MATLAB:
a Brief Introduction

by

Robert L. Rankin

Spring 1996

MATLAB: a Brief Introduction

R.1 Introduction to MATLAB

MATLAB is an acronym for **MAT**rix **LAB**oratory and is a highly optimized numerical computation and matrix manipulation package. Within the environment you can execute all MATLAB functions from the command line or you can write MATLAB programs to issue entire sequences of commands. The built–in programming language is a variant of the C programming language with many additions and a few deletions.

This review assumes that MATLAB (either the student or professional version) is already installed in a DOS, Windows or Macintosh operating environment and that you already know how to run the program. After starting the program a number of messages will appear and you will eventually see the MATLAB prompt, ». This means that MATLAB is waiting for you to enter a command. For example you might type `demo`, or `help`, or `info` to see some of the features and commands in MATLAB.

To leave the MATLAB environment, simply type `quit` at any prompt or select ***Quit*** from the ***File Menu***.

R.1.1 Desktop Calculations

MATLAB can be used in much the same way as a calculator. The four elementary arithmetic operators in MATLAB are + for *addition*, – for *subtraction*, * for *multiplication* and / or \ for *division*. A few examples of these operations are shown below.

```
»2 + 3
ans =
     5
»2 - 2
ans =
     0
»2 * 2
ans =
     4
»2/3
ans =
    0.6667
```

Notice that all un–named operations produce an immediate output in the form `ans = value`. The symbol ans is actually a variable in MATLAB with can then be used in another calculation. For example:

```
»45 *  pi / 180
ans =
    0.7854
»sin(ans)
ans =
    0.7071
```

Another operator is the *power* operator, ^, as illustrated below:

```
»5^3
ans =
   125
»5^(-3)
ans =
    0.0080
```

MATLAB has two versions of the *division* operator, one for *right division*

```
»6 / 3
ans =
     2
```

and one for *left division*

```
»3 \ 6
ans =
     2
```

When used with scalars, both operators yield the same results. However later when we discuss vectors and matrices we will see that they are not interchangeable.

When dealing with more complicated expressions you may want to use parentheses in the usual way. For example:

```
»3 * (sin(4) + sqrt(6)) / 12.5
ans =
    0.4062
```

If you type in a complex expression and make an error, you can edit the previous expression by pressing the ***Up Arrow*** ↑ to recall the last line. Press the ***Up Arrow*** ↑ or ***Down Arrow*** ↓ several times to recall other command lines in the *command line stack*.

MATLAB deals with complex numbers just as well as with real numbers. For example:

```
»sqrt(-5)
ans =
        0 + 2.2361i
»(2i + 3) * (3 - 5i)
ans =
  19.0000 - 9.0000i
»2 / (3 - 6i)
ans =
   0.1333 + 0.2667i
```

Some important points for operations and operators are given below:

- Spaces before and after operators are generally ignored by MATLAB and are often added for clarity.
- Parentheses can be freely used to indicate operation precedence. The general operator precedence is *multiplication* first, *division* second, *addition* third and *subtraction* last.

R.1.2 Constants and Variables

Any operation carried out in MATLAB can be given a *variable name* as shown in the following example:

```
»a = 6; b = 3; c = 4;
»d = a + b^3 - sin(c)
d =
   33.7568
```

Notice that the built–in variable `ans` has been replaced with the user defined variable `d`. Some important points to remember about MATLAB variables are:

- MATLAB variable names are *case sensitive*, hence $A \neq a$. Variable names can be up to 32 characters long.
- In addition to the standard keyboard characters, you can use the underscore, _, in a variable name. Hence variables names like `radius_of_curvature` and `_time` are perfectly valid.
- Semicolons at the ends of statements in MATLAB are optional, whereas in standard C they are required. In MATLAB, the semicolon serves *only* to inhibit the display of that particular operation on the screen.
- As in standard C, multiple statements can appear on a single line. These statements can be separated either by semicolons or by commas. Commas do not inhibit the output whereas semicolons do.
- There are only three types of variables in MATLAB: *real numbers* (equivalent to type `double` in standard C), *arrays of real numbers* (similar to standard C's arrays) and *character strings*. The standard C types like `int`, `float`, `double`, `unsigned int`, `char`, etc., *do not exist* in the MATLAB programming language. Hence even if you write `a = 6`, the variable `a` is a double precision real number, NOT an integer. In fact there are no true integers in MATLAB.

