MATLAB FOR STATISTICS

David A Stephens

March 4, 2004

[0]With acknowledgements to Dr Rob Beardmore, Department of Mathematics.

1 INTRODUCTION TO MATLAB

1.1 THE MATLAB ENVIRONMENT

To make a directory into which you should put your Matlab work, type

```
mkdir MyMatlabDirectory
cd MyMatlabDirectory
```

Typing

```
ls
```
or
```
dir
```

lists the files available in the current directory. Typing

```
cd
```

returns the current directory. Typing

```
quit
```

closes the Matlab window and finishes the session.

1.2 BASIC COMMANDS

Matlab views most objects as vectors, even functions are generally viewed in the same way as vectors! For example, the following Matlab code plots the sin function

```
1.  x = 0:pi/100:2*pi;
2.  y = sin(x);
3.  plot(x,y);
4.  xlabel('x = 0:\pi');
5.  ylabel('Sine of x');
6.  title('Plot of the Sine Function','FontSize',12);
```

Having entered these commands (although not the numbers), type

```
whos
```

followed by enter. This shows information regarding the variables that are kept for later use by Matlab.

Now type

```
x
```

followed by enter. This displays the elements of the vector `x`, although they disappear from the screen rather quickly; this is because `x` is a *row vector*. So now type

```
x'
```

and this displays the vector `x` in a *column* format.

To explain how the vector `x` is defined, consider again the Matlab expression

```
x = 0:pi/100:2*pi;
```

This says that `x` is defined to be the vector which has 0 in its first element, $\pi/100$ in its second, $2\pi/100$ in its third, and so on, until you reach 2π in the final element of `x`. So, the command

```
z = 0:0.5:10;
```

defines the vector $\mathtt{z} = [0, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 2\frac{1}{2}, 3, ..., 10]$. You can manipulate vectors just as you would numbers, so type

```
w = sin(z);
```

This defines the vector

$$\mathtt{w} = [\sin(0), \sin(\frac{1}{2}), \sin(1), \sin(1\frac{1}{2}), \sin(2), \sin(2\frac{1}{2}), \sin(3), ..., \sin(10)].$$

Now type

```
plot(x,sin(x),'-k',z,sin(z),'xr');
```

This is a *multiple plot* showing the original sine curve using a black line and the new dataset $\{(z_i, \sin(z_i))\}$, where z_i is an element of the vector `z`. The vector `z` differs from the set $Z = \{0, \frac{1}{2}, ..., 10\}$ as the elements of the set Z are in no particular order. In terms of notation, either square or round brackets are used to denote vectors, whereas curly braces are used for sets.

Another way of defining a vector is to use square brackets and write, for instance,

```
u = [-1 2 0 1 2 0];
```

1.3 MORE ON PLOTTING FUNCTIONS

Another method for plotting functions is to use the

```
ezplot
```

command. If you type

```
help ezplot
```

you will see some examples of how to use this command. For instance,

```
ezplot('sin(x)',[0,2*pi]);
```

does the same job as the set of commands we issued above

1.4 IN-BUILT FUNCTIONS

Matlab has the usual array of mathematical operations and elementary functions and their inverses such as

$$\exp, \cos, \sin, \ln, ...;$$

type

```
help elfun;
```

to find out about these.

1.5 MATRICES AND VECTORS

You have seen in your first-year linear algebra course that matrices are objects that act on vectors via a multiplication operation that is written

$$\mathtt{M}\ \mathtt{x},$$

where `M` is a matrix and `x` is a vector. In the case of 2×2 matrices, this is defined by

$$\begin{bmatrix} ax+by \\ cx+dy \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \mathtt{M}\ \mathtt{x}.$$

In order to define matrices in Matlab, we use an extension of the vector notation. Thus, to define the matrix

$$\mathtt{M} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

in Matlab, enter

```
M = [[1 1]' [0 1]']'
```

The apostrophe is an operator which takes a column vector and returns a row vector, and vice-versa. Thus, `[1 1]' =` $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 1 \end{bmatrix}'$ `= [1 1].` This operation is called the vector *transpose*, and its mathematical symbol is a superscript T.

