

Matlab Tutorial

a) Starting Up

Launch the Matlab application from the **Start** button, **Programs** menu. You should eventually see the Matlab window with an entry prompt, `>>`, on the left. The front end of Matlab operates in sequential fashion as you type in commands line by line. For each command, you get a result. It's basically a fancy calculator.

b) Assignment of Values to Variable Names

Assignment of values to scalar variables is similar to other computer languages. Try typing

```
a = 4
and
A = 6
```

Note how the assignment echoes to confirm what you have done. This is a characteristic of Matlab. The echo can be suppressed by following the command line with the semicolon (;) character. Try typing

```
b = -3 ;
```

Matlab treats names in a case-sensitive manner; that is, the name **a** is not the same as the name **A**. To illustrate this, enter

```
a
and
A
```

See how their values are distinct. They are distinct names.

Variable names in Matlab generally represent matrix quantities. A row vector can be assigned as follows

```
a = [ 1 2 3 4 5 ]
```

The echo confirms the assignment again. Notice how the new assignment of **a** has taken over. In practice, row vectors are rarely used to solve mathematical problems. The assignment of **a** is shown simply to illustrate their use in Matlab. When we speak of *vectors*, this usually refers to column vectors, which are more commonly used. A column vector can be entered in several ways. Try them

```
b = [ 1 ; 2 ; 3 ; 4 ; 5 ]
or
b = [ 1;
      2;
      3;
      4;
      5 ]
```

or, by transposing a row vector with the `'` operator

```
b = [ 1 2 3 4 5 ]'
```

A two-dimensional matrix of values can be assigned as follows. Note that the last row of the matrix A is 7 8 **8**, not 7 8 9.

```
A = [ 1 2 3 ; 4 5 6 ; 7 8 8 ]  
or  
A = [ 1 2 3 ;  
      4 5 6 ;  
      7 8 8 ]
```

The values stored by a variable can be examined at any time by typing the name alone, e.g.

```
b  
or  
A
```

Also, a list of all current variables can be obtained by entering the command

```
who  
or, with more detail, enter  
whos
```

There are several predefined variables, e.g. **pi**. Try typing

```
pi
```

It is also possible to assign complex values to variables, since Matlab handles complex arithmetic automatically. To do this, it is convenient to assign a variable name, usually either **i** or **j**, to the square root of **-1**,

```
i = sqrt (-1 )
```

Then a complex value can be assigned

```
x = 2 + i * 4
```

c) Mathematical Operations

Operations with scalar quantities are handled in a straightforward manner, similar to computer languages. The common operators, in order of precedence, are

- ^** exponentiation
- * /** multiplication and division
- ** left division (applies to matrices)
- + -** addition and subtraction

These operators will work in calculator fashion. Try

```
2 * pi
```

Also, scalar real variables can be included.

$$y = \pi / 4$$

$$y ^ 2.45$$

Results of calculations can be assigned to a variable, as in the next-to-last example, or simply displayed, as in the last example.

Calculations can also involve complex quantities. Using the **x** defined above, try

$$3 * x$$

$$1 / x$$

$$x ^ 2$$

$$x + y$$

The real power of Matlab is illustrated in its ability to carry out matrix calculations. In order to illustrate vector-matrix multiplication, first redefine **a** and **b**,

$$a = [1 ; 2 ; 3]$$

and

$$b = [4 ; 5 ; 6]$$

The inner product of two vectors (dot product) can be calculated using the ***** operator,

$$a' * b$$

and likewise, the outer product

$$b * a'$$

Now, try

$$a' * A$$

or

$$A * b$$

What happens when dimensions are not those required for the operation? Try

$$A * a'$$

Matrix-matrix multiplication is carried out in likewise fashion

$$A * A$$

Mixed operations with scalars are also possible.

$$A / \pi$$

It is important to remember always that Matlab will apply the simple arithmetic operators in vector-matrix fashion if possible. At times, you will want to carry out calculations item-by-item in a matrix. Matlab provides for that too. For example,

```
A ^ 2
```

results in matrix multiplication of **A** with itself. What if you want to square each element of **A**? That can be accomplished with

```
A .^ 2
```

The `.` preceding the operator `^` signifies that the operation is to be carried out item-by-item. The Matlab manual calls these *array operations*.