There are also some *built–in constants* in MATLAB as shown below:

```
»pi
ans =
   3.1416
»inf
ans =
   ∞
»eps
eps =
   2.2204e-16
»10*PI
ans =
   []
```

In the last example, since `pi` *is* defined but `PI` is *not* (remember, variable names are *case sensitive*), the result is null, indicated by `ans = []`. Be careful with these variables, because they can easily be renamed by mistake and their original value is then lost for the rest of that MATLAB session or until the `clear` command is issued.

R.1.3 Operators in MATLAB

All of the standard arithmetic operators are supported in MATLAB. A list of the standard operators is shown in the table below:

```
Char    Name                          HELP topic
   +      Plus                          arith
   -      Minus                         arith
   *      Matrix multiplication         arith
   .*     Array multiplication          arith
   ^      Matrix power                  arith
   .^     Array power                   arith
   \      Backslash or left division    slash
   /      Slash or right division       slash
   ./     Array division                slash
   kron   Kronecker tensor product      kron
   :      Colon                         colon
   ( )    Parentheses                   paren
   [ ]    Brackets                      paren
   .      Decimal point                 punct
   ..     Parent directory              punct
   ...    Continuation                  punct
   ,      Comma                         punct
   ;      Semicolon                     punct
   %      Comment                       punct
   !      Exclamation point             punct
   '      Transpose and quote           punct
   =      Assignment                    punct
   ==     Equality                      relop
   < >    Relational operators          relop
   &      Logical AND                   relop
   |      Logical OR                    relop
   ~      Logical NOT                   relop
   xor    Logical EXCLUSIVE OR          xor
```

Notice than many of these are either different from the corresponding operator in standard C or don't even exist in C. The `+, -` and `*` operators all work on matrices (as long as the sizes are correct) and complex numbers as well as scalars. In addition, the operators `.*`, `.^` and `./` are special array operators that allow element by element multiplication and division of equal size arrays. The backslash operator, `\`, invokes the built–in equation solver to solve systems of equations of the form A * x = b as in

```
»A = [1 -2 3 4;2 1 1 3;-1 0 1 3;2 2 1 5];        % 4 x 4 matrix
»b = [1 1;2 3;2 -1;1 3];                         % 4 x 2 matrix (two right
                                                 % hand side vectors, b1, b2)
»x = A \ b ;                                     % use built-in solver
x =
   -3.0000    0.8000                             % first column is for b1,
    9.0000    1.6000                             % second column is for b2
   14.0000    2.2000
   -5.0000   -0.8000
```

R.1.4 Built–in Math Functions in MATLAB

The elementary math functions are shown in the table below. Each of them works on matrices and complex numbers as well as scalars.

```
Trigonometic
sin          - Sine.
sinh         - Hyperbolic sine.                    Exponential.
asin         - Inverse sine.                       exp         - Exponential.
asinh        - Inverse hyperbolic sine.            log         - Natural logarithm.
cos          - Cosine.                             log10       - Common logarithm.
cosh         - Hyperbolic cosine.                  sqrt        - Square root.
acos         - Inverse cosine.
acosh        - Inverse hyperbolic cosine.          Complex.
tan          - Tangent.                            abs         - Absolute value.
tanh         - Hyperbolic tangent.                 angle       - Phase angle.
atan         - Inverse tangent.                    conj        - Complex conjugate.
atan2        - Four quadrant inverse tangent.      imag        - Complex imaginary part.
atanh        - Inverse hyperbolic tangent.         real        - Complex real part.
sec          - Secant.
sech         - Hyperbolic secant.                  Numeric.
asec         - Inverse secant.                     fix         - Round towards zero.
asech        - Inverse hyperbolic secant.          floor       - Round towards minus infinity.
csc          - Cosecant.                           ceil        - Round towards plus infinity.
csch         - Hyperbolic cosecant.                round       - Round towards nearest integer.
acsc         - Inverse cosecant.                   rem         - Remainder after division.
acsch        - Inverse hyperbolic cosecant.        sign        - Signum function.
cot          - Cotangent.
coth         - Hyperbolic cotangent.
acot         - Inverse cotangent.
acoth        - Inverse hyperbolic cotangent.
```

Notice that many of the names are either different from their C counterparts or don't even exist in C.

