathbf{v} is 4 by 1 and \mathbf{w} is 1 by 4. If you try to do this, MATLAB will give an error message:

```
>> v+w
??? Error using ==> +
Matrix dimensions must agree.
```

The transpose of \mathbf{w} is denoted \mathbf{w}' :

```
>> w'
ans =
     5
     6
     7
     8
```

You can add \mathbf{v} and \mathbf{w}' using ordinary vector addition:

```
>> v + w'
ans =
     6
     8
    10
    12
```

Suppose you wish to compute the sum of the entries in \mathbf{v} . One way to do this is as follows:

```
>> v(1) + v(2) + v(3) + v(4)
ans =
    10
```

Another way is to use a `for` loop:

```
>> sumv = 0;
>> for i=1:4, sumv = sumv + v(i); end;
>> sumv
sumv =
    10
```

This code initializes the variable `sumv` to 0. It then loops through each value $i = 1, 2, 3, 4$ and replaces the current value of `sumv` with that value plus `v(i)`. The line with the `for` statement actually contains three separate MATLAB commands. It could have been written in the form

```
for i=1:4
    sumv = sumv + v(i);
end
```

MATLAB allows one line to contain multiple commands, provided they are separated by commas or semicolons. Hence in the one-line version of the `for` loop, we had to put a comma (or a semicolon) after the statement `for i=1:4`. This could have been included in the three-line version as well, but it is not necessary. Note also that in the three-line version, we have *indented* the statement(s) inside the `for` loop. This is not necessary, but it is good programming practice. It makes it easy to see which statements are inside and which are outside the `for` loop. Note that the statement `sumv = 0` is followed by a semicolon, as is the statement `sumv = sumv + v(i)` inside the `for` loop. Following a statement by a semicolon suppresses printing of the result. Had we not put the semicolon at the end of the first statement, MATLAB would have printed out the result `sumv = 0`. Had we not put a semicolon after the statement `sumv = sumv + v(i)`, then each time through the `for` loop, MATLAB would have printed out the current value of `sumv`. In a loop of length 4, this might be acceptable; in a loop of length 4 million, it probably would not be! To see the value of `sumv` at the end, we simply type `sumv` without a semicolon and MATLAB prints out its value. Of course, if the answer is not what we were expecting, then we might go back and omit the semicolon after the statement `sumv = sumv + v(i)`, since then we could see the result after each step. Extra output is often useful as a program debugging tool.

2.3 Getting help

Actually, the entries in a vector are most easily summed using a built-in MATLAB function called `sum`. If you are unsure of how to use a MATLAB function or command, you can always type `help` followed by the command name, and MATLAB will provide an explanation of how the command works:

```
>> help sum
SUM Sum of elements.
    For vectors, SUM(X) is the sum of the elements of X. For
    matrices, SUM(X) is a row vector with the sum over each
    column. For N-D arrays, SUM(X) operates along the first
    non-singleton dimension.
```

`SUM(X,DIM)` sums along the dimension `DIM`.

Example: If $X = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$

then `sum(X,1)` is `[3 5 7]` and `sum(X,2)` is `[3
12];`

See also `PROD`, `CUMSUM`, `DIFF`.

In general, you can type `help` in MATLAB and receive a summary of the classes of commands for which help is available. If you are not sure of the command name for which you are looking, there are two other helpful commands. First, typing `helpdesk` displays the help browser, which is a very *helpful* tool. Additionally, you can type `doc` for the same help browser, or `doc sum` for the hypertext documentation on the MATLAB `sum` command. Second, if you are interested in commands related to summing, you can type `lookfor sum`. With this command, MATLAB searches for the specified keyword “sum” in all help entries. This command results in the following response:

```
>> lookfor sum
TRACE Sum of diagonal elements.
CUMSUM Cumulative sum of elements.
SUM Sum of elements.
SUMMER Shades of green and yellow colormap.
UIRESUME Resume execution of blocked M-file.
UIWAIT Block execution and wait for resume.
RESUME Resumes paused playback.
RESUME Resumes paused recording.
```

It may take some searching to find precisely the topic and/or command that you are looking for.

2.4 Matrices

MATLAB also works with matrices:

```
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 0]
A =
     1     2     3
     4     5     6
     7     8     0
```

```
>> b = [0; 1; 2]
b =
     0
```

```
1
2
```

There are many built-in functions for solving matrix problems. For example, to solve the linear system $Ax = b$, type `A\b`:

```
>> x = A\b
x =
    0.6667
   -0.3333
    0.0000
```

Note that the solution is printed out to only four decimal places. It is actually stored to about sixteen decimal places (see Chapter 5). To see more decimal places, you can type

```
>> format long
>> x
x =
    0.666666666666667
   -0.333333333333333
    0.000000000000000
```

Other options include `format short e` and `format long e` to display numbers using scientific notation.

