Introduction to MATLAB

MATLAB is a general-purpose computing environment that is ideally suited for implementing mathematical and numerical methods. It is used as a high-powered calculator for small problems and as a full-featured programming language for large problems. A helpful feature of MATLAB is its long list of high-quality library functions that can make complicated calculations short, precise, and easy to write in high-level code.

This section contains a brief introduction to MATLAB's commands and features. Much more detailed accounts can be found in MATLAB's help facilities, the MATLAB User's Guide, and in books [3, 1] and websites devoted to the package.

B.1 STARTING MATLAB

On PC-based systems, MATLAB is started by clicking the appropriate icon and ended by clicking on File/Exit. On Unix-based systems, type MATLAB at the system prompt:

```
$ matlab
```

Then type

```
>> exit
```

to exit.

Type the command

```
>> a=5
```

generated by the return key. MATLAB will echo the information back to you. Type the additional commands

```
>> b=3
>> c=a+b
>> c=a*b
>> d=log(c)
>> who
```

to get an idea of how MATLAB works. You may include a semicolon after a statement to suppress echoing of the value. The who command gives a list of all variables you have defined.

MATLAB has an extensive online help facility. Type help log for information on the log command. The PC version of MATLAB has a Help menu that contains descriptions and usage suggestions on all commands.
To erase the value of the variable \( a \), type `clear a`. Typing `clear` will erase all previously defined variables. To recover a previous command, use the up cursor key. If you run out of room on the current command line, end the line with three periods and a return; then resume typing on the next line.

To save values of variables for your next login, type `save`, then `load` on your next login to MATLAB. For a transcript of part or all of the MATLAB session, type `diary filename` to start logging, and `diary off` to end. Use a filename of your choice for filename. This is helpful for submitting your work for an assignment. The `diary` command produces a file that can be viewed or printed once your MATLAB session is over.

MATLAB normally performs all computations in IEEE double precision, about 16 decimal digits of accuracy. The numeric display format can be changed with the `format` statement. Typing `format long` will change the way numbers are displayed until further notice. For example, the number 1/3 will be displayed differently depending on the current format:

```matlab
format short 0.3333 format short e 3.3333e-001 format long 0.33333333333333 format long e 3.3333333333333333e-001 format bank 0.33 format hex 3f05555555555555
```

More control over formatting output is given by the `fprintf` command. The commands

```matlab
>> x=0:0.1:1;
>> y=x.^2;
>> fprintf('%.5f %.5f \n',[x;y])
```

will print the table

```
0.00000 0.00000
0.10000 0.01000
0.20000 0.04000
0.30000 0.09000
0.40000 0.16000
0.50000 0.25000
0.60000 0.36000
0.70000 0.49000
0.80000 0.64000
0.90000 0.81000
1.00000 1.00000
```

### B.2 GRAPHICS

To plot data, express the data as vectors in the X and Y directions. For example, the commands

```matlab
>> a=[0.0 0.4 0.8 1.2 1.6 2.0];
>> b=sin(a);
>> plot(a,b)
```

will draw a piecewise-linear approximation to the graph of \( y = \sin x \) on \( 0 \leq x \leq 2 \), as shown in Figure B.1(a). In this case, \( a \) and \( b \) are 6-dimensional vectors, or 6-element arrays. The font of the axis numbers can be set to 16-point, for example, by the command `set(gca,'FontSize',16)`. A shorter way to define the vector \( a \) is the command
Figure B.1 MATLAB figures. (a) Piecewise linear plot of \( f(x) = \sin x \), with \( x \) increment of 0.4. (b) Another piecewise plot looks smooth because the \( x \) increment is 0.02.

\[
\begin{align*}
&\text{>> a}=0:0.4:2; \\
&\text{This command defines } a \text{ to be a vector whose entries begin at 0, increment by 0.4, and end at 2, identical to the previous longer definition. A more accurate version of one entire cycle of the sine curve results from} \\
&\text{>> a}=0:0.02:2*pi; \\
&\text{>> b} = \sin(a); \\
&\text{>> plot(a,b)}
\end{align*}
\]

and is shown in Figure B.1(b).