R.1.5 Formatted Output

If you have been observant, you have noticed that all results have been *displayed as* either integers or real numbers with four decimal places. Displaying results with four decimal places is the default format in MATLAB, but this display format can be easily changed, as shown below:

```
»clear
»pi
ans =
   3.1416
»format long
»pi
ans =
   3.14159265358979
»format long e
»pi
ans =
   3.141592653589793e+00
»format rat
»pi
ans =
   355/113
```

In the sequence above, notice that first command issued was `clear` . This command removes all

previously defined user variable names and their values from memory; essentially like restarting MATLAB. The default format is `format short` and the first value of `pi` is displayed in this format. the default format. The other examples illustrate `format long e` , the long exponential format and `format rat` , which gives a small integer rational approximation (i.e., ratio of integers) for any MATLAB variable.

R.1.6 The Diary Command

Calculations performed in MATLAB can be recorded in a simple way by using the `diary` command. The command `diary file_name` causes a copy of all subsequent terminal input and most of the resulting output to be written to the named file. The commands `diary off` and `diary on` turn the diary function on and off so that you can selectively send output to the same file, as shown in the following example:

```
»clear                    % clear variables and recover memory
»clc                      % clear the command screen
»diary c:\temp\cylinder.txt
»r = 0.1;                 % radius of the cylinder base
»h = 2;                   % height of the cylinder
»S = 2 * pi * r * h       % lateral surface area of the cylinder
S =
    1.2566
»V = pi * r^2 * h         % volume of the cylinder
V =
    0.0628
»diary off
```

Everything between the command `diary cylinder.txt` and the command `diary off` is sent to the file `cylinder.txt` on the `C: Drive` in the subdirectory `temp` . Some important points associated with the above code are:

- The MATLAB *comment specifier* , `%`, was used in the above example. Anything on a given line following the symbol `%` is ignored by MATLAB as a comment. In standard C each comment must have a beginning specifier `/*` and an ending specifier `*/`. In MATLAB the ending specifier is the end of a line.
- You should get in the habit of using comments in MATLAB in order to *document* your work.
- Some statements have semicolons, some do not. For ones that don't, the output is immediately displayed on the screen; for ones that do, the output to the screen is inhibited.

The `type` command can be used to display any ASCII text file on the screen. Hence you can check the diary file generated above as follows:

```
»type cylinder.txt
```

and you will see the following display:

```
r = 0.1;                  % radius of the cylinder base
h = 2;                    % height of the cylinder
S = 2 * pi * r * h        % lateral surface area of the cylinder
S =
    1.2566
```

```
V = pi * r^2 * h          % volume of the cylinder
V =
    0.0628
diary off
```

R.1.7 One Dimensional Arrays and Plotting

Unlike C, MATLAB has sophisticated plotting capabilities. These built–in `plot` command has several forms. The most commonly encountered from is `plot(x,y)` , which plots vector `x` versus vector `y`, if both have the same length. Various line types, plot symbols and colors may be obtained with `plot(x, y, s)` where `s` is a 1, 2 or 3 character string made up of the following characters:

symbol	colors	symbol	data point and line styles
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

For example, `plot(x,y,'c+')` plots a cyan plus at each data point, while `plot(x,y,'y-',x,y,'go')` plots the data twice, with a solid yellow line interpolating green circles at the data points. In the following example:

```
»x = [-1 0 1 2 3 4 5];
»y = sin(x)
y =
   -0.8415         0    0.8415    0.9093    0.1411   -0.7568   -0.9589
»plot(x,y)
```

the `plot(x,y)` command produces the output shown in Figure 1.1, in a separate window:

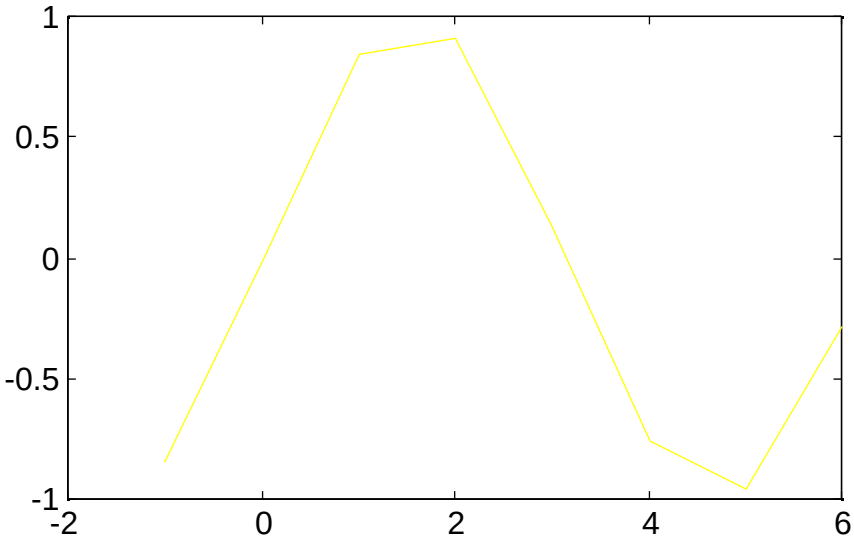

Figure 1.1

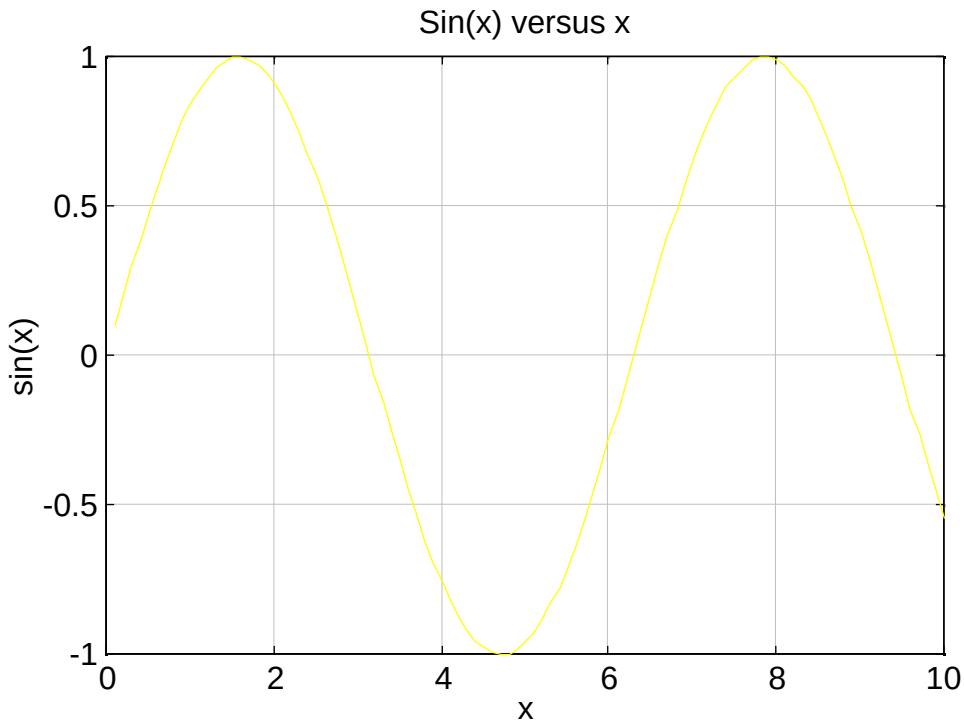

Figure 1.2

In the example above, `x` is an array (a vector) and we see that the built–in function `sin` can operate on arrays (standard C math functions can only operate on scalars). The statement y = sin(x); creates a new array `y` with the same length as `x`. The elements of the vector `y` are the sines of each of the elements of the `x` array. A more sophisticated example would be

```
»clear
```

```
»for i = 1 : 100, x(i) = i / 10; y(i) =     sin(x(i)); end
»plot(x,y),  grid, title('Sin(x) versus x'),    xlabel('x'),  ylabel('sin(x)')
```

In this example we see that MATLAB has some programing structures (in this case the `for` loop) very much like any other programming language. These commands produce the plot shown in Figure 1.2. The examples above give only a brief indication of what can be done with MATLAB's plotting features. For more examples, take a look at the `demo` programs.

R.2 Storing Files Created by MATLAB

Normally when you save any file created by the Notepad in conjunction with MATLAB, the file is stored in a subdirectory associated with the MATLAB program. However, when you are on the campus Networks, these subdirectories are not accessible and the files are stored in whatever drive (possible a Network drive) and directory from which the program was executed.

In order to minimize problems with MATLAB finding the files you create, issue the following command from the MATLAB command window:

```
»cd c:\temp
```

and make sure that all files created by MATLAB and by Notepad are saved in this directory. The network version of MATLAB has been configured so that it can *always* find `m-files` in the `temp` directory. You can always check to see what files MATLAB can see if you issue the following command from the MATLAB command window:

```
»dir
```

or

```
»dir *.m
```

When you have finished your MATLAB session, make sure you copy any files you have created from `c:\temp` to your own floppy disk and/or to your disk space on the `m: drive`. The `temp` directory is local to the PC you are using and this directory is regularly erased, so the next time you log onto the network, your files will probably be gone.