You can check this answer by typing `b - A*x`. The notation `A*x` denotes standard matrix-vector multiplication, and standard vector subtraction is used when you subtract the result from `b`. This should give a vector of 0s, if `x` solves the system exactly. Since the machine carries only about 16 decimal digits, we do not expect it to be exactly zero, but, as we will see later, it should be just a moderate size multiple of 10^{-16} :

```
>> format short
>> b - A*x
ans =
    1.0e-15 *

   -0.0740
   -0.2220
         0
```

This is a good result!

2.5 Creating and running .m files

Typing MATLAB commands at the keyboard is fine if you are doing a computation once and will never need to make modifications and run it again. Once you `exit` MATLAB, however, all of the

commands that you typed may be lost. To save the MATLAB commands that you type so that they can be executed again, you must enter them into a file called *filename.m*. Then, in MATLAB, if you type *filename*, it will run the commands from that file. We will refer to such files as M-files. The M-file can be produced using any text editor, such as the one that comes up as part of the MATLAB window. Once you save this file, it will be available for future use. Before attempting to execute an M-file from MATLAB, you must remember to change the working directory of MATLAB to the directory in which that file resides.

2.6 Comments

Adding documentation to your MATLAB code allows you and others to maintain your code for future use. Many a programmer has coded what appears to be a crystal clear implementation of an algorithm and later returned to be lost in the listing of commands. Comments can help to alleviate this problem. Adding comments to MATLAB code is easy. Simply adding a % makes the remaining portion of that line a comment. In this way, you can make an entire line into a comment:

```
% Solve Ax=b
```

or you can append a comment after a MATLAB command, as in

```
x = A\b; % This solves the linear system Ax=b and stores the result in x.
```

The text following the % is simply a comment for the programmer and is ignored by MATLAB.

2.7 Plotting

Tables and figures are usually more helpful than long strings of numerical results. Suppose you are interested in viewing a plot of $\cos(50x)$ for $0 \leq x \leq 1$. You can create two vectors, one consisting of x values and the other consisting of the corresponding $y = \cos(50x)$ values, and use the MATLAB `plot` command. To plot the values of $\cos(50x)$ at $x = 0, 0.1, 0.2, \dots, 1$, type

```
>> x = 0:0.1:1; % Form the (row) vector of x values.
>> y = cos(50*x); % Evaluate cos(50*x) at each of the x values.
>> plot(x,y) % Plot the result.
```

Note that the statement `x = 0:0.1:1;` behaves just like the `for` loop

```
>> for i=1:11, x(i) = 0.1*(i-1); end;
```

Note also that, unless otherwise specified, each of these statements produces a *row* vector. In order to produce a column vector, one could replace `x(i)` in the above `for` loop by `x(i,1)`, or one could replace the statement in the original code by `x = [0:0.1:1]'`;. Clearly, this small number of evaluation points will not produce very high resolution, as is seen in Figure 2.2(a).

In the next piece of code we change `x` to include more points and we also include a title and labels for the axes.