Matlab contains a helpful shortcut for performing calculations that you've already done. Press the up-arrow key. You should get back the last line you typed in.

```
A .^ 2
```

Pressing Enter will perform the calculation again. But, you can also edit this line. For example, change it to the line below and then press Enter.

```
A .^ 3
```

Using the up-arrow key, you can go back to any command that you entered. Press the up-arrow until you get back the line

```
b*a'
```

Alternatively, you can type **b** and press the up-arrow once and it will automatically bring up the last command beginning with the letter "b." The up-arrow shortcut is a quick way to fix errors without having to retype the entire line.

d) Use of Built-in Functions

Matlab has a rich collection of built-in functions. You can use on-line help to find out more about them. One of their important properties is that they will operate directly on vector and matrix quantities. For example, try

```
log ( A )
```

and you will see that the natural logarithm function is applied in array style, element by element, to the **A** matrix. Most functions, like *sqrt*, *abs*, *sin*, *acos*, *tanh*, *exp*, operate in array fashion. Certain functions, like exponential and square root, have matrix definitions also. Matlab will evaluate the matrix version when the letter **m** is appended to the function name. Try

```
sqrtm ( A )
```

A common use of functions is to evaluate a formula for a series of arguments. Create a column vector **t**, which contains values from 1 to 101 in steps of 5,

```
t = [ 1 : 5 : 101 ]'
```

Check the number of items in the **t** array with the *length* function,

```
length ( t )
```

Now, say that you want to evaluate a formula $y = f(t)$, where the formula is computed for each value of the **t** array, and the result is assigned to a corresponding position in the **y** array. For example, (caution: watch the spacing around the periods)

```
y = t .^ 0.34 - log10(t) + 1 ./ t
```

Done! [Note the use of the array operators with the decimal point.] This is similar to creating a column of the **t** values on a spreadsheet and copying a formula down an adjacent column to evaluate **y** values.

There are many, many functions in Matlab. Refer to the help facility.

e) Graphics

Matlab's graphics capabilities have similarities with those of a spreadsheet program: graphs can be created quickly and conveniently, however, there is not much flexibility to customize them.

For example, to create a graph of the **t**, **y** arrays from above, enter

```
plot ( t , y )
```

That's it! You can customize the graph a bit with commands like the following

```
title ( 'Plot of y versus t by yourname' )
```

Your plot is created in a separate window. You can move back and forth from the plot window to the main Matlab window by clicking in it or using the Alt-Tab key combination.

```
xlabel ( 'Values of t' )
```

```
ylabel ( 'Values of y' )
```

```
grid
```

There are other features of graphics which will become useful, plotting objects instead of lines, families of curves, plotting on the complex plane, multiple graph windows on the screen, log-log or semilog plots, 3-D mesh plots, and contour plots.

f) Programming in Matlab

In the front end of Matlab, a series of commands can be written by stringing commands together on a single line, separated by either a semicolon (;) or a comma (,). Try typing

```
z = 5; y = 4, x = y * z
```

Notice that the comma does not suppress the echo feature of Matlab. Stringing commands together in this way is valuable when finding the speed of an operation. Type the following.

```
t0 = clock; inv(A), etime(clock,t0)
```

This series of commands determines the inverse of the matrix A and displays the time required to perform this calculation. Although this can be a useful feature, it can only take you so far. In order to write a full program, we must create a Matlab document, called an *m-file*. Click on **File, New, M-file**. A new window will open with the heading "MATLAB Editor/Debugger." In this window, you can type and edit Matlab programs. Type the following short program

```
height = 3;  
radius = input('Enter Radius: ');  
Volume = pi * radius ^ 2 * height
```

This program will calculate the volume of a cylinder. These commands are written exactly as they would be written in the front end of Matlab. Notice that the height of the cylinder is set at 3, but the radius will be input by the user. Notice also the use of semicolons. The first two lines have semicolons to keep Matlab from printing height and radius. The last line, however, has no semicolon because we want the program to display the volume. Save this program with the name **cylinder**. The file is automatically saved with the extension **.m** to mark it as an m-file. To run this program you must go back to the front end of Matlab. One way to do this is to click the "MATLAB Command Window" button on the task bar.