We can operate on the matrix `M` directly, for instance typing

```
M^2
```

returns the matrix

$$\mathtt{M}^2 = \mathtt{M} \cdot \mathtt{M} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}.$$

It is also possible to apply some elementary mathematical functions to matrices, for instance you can evaluate the matrix exponential

$$\mathtt{expm}(\mathtt{M}) = \sum_{n=0}^{\infty} \frac{1}{n!} M^n,$$

which returns

$$\mathtt{expm}(\mathtt{M}) = \begin{bmatrix} 2.1783 & 2.1783 \\ 0 & 2.1783 \end{bmatrix}.$$

You should also evaluate

$$\mathtt{logm}(\mathtt{M}), \mathtt{sqrtm}(\mathtt{M}),$$

noting that

$$\mathtt{expm}(\mathtt{logm}(\mathtt{M})) = \mathtt{M} \text{ and } \mathtt{sqrtm}(\mathtt{M}) * \mathtt{sqrtm}(\mathtt{M}) = \mathtt{M}.$$

It is actually possible to apply the elementary mathematical operations in a *pointwise* sense, which simply means that the function is applied to each of the elements in turn, viz:

$$\exp(\mathtt{M}) = \begin{bmatrix} 2.7183 & 2.7183 \\ 1.0000 & 2.7183 \end{bmatrix};$$

and analogous commands work for other functions too. Matlab calls these operations *array operations* as distinct from the mathematical or algebraic operations that we can perform on matrices. In this context, *array* is a synonym for the term *pointwise* which we shall discuss in due course.

The command to obtain the determinant of a matrix `M` is

$$\mathtt{det(M)},$$

and to multiply two matrices together one uses the asterisk notation

$$\mathtt{A * B}.$$

Remark 1.1 *One can also enter matrices using a command like*

$$\mathtt{M = [1\ 1; 0\ 1]},$$

use whichever one you prefer.

1.6 LINEAR EQUATIONS

Let us now briefly discuss the solution of a linear system of equations. These are equations can be written in the form

$$\begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix},$$

which, using a single matrix-vector multiplication can be written in the form

$$Ax = b. \tag{1.1}$$

In order to find the solution of this equation, which is known to exist and be unique if $\det(A) \neq 0$, then we might think it reasonable to form the inverse matrix, B, of A:

$$AB = I, \tag{1.2}$$

and then set

$$y = Bb.$$

It then follows that $Ay = ABb = b$.

This procedure will work and can be done in Matlab using the command

$$\texttt{B} = \texttt{inv(A)}.$$

However, you may realise that equation (1.2), which is a single matrix equation, represents n vector equations of the form (1.1). This means that we are doing far more work than is necessary to solve the original problem (1.1).

For this reason, in order to solve (1.1) one should actually use the

slash

command. This is written using a backslash, \, so the actual Matlab command you should use is

$$\texttt{y} = \texttt{A}\backslash\texttt{b}.$$

Example 1.1 *In Matlab, enter the commands*

```
A = [[1 2 3]' [2 3 4]' [4 5 6]']'
(or A = [1  2  3; 2  3  4; 4  5  6])
b = [1 3 9]'
```

Then type $\texttt{y} = \texttt{A}\backslash\texttt{b}$*, this should give an* approximate *solution of the equation* $Ax = b$*, which is written in terms of coordinates as*

$$\begin{aligned} x + 2y + 3z &= 1, \\ 2x + 3y + 4z &= 3, \\ 4x + 5y + 6z &= 9. \end{aligned}$$

It actually returns a warning that the matrix `A` *is singular, which means that it has no inverse. If you now type*

$$\texttt{det(A)}$$

you will see why.

Now type

```
A(1,1) = 2;
det(A)
```

and this returns the value -2*. Now, typing*

```
x = A\ b;
```

returns the vector `[1 7 -5]'`*. And*

```
A * x
```

returns the vector `b`*, as expected.*

1.7 THE DOT AND COLON OPERATORS

Example 1.2 *In order to evaluate the integral*

$$\text{erf}(z) = \int_0^z e^{-x^2} dx$$

in Matlab, one uses the Matlab functions **quad** *and* **quadl**.