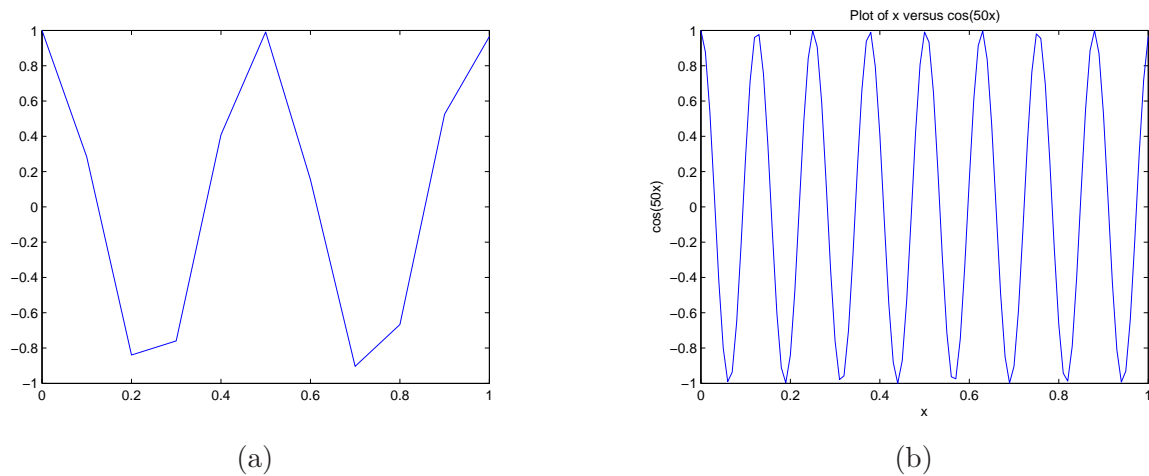


Figure 2.2: Basic MATLAB plots of cosine.

```
>> x = 0:0.01:1;      % Create a vector of 101 x values.
>> plot(x,cos(50*x)) % Plot x versus cos(50*x).
>> title('Plot of x versus cos(50x)')
>> ylabel('cos(50x)')
>> xlabel('x')
```

Figure 2.2(b) contains the plot resulting from these commands.

It is also possible to plot more than one function on the same graph. To plot the two functions $f(x) = \cos(50x)$ and $g(x) = x$ on the same graph, type

```
>> plot(x,cos(50*x),x,x)
```

The result is shown in Figure 2.3. Note the small boxed legend on the plot. This was added to the plot with the command

```
legend('cos(50x)', 'x')
```

Another way to plot two functions on the same graph is to first plot one, then type **hold on**, and then plot the other. If you do not type **hold on**, then the second plot will replace the first, but this command tells MATLAB to keep plots on the screen after they are created. To go back to the default of removing old plots before new ones are added, type **hold off**. Again, type **help plot** for more information on plotting or type **doc plot** which posts the Helpdesk documentation for the **plot** command. Other useful commands are **axis** and **plot3**. For a bit of fun, type the commands

```
>> x = 0:0.001:10;
>> comet(x,cos(3*x))
```

For more information on options available for plots, type the commands **hndlggraf**, **hndlaxis** and **ardemo**.

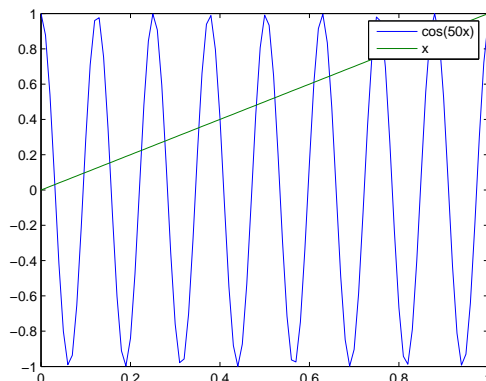


Figure 2.3: A basic MATLAB plot of cosine and a line along with a legend.

2.8 Creating Your Own Functions

You can create your own functions to use in MATLAB. If your function is simple (e.g., $f(x) = x^2 + 2x$), then you may enter it using the command `inline`:

```
>> f = inline('x.^2 + 2*x')
f =
    Inline function:
    f(x) = x.^2 + 2*x
```

Note the `.^2` notation. The expression `x^2` produces the square of `x` if `x` is a scalar, but it gives an error message if `x` is a vector, since standard vector multiplication is defined only if the inner dimensions of the vectors are the same (i.e., the first vector is 1 by n and the second is n by 1 or the first is n by 1 and the second is 1 by m). The operation `.^` applied to a vector, however, squares each entry individually. Since we may wish to evaluate the function at each entry in a vector of `x` values, we must use the `.^` operation. To evaluate `f` at the integers between 0 and 5, type

```
>> f([0:5])
ans =
     0     3     8    15    24    35
```

Similarly, you can create an anonymous function by typing

```
>> f = @(x)(x.^2 + 2*x)
f =
    @(x)(x.^2+2*x)
```

If the function is more complicated, you may create a file whose name ends in `.m` which tells MATLAB how to compute the function. Type `help function` to see the format of the function file. Our function here could be computed using the following file (called `f.m`):

```
function output = f(x)
output = x.^2 + 2*x;
```

This function is called from MATLAB in the same way as above; that is, `f(x)`, where `x` can be a scalar or vector.

2.9 Printing

While graphical output can be an important visualization tool, numerical results are often presented in tables. There are several ways to print output to the screen in MATLAB. First, to simply display a variable's contents, use the command `display`.

```
>> x = 0:.5:2;
>> display(x)
x =
      0      0.5000      1.0000      1.5000      2.0000
```

In many cases, the extra carriage return imposed by the `display` command clutters printed results. Therefore, another helpful command is `disp`, which is similar to the `display` command.

```
>> disp(x)
      0      0.5000      1.0000      1.5000      2.0000
```

You may still wish to have the variable name printed. Concatenating an array of text for output accomplishes this purpose.

```
>> disp(['x = ', num2str(x)])
x = 0      0.5      1      1.5      2
```

For more information, type `help num2str`.