R.3 Printing MATLAB Plots

When you are using a non–network version of MATLAB you can send any plots created directly to your printer. However, on the network you cannot do this. To get a hard copy of your plots on the network, do the following:

- From the **Program Manager** open the **Utilities** group and double click on **MSDOS Prompt**.
- In the **Dos** window type `help`
- In the **Help** window select **Applications**
- In the **Applications** window select **MATLAB**
- In the **MATLAB** window select **Plotting**
- In the **Plotting** window, follow the instructions given for plotting

The instructions are network and operating system dependent and may change with future network modifications and operating system upgrades. Hopefully these instructions will be updated as well.

R.4 Writing Programs and Functions in MATLAB

In the examples above, each command was entered directly from the MATLAB command line environment. Although this is convenient for small sets of commands that are not going to be repeated, if we wish to regularly use the same sequence of commands with different input and output, then we need to write a MATLAB *program.* MATLAB has a built–in editor (in the case of Windows 3.1 and below it uses the ***Notepad*** editor as the built–in editor) which can be used to write sequences of MATLAB commands to a file. Such files are called *script files* (or *m–files*) and they must have a file name extension of `.m`.

To create a new empty file, invoke the ***File Menu*** and select ***New*** and type in the sequence of commands. Once the file is created and saved with the appropriate extension, it can be run from *within* the MATLAB environment by typing its file name (without the `.m` extension) from the command line.

An important part of understanding MATLAB programming is the knowledge of where MATLAB stores the files you create. By default, the files created with the editor are saved in the directory or folder from which MATLAB was invoked (see Section R.2 for network variations). For example if we create a file with the name `hello.m` in one MATLAB session and try to execute in a later session, we might get the error message

```
»hello
??? Undefined function or variable hello.
```

This means that MATLAB cannot find the file `hello.m` . A likely reason for this error message is that you started the MATLAB session from a different directory than you did the previous time.

The MATLAB *path command* can be used to tell MATLAB the location of the file `hello.m` . The following examples of the *path command* change the MATLAB path to whatever is in the quotes:

```
path('C:\MATLAB\MYCODE\')           % DOS or Windows
path('Big Mo:MATLAB:My Stuff')      % Mac
```

In order to append new path to the default path, you would issue the sequence of commands:

```
p = path                                % returns a string containing the path
                                        % in p.
path(p, 'C:\MATLAB\MYCODE\')            % adds the new path to the old path
                                        % in DOS or Windows
path(p, 'Big Mo:MATLAB:My Stuff')       % adds the new path to the old path
                                        % on a Mac
```

R.4.1 Input and Output in MATLAB

MATLAB's input and output (I/O) commands are generally much easier to use than those in standard C. The built–in function `input` can be used to enter data directly from the screen and the `disp` function can be used to output text and data to the screen. To illustrate, we create a small *m–file* called `hello.m` with the following statements in it:

```
% The program in the file HELLO.M greets you and asks for your name. Then
% it greets you by name and tells you the date.

disp('Hello! What is your first name?')
name =  input('Please enter your name','s');
```

```
d = date;
response = ['Hello ', name, '. Today is ', d, '.'];
disp(response)
```

and this is what is displayed when you run the program:

```
»hello                                  % this runs the program
Hello! What is your first name?         % the first line displayed
Please enter your name    Bob           % Bob was typed in
Hello Bob. Today is 20-Jan-96.          % the greeting
```

Here is what you get when you ask for help for this program:

```
»help hello
   The program in the file HELLO.M greets you and asks for your name. Then
   it greets you by name and tells you the date.
```

The `help` function in MATLAB simply displays the first set of contiguous comment lines in the file (and no others). Hence you should always make the first few lines of a file be comment lines that describe its workings.

The `disp` function can be used to display any string. The `input` function displays a prompt string (in this case, `'Please enter your name'`) and saves whatever is entered into the variable to the left of the equal sign. The optional second argument of `input` is `'s'`, which tells the function to expect a string input (as opposed to a numerical input). The next statement stores the date in `d`. The next to last statement shows how strings are concatenated in MATLAB. A string is simply an array of characters (a vector) and the variable `response` is a new string put together using the variables `name` and `d`. Finally the last statement actually displays the string `response` . These last two statements could have been written as one using

```
disp(['Hello ' name '. Today is ' d '.']);
```

Notice that the commas in the original variable `response` were all optional.

Another output function in MATLAB is `fprintf` , which has exactly the same syntax as `printf` does in standard C (standard C's `printf` does not exist in MATLAB, only `fprintf`). This function will be discussed in Section R.5.4.