If you type **help quad** *then you should get help on numerical integration, however the command*

```
quad('exp(-x^2)',0,2)
```

will not evaluate the integral

$$\int_0^2 \exp(-x^2)dx.$$

In order to evaluate this integral it is necessary to put a "point" in the previous command:

```
quad('exp(-x.^2)',0,2)
```

which will work!

Why is this?

The reason is that the multiplication operator for matrices is the asterisk $*$, and to denote an operation that works *pointwise*, we use a point in the notation. Thus, due to the rules of matrix-vector multiplication, the command

$$[1\ 2\ 3] * [2\ 3\ 1]$$

makes no sense, whereas in Matlab the pointwise command

$$[1\ 2\ 3]. * [2\ 3\ 1] = [2\ 6\ 3]$$

does make sense.

This reasoning also applies to the power operator ^ which operates on matrices, so that for any vector or matrix,

```
x.^2
```

represents the pointwise product of `x` with itself:

$$[1\ 2\ 3].^2 = [1\ 4\ 9].$$

Therefore, for any given vector `x`, the syntax

```
exp(x.^2)
```

forms the pointwise square of `x`, which is then exponentiated pointwise, whereas

```
exp(x^2)
```

makes no sense because the square of a vector is not defined.

We shall see more of this in the next section where we define functions in Matlab.

The colon operator is fundamental to the way Matlab works, but its use will only become clear as you become more experienced in using the package. Suffice to say, for now at least, that the colon is a way of obtaining a subset of a vector or matrix. For instance, if we define the vector

`z =` $[1:3:40]$

so that

$$\texttt{z} = [1\ 4\ 7\ 10\ 13\ 16\ 19\ 22\ 25\ 28\ 31\ 34\ 37\ 40],$$

then

$$\texttt{z}(2:5) = [4\ 7\ 10\ 13],$$

and

$$\texttt{z}(3:8) = [7\ 10\ 13\ 16\ 19\ 22].$$

These examples also show us that to obtain elements of arrays, matrices and vectors, we must use parentheses (round brackets). So that if we issue the command

```
A = [[1 2]' [3 4]']'
```

then

$$A(1,1) = 1, A(1,2) = 2, A(2,1) = 3, A(2,2) = 4.$$

Now, we can also use the colon operator in this context to extract rows and columns from the matrix thus.

```
A(1,:)  = [1 2]
```

and

```
A(:,1) = [1 3]'
```

and so on. Using the same logic,

```
A(:,:)  = A
```

2 MATLAB FUNCTIONS

2.1 M FILES

"M files" are the way in which we record sequences of Matlab operations that we want to perform several times. In order to create an M-file, go to the menu and choose the option

```
file\New\M-file
```

which will bring up a new window. In the window type

```
% This is an mfile.
```

Now execute the menu command

```
file\Save
```

and at the prompt type

```
test.m
```

Return to the command window and type

```
help test
```

and you should see the message. Now, in the m-file type the commands

```
A = [[1 2]' [3 4]']
inv(A)
A*inv(A)
```

and save the mfile via the menu command `file\Save`

Go to the Matlab *command window* and type `test`

You will now see a 2×2 matrix, its inverse and an identity matrix.

2.2 PASSING PARAMETERS

In order to make the m-file more flexible, we can pass parameters into it and define a matlab function. So, go to the top line of the file `test.m` and add the line

```
function test(a)
```

and alter the file so that it now reads

```
function test(a)
A = [1 2; 3 a]
inv(A)
A*inv(A)
```

Back in the Matlab command window, typing `test` will force an error message, but entering

```
function test(1)
```

or

```
function test(2.4)
```

will return a matrix and its inverse.

2.3 RETURNING VARIABLES

Suppose that we want to return the determinant of the matrix which we might like to use later on in our work. We then change the first line of `test.m` to read

```
function d = test(a)
```

and by adding the last line

```
d = det(A)
```

to the m-file, we can return the determinant. So, go to the command window and type

```
d = test(10);
```

and then type

```
whos
```

and you will see that a variable `d` has been defined.