Tables such as the following can be created using `disp`.

```
>> disp('      Score 1      Score 2      Score 3'), disp(rand(5,3))
      Score 1      Score 2      Score 3
      0.4514      0.3840      0.6085
      0.0439      0.6831      0.0158
      0.0272      0.0928      0.0164
      0.3127      0.0353      0.1901
      0.0129      0.6124      0.5869
```

You may find the `fprintf` command easier to use for tables of results. For instance, consider the simple loop

```
>> fprintf('      x      sqrt(x)\n=====\\n')
for i=1:5, fprintf('%f      %f\\n',i,sqrt(i)), end
      x      sqrt(x)
```

```
=====
1.000000    1.000000
2.000000    1.414214
3.000000    1.732051
4.000000    2.000000
5.000000    2.236068
```

The `fprintf` command takes format specifiers and variables to be printed in those formats. The `%f` format indicates that a number will be printed in fixed point format in that location of the line, and the `\n` forces a carriage return after the two quantities `i` and `sqrt(i)` are printed. You can specify the total field width and the number of places to be printed after the decimal point by replacing `%f` by, say, `%8.4f` to indicate that the entire number is to be printed in 8 spaces, with 4 places printed after the decimal point. You can send your output to a file instead of the screen by typing

```
fid = fopen('sqrt.txt','w');
fprintf(fid,'    x      sqrt(x)\n===== \n');
for i=1:5, fprintf(fid,'%4.0f    %8.4f\n',i,sqrt(i)); end
```

which prints the following table in a file called `sqrt.txt`.

```
    x      sqrt(x)
=====
    1      1.0000
    2      1.4142
    3      1.7321
    4      2.0000
    5      2.2361
```

Again, for more information, refer to MATLAB documentation.

2.10 More loops and conditionals

We have already seen how `for` loops can be used in MATLAB to execute a set of commands a given number of times. Suppose, instead, that one wishes to execute the commands until some condition is satisfied. For example, one might approximate a root of a given function $f(x)$ by first plotting the function on a coarse scale where one can see the approximate root, then plotting appropriate sections on finer and finer scales until one can identify the root to the precision needed. This can be accomplished with the following MATLAB code.

```
xmin = input(' Enter initial xmin: ');
xmax = input(' Enter initial xmax: ');
tol = input(' Enter tolerance: ');
while xmax-xmin > tol,
```

```

x = [xmin:(xmax-xmin)/100:xmax];
y = f(x);
plot(x,y)
xmin = input(' Enter new value for xmin: ');
xmax = input(' Enter new value for xmax: ');
end;

```

The user looks at each plot to determine a value `xmin` that is just left of the root and a value `xmax` that is just right of the root. The next plot then contains only this section, so that closer values `xmin` and `xmax` can be determined. In Chapter 4 we discuss more efficient ways of finding a root of $f(x)$.

Another important statement is the conditional `if` statement. In the above code segment, one might wish to let the user know if the code happens to find a point at which the absolute value of f is less than some other tolerance, say, `delta`. This could be accomplished by inserting the following lines after the statement `y = f(x);`:

```

[ymin,index] = min(abs(y)); % This finds the minimum absolute value of y
                           % and its index.
if ymin < delta,
    fprintf(' f( %f ) = %f\n', x(index), y(index)) % This prints the x and y
                                                    % values at this index.
end;

```

The `if` statement may also contain an `else` clause. For example, to additionally write a message when `ymin` is greater than or equal to `delta`, one could modify the above `if` statement to say:

```

if ymin < delta,
    fprintf(' f( %f ) = %f\n', x(index), y(index)) % This prints the x and y
                                                    % values at this index.
else
    fprintf(' No points found where |f(x)| < %f\n', delta)
end;

```

2.11 Clearing variables

You may clear a particular variable by typing

```
>> clear x
```

or all variables with

```
>> clear all
```

This is important when you want to be sure that all variable names have been erased from memory.

2.12 Logging your session

You can keep a record of your MATLAB session by typing

```
>> diary('hw1.txt')
... some other commands ...
>> diary off
```

This command records all subsequent commands that you type and all responses that MATLAB returns in a file named `hw1.txt`. You will want to name the file by replacing `hw1.txt` with a more descriptive name related to your work. Note, however, that you *cannot* then run the file from MATLAB; this is simply a device for recording what happened during your keyboard session.