To draw the graph of \( y = x^2 \) on \( 0 \leq x \leq 2 \), one could use

\[
\begin{align*}
&\text{>> a}=0:0.02:2; \\
&\text{>> b} = a.^2; \\
&\text{>> plot(a,b)}
\end{align*}
\]

There is one strange and unexpected character. The period preceding the power operator tells MATLAB to square each entry of the vector \( a \). As we will see in the next section, MATLAB treats every variable as a matrix, or doubly indexed array. Omitting the period in this instance would mean multiplying the \( 10 \times 1 \) matrix \( a \) by itself, under the rules of matrix multiplication, which is impossible. If you ask MATLAB to do this, it will complain. In general, MATLAB interprets an operation preceded by a period to mean that the operation should be applied entry-wise, not as matrix multiplication.

There are more advanced techniques for plotting graphs. MATLAB will choose axis scaling automatically if it is not specified, as in Figure B.1. To choose the axis scaling manually, use the \texttt{axis} command. For example, following a plot with the command

\[
\text{>> v} = [-1 \ 1 \ 0 \ 10]; \ \text{axis(v)}
\]

sets the graphing window to \([-1, 1] \times [0, 10]\). The \texttt{grid} command draws a grid behind the plot.
Use the command `plot([x1,y1,x2,y2,x3,y3])` to plot three curves in the same graph window, where \(x_i, y_i\) are pairs of vectors of the same lengths. Type `help plot` to see the choices of solid, dotted, and dashed line types and various symbol types (circles, dots, triangles, squares, etc.) for plots. Semilog plots are available through the `semilogx` and `semilogy` commands.

The `subplot` command splits the graph window into multiple parts. The statement `subplot(abc)` breaks the window into an \(a \times b\) grid and uses the \(c\) box for the plot. For example,

```matlab
>> subplot(121), plot(x,y)
>> subplot(122), plot(x,z)
```

plots the first graph on the left side of the screen and the second on the right. The `figure` command opens up new plot windows and moves among them, if you need to view several different plots at once.

Three-dimensional surface plots are drawn with the command `mesh`. For example, the function \(z = \sin(x^2 + y^2)\) on the domain \([-1, 1] \times [-2, 2]\] can be graphed by

```matlab
>> [x,y]=meshgrid(-1:0.1:1,-2:0.1:2);
>> z=sin(x.^2+y.^2);
>> mesh(x,y,z)
```

![Figure B.2 Three-dimensional MATLAB plot. The `mesh` command is used to plot surfaces.](image)
The vector `x` created by `meshgrid` is 41 rows of the 21-vector `-1:0.1:1`, and similarly, `y` is 21 columns of the column vector `-2:0.1:2`. The graph produced by this code is shown in Figure B.2. Replacing `mesh` with `surf` plots a colored surface over the mesh.

### B.3 PROGRAMMING IN MATLAB

More sophisticated results can be achieved by writing programs in the MATLAB language. An m-file is a file containing a list of MATLAB commands. The filename of an m-file has a suffix of `.m`. For example, you might use your favorite editor, or the MATLAB editor if available, to create the file `cubrt.m`, containing the following lines:

```
% The program `cubrt.m` finds a cube root by iteration
y=1;
n=15;
y=input('Enter z:');
for i = 1:n
    y = 2*y/3 + z/(3*y^2)
end
```

To run the program, type `cubrt` at the MATLAB prompt. The reason that this code converges to the cube root will become evident from our study of Newton's method in Chapter 1. Notice that the semicolon was dropped from the line that defines the new `y` by iteration. This allows you to see the progression of approximants as they approach the cube root.

With the graphics ability of MATLAB, we can analyze the data from the cube root algorithm. Consider the program `cubrt1.m`:

```
% The program `cubrt1.m` finds cube roots and displays its progress
y(1)=1;
n=15;
y=input('Enter z:');
for i = 1:n-1
    y(i+1) = 2*y(i)/3 + z/(3*y(i)^2);
end
plot(1:n,y)
title('Iterative method for cube roots')
xlabel('Iteration number')
ylabel('Approximate cube root')
```

Run the foregoing program with `z = 64`. When finished, type the commands

```
>> e=y-4;
>> plot(1:n,e)
>> semilogy(1:n,e)
```

The first command subtracts the correct cube root 4 from each entry of the vector `y`. This remainder is the error `e` at each step of the iteration. The second command plots the error, and the third plots the error in a semilog plot, using logarithmic units in the y-direction.