R.4.2 Functions in MATLAB

Another kind of script file is the *function m–file*. *Function m–files* can be used to extend the capabilities of MATLAB to include user defined functions. Hence MATLAB is infinitely extendable simply by adding additional scripts. In fact, many of the functions already in MATLAB are nothing more than script files. If you know where to look, you can examine these files using the built–in editor. By the way, this is an excellent method of learning *how* to write more sophisticated MATLAB functions.

A *function m–file* differs from other *m–files* in that the very first non–comment word in the file ***must*** be the keyword `function` . The general syntax for a function statement is

```
function [out1, out2, ...] = name(in1, in2, ...)
```

The word `function` is the required keyword mentioned above. The variables `out1` , `out2` , etc. are the names of any output variables that are computed and `name` is the name of the function. The name of the function must be the same as the name of the *m–file* it is stored in (without the `.m`). Finally the variables `in1`, `in2`, etc. are the names of any input variables, i.e., variables that the

function needs to compute its results. The input list of variables can be empty, in which case the parentheses are not needed. The output list can be a single variable, in which case the brackets are not needed.

All of the built–in trigonometric functions in MATLAB require the argument be a number in radians. Since we often need to input the argument in degrees, we can write our own function to do this. In the example below, we create a new function called `sind` , hence the *m–file* name must be `sind.m` :

```
function s = sind(a)

% SIND: This function computes the sine of the angle, a, when the
% angle is expressed in degrees rather than radians.
% INPUT: a, the angle in degrees
% OUTPUT: s, the sine of that angle

a = a * pi / 180;      % convert a to radians
s = sin(a);            % compute the sine of a
```

The first line is the required function definition, indicating the function name (`sind`), its input variables (just `a` in this case) and it output variables (just `s` in this case). The first four comment lines are displayed by `help` and the last two lines are self–explanatory. The following sequence shows how the function is called:

```
»sind(15)
ans =
    0.2588
```

or

```
»x = 45;
»y = sind(x)
y =
   0.7071
```

With this approach, you can now create similar *m–files* for all the other trig functions to enhance MATLAB's capabilities to match you needs.

R.5 MATLAB Control Structures

MATLAB has most of the repetitive and logical control structures associated with standard C, albeit with some modifications. Although all of these control structures can be used directly in the command line environment, they are most effectively used within MATLAB programs and functions.

R.5.1 The `for` Loop

A `for` loop is a control structure used to repeat a statement or a block of statements a fixed number of times. The general syntax is:

```
for variable = v_start: increment : v_end
   statements
   ...
end
```

where `variable` is any valid variable name, `v_start` in the initial value of `variable`, `increment` is the amount by which `variable` is to be increment each time through the loop, `v_end` indicates the value of `variable` at which the loop should end, and the keyword `end` indicates the end of the loop. An abbreviated form of the syntax is:

```
for variable = v_start : v_end
   statements
   ...
end
```

When no `increment` is specified, the default value of 1 is used. If `v_start` is larger then `v_end`, then `increment` can be negative. For example, if we want to compute the square root of the first 100 integers, we can write:

```
for i = 1 : 100                  % default increment is 1
   y(i) = sqrt(i);               % store the result in the array y
end                              % required end statement
```

If we wanted to find the natural log of all the numbers from 0.1 to 1 in steps of 0.2, we can write:

```
i = 0;                           % initialize index counter
for k = 0.1 : 0.2 : 1            % increment is 0.2 for k
   i = i + 1;                    % increment the index i
   z(i) = log(k);                % store the result in the array z
end                              % required end statement
```

Notice that `log(1)` is never computed in this case, since the values of `k` are `0.1, 0.3, 0.5, 0.7`, and `0.9`. The next value of `k` is `1.1` which is beyond `v_end = 1`, hence the loop stops when `k = 0.9`. The use of the additional counter variable `i` is important here, since the index of an array *must be* an integer starting at 1 (not 0, like standard C).

It turns out, however, that there is a much easier way to perform many of these kinds of loops in MATLAB. The last example can be written in the compact form:

```
»v = 0.1 : 0.2 :1;      % create a vector going from 0.1 to 1 in steps of 0.2
»w = log(v);            % create a vector whose elements are the logs of v
```

and this results in

```
v =   0.1000    0.3000    0.5000    0.7000    0.9000
w = -2.3026   -1.2040   -0.6931   -0.3567   -0.1054
```

R.5.2 The `if` Statement

The `if` statement is a conditional structure whose general syntax is:

```
if condition
   statements
   ...
elseif condition        % notice that elseif is one word
   statements
   ...
else
   statements
   ...
end
```