Note also that if you execute an M-file in a session logged with the `diary` command, you may want to type `echo on` before executing the M-file. In this way, the commands in the M-file are echoed along with MATLAB's response. Otherwise, the diary file will contain only the responses, not the commands.

2.13 More Advanced Commands

It is perhaps apparent from the reference at the beginning of this chapter to the work on the Mars Exploration Rover, that MATLAB has a large number of commands. To close this chapter, we demonstrate some of the graphical capabilities through an example.

The commands

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```

produce the plot in Figure 2.4. In order to understand this plot, search MATLAB documentation for the commands `meshgrid`, `peaks`, `meshc`, and `axis`. While this chapter will get you started using MATLAB, effective use of the MATLAB documentation will be the key to proceeding to more complicated programs.

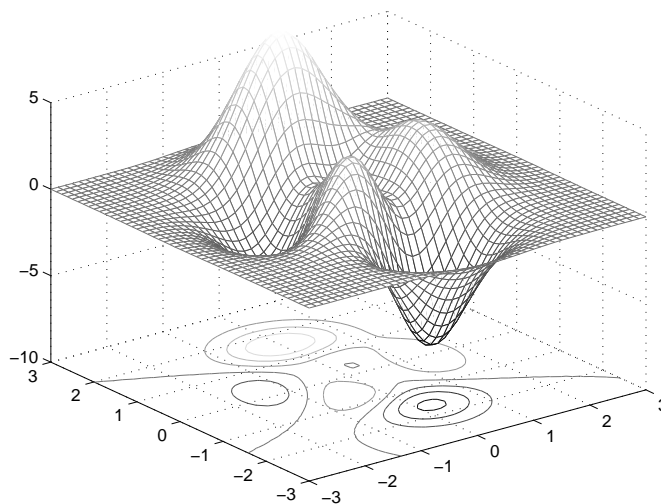


Figure 2.4: A 3D MATLAB plot.

Chapter 2 Exercises

1. Run the examples in this chapter using MATLAB to be sure that you see the same results.
2. With the matrices and vectors

$$A = \begin{pmatrix} 10 & -3 \\ 4 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}, \quad v = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad w = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

compute the following *both* by hand and in MATLAB. For the MATLAB computations, use the `diary` command to record your session.

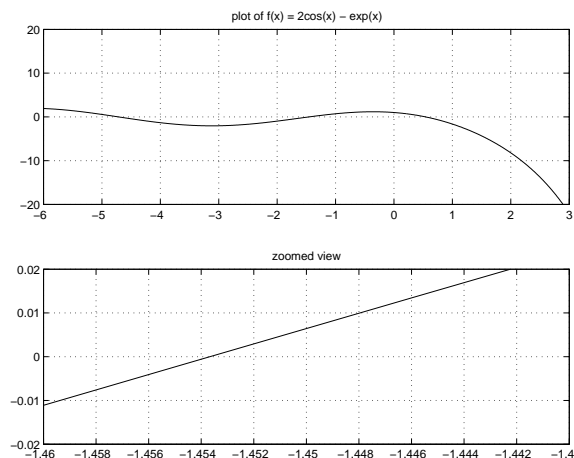
- | | |
|-------------------------------|--|
| (a) $\mathbf{v}^T \mathbf{w}$ | (f) BA |
| (b) \mathbf{vw}^T | (g) $A^2 (= AA)$ |
| (c) $A\mathbf{v}$ | (h) the vector \mathbf{y} for which $B\mathbf{y} = \mathbf{w}$ |
| (d) $A^T \mathbf{v}$ | (i) the vector \mathbf{x} for which $A\mathbf{x} = \mathbf{v}$ |
| (e) AB | |
3. Use MATLAB to produce a single plot displaying the graphs of the functions $\sin(kx)$ across $[0, 2\pi]$, for $k = 1, \dots, 5$.
 4. Use MATLAB to print a table of values x , $\sin x$, and $\cos x$, for $x = 0, \frac{\pi}{6}, \frac{2\pi}{6}, \dots, 2\pi$. Label the columns of your table.

MATLAB'S ORIGINS



CLEVE MOLER—is chairman and chief scientist at MathWorks. In the mid to late 1970s, he was one of the authors of LINPACK and EISPACK, Fortran libraries for numerical computing. To give easy access to these libraries to his students at the University of New Mexico, Dr. Moler invented MATLAB. In 1984, he cofounded The MathWorks with Jack Little to commercialize the MATLAB program. Cleve Moler received his bachelor's degree from California Institute of Technology, and a Ph.D. from Stanford University. Moler was a professor of math and computer science for almost 20 years at the University of Michigan, Stanford University and the University of New Mexico. Before joining The MathWorks full-time in 1989, he also worked for the Intel Hypercube Company and Ardent Computer. Dr. Moler was elected to the National Academy of Engineering in 1997. He also served as president of the Society for Industrial and Applied Mathematics (SIAM) during 2007–2008 [?].