### B.4 FLOW CONTROL

The `for` loop was introduced in the previous cube root program. MATLAB has a number of commands to control the flow of a program. A number of these, including `while` loops
and if and break statements, will be familiar to anyone with knowledge of a high-level programming language. For example,

```matlab
n=5;
for i=1:n
    for j=1:n
        a(i,j)=1/(i+j-1);
    end
end
a
```

creates and displays the $5 \times 5$ Hilbert matrix. The semicolon avoids repeated printing of partial results, and the final a displays the final result. Note that each for must be matched with an end. It is a good idea, though not required by MATLAB, to indent loops for greater readability.

The while command works similarly:

```matlab
n=5;i=1;
while i<=n
    j=1;
    while j<=n
        a(i,j)=1/(i+j-1);
        j=j+1;
    end
    i=i+1;
end
a
```

This produces the same result as the double for loop.

The if statement is used to make flow decisions, and the break command provides an exit jump out of the next inner loop. Both are illustrated as follows:

```matlab
% To compute the nth derivative of sin(x) at x=0
n=input('Enter n, negative number to quit:');
if n<=0,break,end
r=rem(n,4) % rem is the remainder function
if r==0
    y=0
elseif r==1
    y=1
elseif r==2
    y=0
else
    y=-1
end
y
```

The logical operators & and | stand for AND, OR, respectively. The error command stops execution of the m-file and reports information to the user.
B.5 Functions

Creating an m-file to hold MATLAB code is preferred if the calculation will take more than a few lines. An m-file can call other m-files, including itself. (Typing (ctrl)-C will usually abort runaway MATLAB processes.)

A MATLAB function is a particular type of m-file to which values can be passed. The syntax of the first line must be adhered to, as in the following example, where the filename is f.m:

```matlab
function y=f(x)
% Evaluates the function sin(log(x)), if it makes sense,
% otherwise returns zero.
if x>0
    y=sin(log(x));
else
    y=0
end
```

The only way a function differs from a regular m-file is in the first line. The filename, with the .m omitted, should agree with the function name in the first line. A MATLAB function is therefore a special m-file. Variables in a function file are local by default, but can be made global with the global command.

A more complicated function can use several variables as inputs and several as outputs. For example, here is a function that calls the existing MATLAB functions mean and std and collects both in an array:

```matlab
function [m,sigma]=stat(x)
% Returns sample mean and standard deviation of input vector x
m=mean(x);
sigma=std(x);
```

If this file stat.m resides in your MATLAB path, typing stat(x), where x is a vector, will return the mean and standard deviation of the entries of the vector.

The nargin command provides the number of input arguments to a function. With this command, the work of a function can change, depending on how many arguments are presented to it. An example of nargin is given in Program 0.1 on nested multiplication.

MATLAB allows a variety of methods of calling a function from another function. To be as clear as possible, in this book we will usually “hard-wire” the call by including a function definition in the calling function. Suppose the goal is to approximate the derivative of sin\(2x\) at \(x = 0\). We could create a file deriv.m that implements a method from Chapter 5:

```matlab
function y=deriv(x,h)
% Returns derivative approximation at x with step size h
y=(f(x+h)-f(x-h))/(2*h);
```

```matlab
function y=f(x)
y=sin(2*x);
```

Then the line command

```
>> deriv(0,0.0001)
```
yields the approximation. The function \( f \) can be defined in \texttt{deriv.m} or as a standalone \texttt{f.m} file. This approach is transparent, but lacks elegance because it requires altering the \texttt{deriv.m} file, or the \texttt{f.m} file, whenever changing the input function.

A more flexible alternative is to input the function as a so-called \textit{inline function}. For example, create the file \texttt{deriv1.m} as follows:

```matlab
function y=deriv1(f,x,h)
% Returns derivative approximation at x with step size h
y=(f(x+h)-f(x-h))/(2*h);
```

Note that the function is now also an input argument. Then the line commands

```matlab
>> f=inline('sin(2*x)','x');
>> deriv1(f,0,0.0001)
```

approximate the derivative. In this approach, it is easy to vary \( f \), and in addition it emphasizes that functions can be viewed as inputs.

In case \( f(x) \) is too complicated to fit on a single line, there is a short detour to defining it as an inline function. We can start with any function file, such as \texttt{f1.m},

```matlab
function y=f1(x)
x=2*pi;
y=sin(x);
```

and follow with the line commands

```matlab
>> f=inline('f1(x)', 'x');
>> deriv1(f,0,0.0001)
```

to approximate the derivative. The reader is encouraged to use the inline idea, where convenient, to write efficient, flexible code.