2.4 FOR LOOPS

If you want to repeat a particular set of commands several times, and you know in advance how many times to repeat them, then you should use a `for` loop. These are used as per the following example which defines a particular matrix `A`:

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

2.5 IF STATEMENTS

Sometimes it is possible to avoid loops and if statements using Matlab's *vectorisation* properties. By way of an example, take the following code.

```
n = 10;
for i = 1:n,
     r = rand;
     if (r>0.5),
          v(i) = 2;
     elseif (r <= 0.5)
          v(i) = 0.5;
     end
end
```

This codes defines a random vector `v` of length 10 by creating a sequence of random numbers denoted by `r`, and the vector `v` satisfies `v(i) = 2 if 1/2` $<$ `r(i)` $\leq$ `1` and `v(i) = 1/2 if 0` $\leq$ `r(i)` $\leq$ `1/2.`

In order to form the logical expression evaluated in an *if* statement, the logical operators

$$\texttt{and}, \texttt{or}, \texttt{not}, \texttt{xor}, \texttt{eq}, \texttt{ne}, \texttt{lt}, \texttt{gt}, \texttt{le}, \texttt{ge}$$

will be useful. Type `help` followed by one of these commands to find out what they do.

A more succinct way of achieving this is through the following commands which avoids the use of loops and of `if` statements:

```
n = 10;
r = rand(1,n);
v = 2*(r(:)> 0.5) + (r(:)< 0.5)*0.5;
```

You should note from this that the Matlab statements

$$\texttt{r} = \texttt{rand}(1, \texttt{n});\ \texttt{T} = \texttt{r}(:) > 0.5$$

define a vector `T` which is either zero or one in each position according to the following rule: `T(i) = 1` if `r(i) > 0.5` and `T(i)=0` otherwise.

When you get used to it, it is always more desirable from an efficiency point of view to write for loops and if statements as operations on vectors, just as shown in the above example. From the point of view of saving memory, it can be beneficial to use `for`-type loops as these do not necessarily require the memory allocation to be done in advance in the way vectors do.

2.6 WHILE STATEMENTS

While statements provide a way of effecting a loop when you do not know in advance how many iterations of a loop will be needed to effect a particular operation, and the way a loop terminates is given by a logical expression (that is, an expression which evaluates to zero or one representing false or true). One can think of a for loop as a special case of a while statement.

Let us consider an example.

```
error = 1;
tolerance = 1e-5;
x = 0.5;
while (error > tolerance)
```

```
        X = cos(x);
        error = abs(x-X);
        x = X;
    end
```

This piece of code takes $x = 1/2$ as an inital guess to find a solution of the equation

$$\cos(x) = x.$$

The iteration is obtained from the rule $x_{n+1} = \cos(x_n)$ and the decision to stop the iteration procedure which accepts x_n as being close enough to a solution is when

$$|x_{n+1} - x_n| \leq \text{tolerance}.$$

2.7 FUNCTION M-FILES WITH FUNCTIONS AS PARAMETERS

In order to create a function in Matlab, one uses an extra line at the top of the m-file, for instance the following m-file

```
function x = double(y)
     x = 2*y;
return
```

is a simple function to double a given number. Note the syntax used.

The function

```
function [x,y] = PolarToCartesian(r,theta)
     x = r cos(theta);
     y = r sin(theta);
return
```

converts between polar and Cartesian coordinates. Notice this time that the function returns a vector. Function files can be as complex as you like, their main purpose is to return a specific value that has been calculated by some algorithm within the function to some other m-file.

For instance, we could extend our earlier `while` loop example to apply to any mathematical function, not just to the cosine function. The code would be

```
function X = FindZero(f)
     error = 1;
     tolerance = 1e-5;
     x = 0.5;
     while (error > tolerance)
          X = feval(f,x);
          error = abs(x-X);
          x = X;
     end
return
```

Here, a function f is passed into the m-file as a parameter and the line `X = feval(f,x)` tells Matlab to evaluate `f` as function of `x`. This gives the algorithm much more flexibility and generality to the programmer and means that a call from within Matlab of the form

```
z = FindZero(cos)
```

and

```
z = FindZero(sin)
```

would both be valid, and would try and solve the equations $z = \cos(z)$ and $z = \sin(z)$.