Now we need to tell Matlab where to look for our m-files. Click on File, Set Path. A window titled "Path Browser" will open. Add the location where cylinder was saved and press enter. Now exit the Path Browser by clicking on the "X" in the upper right corner of the window. Every time you start-up Matlab, you will need to change the path in order to use m-files. Run the program by typing

```
cylinder
```

The program will prompt you for a radius. Enter a value of 3. Hopefully, the volume returned is 84.823. If not, fix your error(s) until you get this result. Notice that the command to use the m-file is the file name "cylinder." Notice also that "cylinder" is never specified in the text of the file.

All that's left now is to learn the syntax and structures of Matlab m-files. First, we will introduce the relational and logical operators in Matlab.

g) Relational and Logical Operators

<i>Relational Operators</i>	<i>Matlab Operators</i>
=	==
<	<
>	>
≤	<=
≥	>=
≠	~=

<i>Logical Operators</i>	<i>Matlab Operators</i>
And	&
Or	 (Above Enter key)
Not	~

h) Conditional Selection

The following three sections contain a few lines of code written in Matlab:

example 1

```
if (x < 0 & y < 0)
    z = -x * y;
elseif (x == 0 | y == 0)
    z = 0;
else
    z = x^2;
end
```

example 2

```
sum = 0;
x = 1;
while(x < 4)
    sum = sum + 1/x;
    x = x + 1;
end
```

solution:
sum = 1.833

example 3

```
sum = 0;
length = 10;
for i = 0 : 2 : length
    sum = sum + i;
end
```

solution:
sum = 30

In Matlab, if the increment is one you may leave it out. In this case, the for-line above would become **for i = 0:length**. When using programs with loops, you should also know how to stop a program in the middle of its execution, in case one of your programs contains an infinite loop. While a program is running, the **Ctrl-C** command will stop it. When you use this command, Matlab will display the line it was trying to execute when the Ctrl-C command took effect. *Do not be misled by this output. It will not necessarily tell you where the error is in your program.*

i) Programming in Matlab

The m-files that you have written so far are like SUBROUTINES in Pascal or FORTRAN in that they perform a series of operations but do not return a specific value. A **function** is a special type of m-file that, not surprisingly, acts like a FUNCTION in Pascal or FORTRAN (or C). A function starts out with a specific line that looks like this:

```
function z = func_name (x,y)
```

The first word must be **function**, and then there must be a variable that will be the output of the function. This variable is followed by the equals sign and then the name of the function (this must be the name of the m-file), and finally the variables that are passed to the function. The function file then operates just like any other m-file, except that it ends by assigning a value to the variable in the definition line. Let's see how it works. Create a new m-file called **quadrat.m**, and type it in as follows:

```
function x = quadrat(a,b,c)
if a == 0
    x = -b/c;
else
    disc = b * b - 4 * a * c;
    x = (-b + sqrt(disc)) / (2 * a);
end
```

save your m-file and type **quadrat(1,2,1)** at the command prompt. Note that Matlab goes out, finds the function, executes it, and returns the correct value. Now, we all know that the quadratic formula has two solutions. Modify your m-file as follows:

```
function [x,y] = quadrat(a,b,c)
if a == 0
    x = -b/c;
    y = -b/c;
else
    disc = b * b - 4 * a * c;
    x = (-b + sqrt(disc)) / (2 * a);
    y = (-b - sqrt(disc)) / (2 * a);
end
```

Run the function by typing **[x1,x2] = quadrat (1,3,2)**. Did you get the results that you thought you should? Note that in this case we also assigned the function results to variables so that we could reuse them (you can confirm this by typing **x1**). In the first case we did not, but we could have.

j) Comments

One final item of note is the comment character, %. When you type % in an m-file, Matlab ignores everything after the % sign. Thus, the following text would be equivalent to the m-file that you typed earlier:

```
% M-file to calculate volume of cylinder of height 3 and user-specified radius
% Ima Student
% APPM 4650
% Fall 2003

height = 3; % Set h = 3
radius = input('Enter Radius: '); % Set r = User Value
Volume = pi * radius ^ 2 * height % V = pi * r^2 * h

% End of m-file.
```

Comments can be particularly valuable for keeping track of what variable names mean and what units you are using. We strongly recommend their use.