## B.6 MATRIX OPERATIONS

The key to MATLAB's power and versatility is the sophistication of its variables' data structure. Each variable in MATLAB is an \( m \times n \) matrix of double precision floating point numbers. A scalar is simply the special case of a \( 1 \times 1 \) matrix. The syntax

```matlab
>> A=[1 2 3
     4 5 6]
```

or

```matlab
>> A=[1 2; 3; 4 5 6]
```

defines a \( 2 \times 3 \) matrix \( A \). The command \texttt{B=A'} creates a \( 3 \times 2 \) matrix \( B \) that is the transpose of \( A \). Matrices of the same size can be added and subtracted with the + and - operators. The command \texttt{size(A)} returns the dimensions of the matrix \( A \), and \texttt{length(A)} returns the maximum of the two dimensions.

MATLAB provides many commands that allow matrices to be easily built. For example, \texttt{zeros(m,n)} produces a matrix full of zeros of size \( m \times n \). If \( A \) is a matrix, then \texttt{zeros(size(A))} produces a matrix of zeros of the same size as \( A \). The commands \texttt{ones(m,n)} and \texttt{eye(m,n)} (for the identity matrix) work essentially the same way. For example,
is a convoluted, but accurate way to construct the $4 \times 4$ identity matrix.

The colon operator can be used to extract a submatrix from a matrix. For example,

```matlab
>> A=[eye(2) zeros(2,2); zeros(2,2) eye(2)]
```

assigns to $\mathbf{b}$ the first three entries of the second column of $\mathbf{A}$. The command

```matlab
>> b=A(:,2)
```

assigns to $\mathbf{b}$ the entire second column of $\mathbf{A}$, and

```matlab
>> B=A(:,1:3)
```

assigns to $\mathbf{B}$ the submatrix consisting of the first three columns of $\mathbf{A}$.

The $n \times n$ matrix $\mathbf{A}$ and the $n \times p$ matrix $\mathbf{B}$ can be multiplied by the command $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$. If the matrices have inappropriate sizes, MATLAB will refuse to do the operation and return an error message.

## B.7 Animation

The field of differential equations includes the study of dynamic systems, or "things that move." MATLAB makes animation easy, and these aspects are exploited in Chapter 6 to follow solutions that are changing with time.

The sample MATLAB program `bounce.m` given next shows a tennis ball bouncing from wall to wall in a unit square. The `first` command sets up parameters of the current figure (gca), including the axis limits, $0 \leq x, y \leq 1$. The `cla` command clears the figure window, and `axis square` equalizes the units in the $x$ and $y$ directions.

Next, the `line` command is used to define a line object called `ball`, along with its properties. The `erase` parameter set to `xor` means that each time the ball is drawn, its previous position is erased. The four `if` statements in the `while` loop cause the ball to reverse velocity when it hits one of the four walls. The loop also contains a `set` command that updates the current $x$ and $y$ coordinates of the line object `ball`, by setting its `xdata` and `ydata` attributes, respectively. The `drawnow` command draws all defined objects to the current figure window. The speed of the moving ball can be adjusted with the `pause` command and through the step sizes `hx` and `hy`. The `while` loop is infinite and can be interrupted by `end`.

```matlab
% bounce.m
% Illustrates Matlab animation using the `drawnow` command
% Usage: Save this file in bounce.m, then type "bounce"
set(gca, 'XLim', [0 1], 'YLim', [0 1], 'Drawmode', 'fast', ...
    'Visible', 'on');
cla
axis square
ball = line('color', 'r', 'Marker', 'o', 'MarkerSize', 10, ...
    'LineWidth', 2, 'erase', 'xor', 'xdata', [], 'ydata', []);
hx=0.05;hy=0.0039;hx=hx0;hy=hy0;
xl=.02;xl=.98;yl=xl;yt=xr;xl=.1;yl=.1;
```
while 1 == 1
    if x < xl
        hx = hx0;
    end
    if x > xr
        hx = -hx0;
    end
    if y < yb
        hy = hy0;
    end
    if y > yt
        hy = -hy0;
    end
    x = x + hx; y = y + hy;
    set(ball, 'xdata', x, 'ydata', y); drawnow; pause(0.01)
end