Both `elseif` and `else` are optional, but the `end` statement is required. The `condition` is a statement using one or more of the following logical operators in MATLAB:

```
>      - greater than
<      - less than
>=     - greater than or equal to
<=     - less than or equal to
==     - logical equal to
~=     - not equal to (different than standard C)
&      - the  and  operator (different than standard C)
~      - the  not  operator (different than standard C)
|      - the  or  operator (different than standard C)
xor    - the  xor  operator (different than standard C)
```

The `if` loop is executed only if `condition` is true. Just like standard C, any `condition` that evaluates to zero in MATLAB is `false` and if `condition` evaluates to anything other than zero, it is considered to be `true`. Some of these will be illustrated in the example in Section R.5.3.

R.5.3 The `while` Loop

The while loop is another repetitive structure which repeats a statement or set of statements until a specified logical condition is met. The general syntax is:

```
while condition
   statements
   ...
end
```

Just like the `if` statement, the statements in the `while` loop are executed only if `condition` is true. As an example of both the `if` and `while` statements, let's say we wanted to write a MATLAB function to perform the equivalent of integer division. Since all MATLAB numbers are real and not integers, this is non–trivial in MATLAB. In this problem we are given a dividend, x, and a divisor, y, and we are looking for an integer quotient q, and a remainder r such that x = qy + r. This will be accomplished by writing a simple (translation, not very elegant) function called `divide`, which will be stored in the *m–file* `divide.m`. If might be done in the following way:

```
function [q, r] = divide(x, y)

% DIVIDE(x,y): This function performs integer division of x by y.
% The operation is carried out by subtracting y from x until
% x < y. The function returns the quotient q and the remainder r.
% See also REM(x,y)

q = 0; c = 1; k = 1;

if y == 0
   error('...division by zero in DIVIDE...')
end
if (x == 0) | (x < y)         % | is the or operator
   q = 0; r = x;
end
```

```
if x < 0
   x = -x; c = -c; k = -k;
end
if y < 0
   y = -y; c = -c;
end
while x >= y
   x = x - y; q = q + 1;
end

q = c * q; r = k * x;            % final values of q and r
```

Several ways of using this function are shown below:

```
»divide(7,3)
ans =
     2                        % quotient only
»v = divide(7,3)
v =
     2                        % quotient only
»[x,y] = divide(7,3)
x =
     2                        % quotient in x
y =
     1                        % remainder in y
»help divide

% DIVIDE(x,y): This function performs integer division of x by y.
% The operation is carried out by subtracting y from x until
% x < y. The function returns the quotient q and the remainder r.
% See also REM(x,y)

»divide(7,0)
??? Error using ==> divide       % built-in error message
...division by zero...           % function error message
```

A couple of important things should be pointed out about this function. First, the function returns two quantities, `q` and `r`. Standard C functions can only return one quantity. Second, a new built–in function called `error` is used in the first `if` statement. This function prints a system error message, a function error message (whatever's between the quotes) and immediately returns to the command line environment.

R.5.4 Reading and Writing Data

There are several ways to input data into MATLAB. For example the data could be assigned to a vector or matrix array by typing the data in at the command line, or as a part of a program or function. However if the data set is large, this can be very tedious. We can also read data into MATLAB from an *external text file*. For example, let's say that a laboratory instrument has been used to measure a sequence of 1000 voltages representing the temperature of a process

and the results have been saved in a text file called `voltdata.txt` . We want to take these voltages and convert them to temperatures in MATLAB using the instruments calibration data. This is accomplished in MATLAB very easily by issuing the command:

```
load voltdata.txt -ascii
```

The data will be stored in the new array variable `voltdata` , created by the load command. The argument `–ascii` indicates that the file `voltdata.txt` is a standard ASCII text file (i.e., not a binary file). The calibration data can then be used to convert the voltages to temperatures.

We can also write data *to* an external file by means of the `fprintf` function. The general syntax of this function takes the two forms. In the first form

```
fprintf('format string', var1, var2, ...)
```

the output goes to the screen. The 'format_string' uses standard C formatting conventions and will not be discussed here. Of course each format specifier in `'format_ string'` must have a corresponding variable to print and these are the variables `var1, var2, ...` in the expression above. In the second form

```
fprintf('file_name_string', 'format_string', var1, var2, ...)
```

the output goes to the file named in the string `'file_name_string'` . If that file does not exist, it is created and if it does exist, the data is appended to it. For example if we wanted to store the temperatures from the temperature array `temp,` computed from the above voltages, into a file called `tempdata.txt` , then we might use the following sequence of commands:

```
file_name = 'tempdata.txt';
fprintf(file_name, '       Voltage    Temperature\n')
fprintf(file_name, '============      ============\n')
for i = 1 : 1000
   fprintf(file_name, '%12.4f     %12.2f\n', voltdata(i), temp(i))
end
```