5. Download the file `plotfunction1.m` from the book's web page and execute it. This should produce the two plots shown below. The top plot shows the function $f(x) = 2\cos(x) - e^x$ for $-6 \leq x \leq 3$, and from this plot it appears that $f(x)$ has three roots in this interval. The bottom plot is a zoomed view near one of these roots, showing that $f(x)$ has a root near $x = -1.454$. Note the different vertical scale as well as the different horizontal scale of this plot. Note also that when we zoom in on this function it looks nearly *linear* over this short interval. This will be important when we study numerical methods for approximating roots.



- (a) Modify this script so that the bottom plot shows a zoomed view near the leftmost root. Write an estimate of the value of this root to at least 3 decimal places. You may find it useful to first use the zoom feature in MATLAB to see approximately where the root is and then to choose your axis command for the second plot appropriately.
- (b) Edit the script from part (a) to plot the function

$$f(x) = \frac{4x \sin x - 3}{2 + x^2}$$

over the range $0 \leq x \leq 4$ and also plot a zoomed view near the leftmost root. Write an estimate of the value of the root from the plots that is accurate to 3 decimal places. Note that once you have defined the vector `x` properly, you will need to use appropriate componentwise multiplication and division to evaluate this expression:

```
y = (4*x.*sin(x) - 3) ./ (2 + x.^2);
```

6. Plot each of the functions below over the range specified. Produce four plots on the same page using the `subplot` command.

- (a) $f(x) = |x - 1|$ for $-3 \leq x \leq 3$. (Use `abs` in MATLAB.)
- (b) $f(x) = \sqrt{|x|}$ for $-4 \leq x \leq 4$. (Use `sqrt` in MATLAB.)
- (c) $f(x) = e^{-x^2} = \exp(-x^2)$ for $-4 \leq x \leq 4$. (Use `exp` in MATLAB.)
- (d) $f(x) = \frac{1}{10x^2 + 1}$ for $-2 \leq x \leq 2$.

7. Use MATLAB to plot the circles

$$(x - 2)^2 + (y - 1)^2 = 2,$$

$$(x - 2.5)^2 + y^2 = 3.5$$

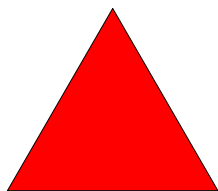
and zoom in on the plot to determine approximately where the circles intersect.

[Hint: One way to plot the first circle is:

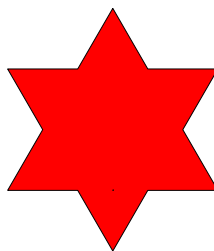
```
theta = linspace(0, 2*pi, 1000);
r = sqrt(2);
x = 2 + r*cos(theta);
y = 1 + r*sin(theta);
plot(x,y)
axis square % so the circles look circular!
```

Use the command `hold on` after this to keep this circle on the screen while you plot the second circle in a similar manner.]

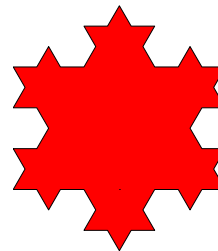
8. In this exercise, you will plot the initial stages of a process that creates a fractal known as *Koch's snowflake*, which is depicted below.



Stage 0



Stage 1



Stage 2

This exercise uses the MATLAB M-file `koch.m`, which you will find on the web page. The M-file contains all the necessary commands to create the fractal, except for the necessary plotting commands. Edit this M-file so that each stage of the fractal is plotted. [Hint: This can be accomplished by adding a plot command just before the completion of the outer for loop.] Add the following commands to keep consistency between plots in the animation.

```
axis([-0.75 0.75 -sqrt(3)/6 1]);
axis equal
```

Note that the `cla` command clears the axes. Finally, add the command `pause(0.5)` in appropriate places to slow the animation. (The `fill` command, as opposed to `plot`, produced the filled fractals depicted above.) We will create fractals using Newton's method in Chapter 4.