We can examine the data in this newly created file by issuing the command

```
»type tempdata.txt
```

from the command line. This results in

```
      Voltage    Temperature
============    ============
      -3.2146          256.36
      -2.4715          310.12
          ...             ...
       3.1792          656.37
```

One major difference between the MATLAB version of `fprintf` and the standard C version is that in the MATLAB version, single quotes are used around strings and in the standard C version, double quotes are used.

R.6 Vectors and Matrices

Vectors in MATLAB can be either row vectors or column vectors, with row vectors being the default. The *transpose operator*, `'`, converts a row vector into a column vector and vice–versa. A collection of vectors and vector operations are shown below:

```
»a = [1, 2, 3, 4, 5]          % columns can be separated by commas
```

```
a =
     1     2     3     4     5
»b = [2 3 4 5 6]              % or columns can be separated by spaces
b =
     2     3     4     5     6
»c = [1;2;3;4;5]              % rows separated by semicolons
c =
     1
     2
     3
     4
     5
»d = [2
      3
      4
      5
      6]                      % rows separated by carriage returns
d =
     2
     3
     4
     5
     6
»a * 3                        % multiply a vector by a scalar
ans =
     3     6     9    12    15
»c / 2                        % divide a vector by a scalar
ans =
    0.5000
    1.0000
    1.5000
    2.0000
    2.5000
»v = dot(a, b)                % dot product between a and b
v =
    70
»d(3)                         % display the third element of vector d
ans =
     4
»[m,n] = size(c)              % compute the size of a vector (or matrix)
m =
     5                        % c has 5 rows
n =
     1                        % c has 1 column
»b == d'                      % logical comparison of b and d transpose
ans =
```

```
     1     1     1     1     1
```

The ones in the last example indicate a logical true condition, i.e., each element in `b` equals the corresponding element in `d` transpose.

Matrices can be thought of as columns of column vectors or rows of row vectors. Some examples of matrices and matrix operations are shown below:

```
»A = [1 2 3 4; 2 3 4 5; 3 4 5 6]     % spaces separate columns, semicolon rows
A =
     1     2     3     4
     2     3     4     5
     3     4     5     6
»A'                                  % compute the transpose of A
ans =
     1     2     3
     2     3     4
     3     4     5
     4     5     6
»A^(-1)                              % compute the inverse of A
??? Error using ==> ^
Matrix must be square.

»mean(A)                          % compute the mean (of each column) of A
ans =
     2     3     4     5
»trace(A)                         % compute the trace of A
ans =
     9
»norm(A)                          % compute the norm of A
ans =
   13.0112
```

Consider the problem of finding the solution to the system of equations

$$\begin{aligned} 3x - 2y + z &= 0 \\ -2x + 3z + u - v &= 2 \\ 2y + z - 4u + 5 &= 0 \\ x + y + z + u - 6 &= -v \\ u + 6v &= 3 \end{aligned} .$$

This system can be written in the matrix/vector form

$$\begin{bmatrix} 3 & -2 & 1 & 0 & 0 \\ -2 & 0 & 3 & 1 & -1 \\ 0 & 2 & 1 & -4 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ -5 \\ 6 \\ 3 \end{bmatrix} \Leftrightarrow \mathbf{Ax} = \mathbf{b},$$

and we can easily solve for the unknown vector **x** in MATLAB as follows:

```
»A = [  3 -2   1   0  0           % separate columns by spaces and
       -2  0   3   1 -1           % rows by carriage returns
        0   2  1 -4   0
        1   1  1   1  1
        0   0  0   1  6]
A =
     3    -2     1     0     0
    -2     0     3     1    -1
     0     2     1    -4     0
     1     1     1     1     1
     0     0     0     1     6
»b = [0; 2; -5; 6; 3]
b =
     0
     2
    -5
     6
     3
»x = A \ b          % the left division operator is used to solve systems
x =                 % the solution vector
    1.0468
    1.8799
    0.6193
    2.3448
    0.1092
»A * x              % check to see if A * x is equal to b
ans =
     0
     2
    -5
     6
     3
```

These examples illustrate some of the basic operations that can be used on vectors and matrices in MATLAB. Examine the help files and demos to see others.