9. A magic square is an arrangement of the numbers from 1 to n^2 in an n by n matrix, where each number occurs exactly once, and the sum of the entries in any row, any column, or any main diagonal is the same. The MATLAB command `magic(n)` creates an n by n (where $n > 2$) magic square. Create a 5 by 5 magic square and verify using the `sum` command in MATLAB that the sums of the columns, rows and diagonals are equal. Create a log of your session that records your work. [Hint: To find the sums of the diagonals, read the documentation for the `diag` and the `flipud` commands.]
10. More advanced plotting commands can be useful in MATLAB programming.
 - (a) In the MATLAB command window type


```
[X,Y,Z] = peaks(30);
surf(X,Y,Z);
```
 - (b) Give this plot the title "3-D shaded surface plot".
 - (c) Type `colormap hot` and observe the change in the plot.
 - (d) Print the resulting plot with the given title.

11. Computer graphics make extensive use of matrix operations. For example, rotating an object is a simple matrix–vector operation. In two dimensions, a curve can be rotated counterclockwise through an angle θ about the origin by multiplying every point that lies on the curve by the rotation matrix

$$R = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

As an example, let us rotate the rectangle with vertex coordinates $[1, 0]$, $[0, 1]$, $[-1, 0]$, and $[0, -1]$ through an angle $\theta = \pi/4$. In MATLAB, type the following code to generate the original and rotated squares plotted one on top of the other.

```
% create matrix whose columns contain the coordinates of each vertex.
U = [1, 0, -1, 0; 0, 1, 0, -1];

theta = pi/4;

% Create a red unit square
% Note U(1,:) denotes the first row of U
fill(U(1,:), U(2,:), 'r')

% Retain current plot and axis properties so that
% subsequent graphing commands add to the existing graph
hold on

% Set the axis
axis([-2 2 -2 2]);

% Perform rotation.
R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
V = R*U;

fill(V(1,:), V(2,:), 'b');

axis equal tight, grid on
```

Note that the `fill` command in MATLAB plots a filled polygon determined by two vectors containing the x - and y -coordinates of the vertices.

- (a) Adapt this code to plot a triangle with vertices $(5, 0)$, $(6, 2)$ and $(4, 1)$. Plot the triangles resulting from rotating the original triangle by $\pi/2$, π , and $3\pi/2$ radians about the origin. Plot all four triangles on the same set of axes.
- (b) Rotating by θ radians and then rotating by $-\theta$ radians leaves the figure unchanged. This corresponds to multiplying first by $R(\theta)$ and then by $R(-\theta)$. Using MATLAB,

verify that these matrices are inverses of each other (i.e., their product is the identity) for $\theta = \pi/3$ and for $\theta = \pi/4$.

- (c) Using the trigonometric identities $\cos \theta = \cos(-\theta)$ and $-\sin \theta = \sin(-\theta)$, prove that $R(\theta)$ and $R(-\theta)$ are inverses of each other for any θ .
- (d) Let R be the matrix that rotates counterclockwise through the angle $\pi/8$, and let $\hat{R} = 0.9 * R$. Then the matrix \hat{R} simultaneously rotates and shrinks an object. Edit the code above to repeatedly rotate and shrink (by the same amounts on each step) the square (again originally with coordinates $[1, 0]$, $[0, 1]$, $[-1, 0]$, and $[0, -1]$) for 50 iterations. The plot should show all of the 51 squares on the same set of axes.
- (e) Apply \hat{R} but now after each rotation translate the square by 1 unit in the x direction and 2 units in the y direction. Note, you must apply the rotation and translation to all the vertices of the square. The resulting squares should visually suggest the existence of a fixed point. Such a point satisfies the equation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \hat{R} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Solve this equation to find the numerical value of this fixed point.

12. The previous exercise used rotation matrices in two dimensions. Now we explore the speed of matrix operations in a computer graphics model in three dimensions. In Figure 2.5, we see a model of Yoda. The tessellation contains 33,862 vertices. Let V be a matrix with 3 columns and 33,862 rows, where row i contains the x -, y -, and z -coordinates of the i th vertex in the model. The image can be translated by t units in the y direction by using a translation matrix T where

$$T = \begin{pmatrix} 0 & t & 0 \\ 0 & t & 0 \\ \vdots & \vdots & \vdots \\ 0 & t & 0 \end{pmatrix}.$$

If $V_t = V + T$, then V_t contains the vertex information for the model after a translation of t units in the y direction.

Download the files `yoda.m` and `yodapose_low.mat` from the web page. Run the file `yoda.m` in MATLAB. You will see an animation of the model being translated in space using matrix addition.

- (a) The image can be rotated by θ radians about the y -axis by multiplying V on the right by R_y where

$$R_y = \begin{pmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{pmatrix}.$$

Edit the code to continuously rotate the image by $\pi/24$ radians until the image has made one full rotation about the y -axis.

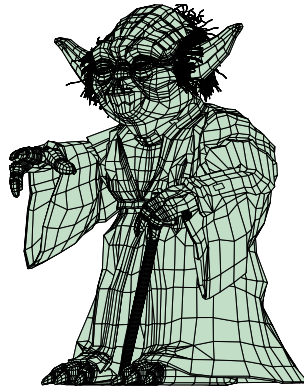


Figure 2.5: A model of Yoda created with 33,862 vertices. Model created by Kecskeneti B. Zoltan. ©Lucasfilm Ltd & TM. All Rights Reserved. Digital Work by Industrial Light & Magic.

- (b) How many multiplications are performed when you use matrix multiplication (with R_y) to rotate the image once by $\pi/24$ radians? (Remember that V is a 33,862 by 3 matrix.) Keep this in mind as you watch how fast MATLAB performs the calculations and displays the results.

13. You can see a picture of a mandrill by typing the MATLAB commands

```
load mandrill, image(X), colormap(map), axis off equal
```

Each row of the 220 by 3 matrix `map` corresponds to a color with the first, second, and third elements specifying the intensity of red, green, and blue, respectively.

- (a) Write down a 3 by 3 matrix T which, when applied to `map` on the right, will reverse the order of the columns. Use the MATLAB commands `map2 = map*T; colormap(map2)` to see the effect. Explain what happened and why.
- (b) Write down a 3 by 3 matrix S which, when applied to `map` on the right, leaves columns one and two unchanged but replaces column three by a column of 0s. Use the MATLAB commands `map3=map*S; colormap(map3)` to see the effect of this change. Explain what happened and why. You may need to type `help colormap` to find out exactly what this new color map does.
14. In this exercise, we will create a fractal coastline. Fractals have many uses, and here we see how they can be used to create qualitatively realistic-looking pictures.
- We will use the following iterative algorithm to create two-dimensional fractal landscapes.

0. Begin with one straight line segment.

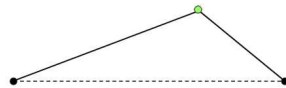
1. For each line segment in the current figure, find the midpoint, denoted by a solid diamond in the picture below.



2. Create a new point by moving a random amount in the x and y directions from that midpoint as seen below. The size of the random displacement will be adjusted at each iteration.

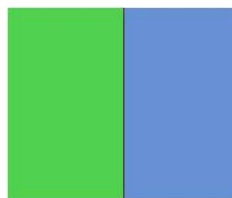


3. Connect the endpoints of the original line with the new point.

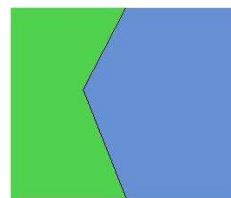


4. If the picture looks good then stop, else adjust the random displacement size and go to step 1.

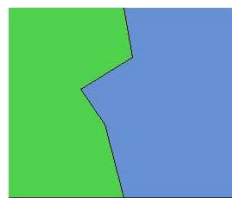
You will need to determine a suitable range for the random displacement at each iteration to obtain a realistic-looking picture. One such choice resulted in the figures below.



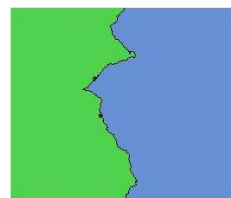
Iterate 0



Iterate 1



Iterate 2

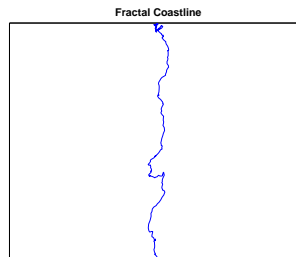


Iterate 8

For the exercise, you need not color the land and water masses (although you may do so by using the `fill` command), but simply generate a realistic-looking coastline. If your implementation stores the x values and y values for points on the fractal coastline in the vectors `xValues` and `yValues`, respectively, then the MATLAB commands:

```
plot(xValues,yValues)
axis equal
```

will plot the fractal coastline with a 1:1 aspect ratio for the axes.



- (a) Write a program to create fractal coastlines using the algorithm above.
 - (b) Describe how your implementation adjusts the range of the random numbers used for displacements in step 2 at each iteration.
 - (c) Create at least two fractal coastlines with your code.
15. Find documentation on the `movie` command by typing `helpdesk` or `doc movie`. At the bottom of the documentation on this command, you will find the code:

```
Z = peaks;
surf(Z);
axis tight
set(gca,'nextplot','replacechildren');
for j = 1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j) = getframe;
end movie(F,5)
```

Cut and paste this code into the MATLAB command window and describe the results. Proficiency in MATLAB programming can increase dramatically through effective use of the available documentation.