Introduction to Matlab

Steve Renals

23 January 2008

1 Introduction

MATLAB is a programming language that grew out of the need to process matrices. It is used extensively in science and engineering. It now has many features and toolboxes to support numerical programming (including machine learning) and visualization. Here you will meet only a fraction of MATLAB features.

1.1 Getting started

```
[dunduff]srenals: matlab
```

And this will bring up the main working window (with panels for command history, etc.) and a help window. You can type MATLAB expressions to the >> prompt, eg:

```
>> 57*64.3
ans = 3.6651e+03
>> power(10, pi/2)
ans = 37.2217
```

The second of these makes use of a builtin function, `power`, and a predefined constant `pi`. If you want to see what a MATLAB function does you can go to the help window and find it using the index, the contents, or by searching. (You can also type `help power` at the command line.) The MATLAB help system is very good, and worth exploring.

MATLAB handles complex numbers (using the symbol `j`) so it is possible to do things such as:

```
>> exp(j*pi)
ans = -1.0000 + 0.0000i
```

we can set up some variables in MATLAB and perform some computations:

```
>> x = 3.4; y = 5.7;
>> z = x*y;
>> z
z = 19.3800
```

Variables `x`, `y` and `z` persist in the MATLAB workspace until they are changed. (Note that a semicolon (';') at the end of a line prevents the result of an expression being printed to the screen—useful when dealing with big vectors and matrices.

1.2 Vectors

To represent a vector quantity in MATLAB write the vector components as a list:

```
>> v = [1 1 1];
>> w = [2 3 4];
>> v+w
ans = 3 4 5
```

Note that these are, in fact, row vectors. To create a column vector you could use the transpose operator `'`:

```
>> v'
 v = 1
     1
     1
```

or you can explicitly create a column vector:

```
>> v = [1; 1; 1];
```

Useful vector operations include:

Vector sum `v+w`

Vector difference `v-w`

Multiplication by a scalar `10*v`

Scalar dot (inner) product `dot(v,w)`

Vector cross product `cross(v,w)`

Vector magnitude (norm) `norm(v)`

Some operations on vectors work on the elements one-by-one, returning the answer as a vector of the same dimension:

```
>> sqrt([1 2 3 4])
ans = 1.0000 1.4142 1.7321 2.0000
```

The colon (':') operator generates equally-spaced point between its first and last inputs:

```
>> 1:4
ans = 1 2 3 4
>> 1:0.5:2
ans = 1.0000 1.5000 2.0000
>> 4:-1:1
ans = 4 3 2 1
```

When there are three arguments, the second one specifies the gap between the outputs (defaults to 1 when there are only two arguments). Here is another example:

```
>> squares = (1:4).^2
squares = 1 4 9 16
```

1.3 Plotting

MATLAB is very good for plotting graphs. For example using the variable created above:

```
>> plot(squares)
```

`plot(Y)` plots the columns of Y versus their index. Use a few more points:

```
>> b = (1:10);
>> plot(b.^2)
```

To plot several curves on the same graph we use the command

```
>> hold on
```

to prevent previous plots being erased. We can now type

```
>> plot(b.^1.5, 'r')
>> plot(b.^1.75, 'g')
```

The second argument defines the line colour. You can also change the line style, etc. Use the Help window to find out more. It also possible to add axis labels, a title, tick marks, etc. to a plot, either through additional arguments (see Help), or interactively using the menus on the plot window.

`plot` also works with vectors of x and y values, for example:

```
>> x = (0:.01:2*pi);
>> y = sin(x);
>> figure
>> plot(x,y)
```

`figure` opens a new plot window.

MATLAB has many demos, several to do with plotting. Either type `demo` or click Help→Demos.

1.4 Exercises

1. On the same graph in the range $(0, 2\pi)$ plot $\sin(x)$ and $\cos(x)$.
2. In a new figure plot, on the same graph, $\lg(x)$, x, $x\lg(x)$, and x^2. (Note that `log` gives $\log_e$ and `log10` gives $\log_{10}$ in MATLAB. lg is not built in.)
3. Plot $\sqrt{x}$ and $x^{\sin(x)}$ on the same axes.
4. Plot x^3 and 1.01^x on the same axes.

Make sure the axes are appropriately scaled and labelled, that the graph has a title, that the individual curves are titled in the legend etc.

1.5 Matrices

MATLAB really comes into its own when you want to do computations with matrices, such as for machine learning and pattern recognition.

You can create a matrix from the keyboard:

```
>> m = [1 2 3; 4 5 6; 7 8 9]
m =
     1     2     3
     4     5     6
     7     8     9
```

Semi-colon is used to separate rows. Here is another matrix:

```
>> n = [1 0 0; 0 1 0; 0 0 1]
n =
     1     0     0
     0     1     0
     0     0     1
```

And we can do the usual matrix operations of add and multiply:

```
>> m+n
ans =
     2     2     3
     4     6     6
     7     8    10

>> m*n
ans =
     1     2     3
     4     5     6
     7     8     9
```

`n` is the 3x3 identity matrix. An easier way to generate an identity matrix is to use the MATLAB command `eye`:

```
eye(3)
```

MATLAB overloads its operators (when it is not ambiguous). For example, add 1 to each matrix element:

```
>> m+1
ans =
     2     3     4
     5     6     7
     8     9    10
```

MATLAB can distinguish between matrix operations and element-by-element operations. The 'dot' syntax is used for element-by-element operations. For example, the following multiplies corresponding elements of two matrices:

```
>> m.*n
ans =
     1     0     0
     0     5     0
     0     0     9
```

Other useful functions include `zeros(i,j)` which generates an $i \times j$ zero matrix, and ones to generate a matrix of ones. The size command returns the dimensions of a matrix:

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
>> o = ones(2,3);
>> [rw,cl]=size(o);
```

And of course it is possible to combine functions:

```
[rw,cl]=size(ones(2,3))
```

To access the i, jth element of a matrix `m`, use `m(i,j)`. The colon operator `:` can be used to select a submatrix. For example to select a submatrix containing rows 2 and 3 and columns 1 and 2 of `m`:

```
>> m(2,3)
ans = 6
>> m(2:3,1:2)
ans =
     4     5
     7     8
```

A colon on its own selects all rows (or columns):

```
>> m(1,:)
ans =
     1     2     3
>> m(1:2,:)
ans =
     1     2     3
     4     5     6
```

Finally, a colon on its own turns a matrix into a 1D column vector, working column by column:

```
>> m(:)
ans =
     1
     4
     7
     2
     5
```

```
     8
     3
     6
     9
```

Relation operators such as < and > work with matrices too, eg:

```
>> m > 5
ans =
     0     0     0
     0     0     1
     1     1     1
```

These are not elements of `m`, but boolean values (MATLAB calls them logical indices) which indicate the positions where the test is true. To access the actual elements that are greater than 5:

```
>> m.*(m > 5)
ans =
     0     0     0
     0     0     6
     7     8     9
```

We can also use these logical indices to extract just those values that obey the condition:

```
>> m(m > 5)
ans =
     7
     8
     6
     9
```

Note that the result is a vector containing only those values that obey the condition. See also the function `find` (look at the help pages).

1.6 Other matrix commands

MATLAB has all the matrix operations you might expect. Given a matrix `a`:

```
>> a = [ 1 4 2 ; 4 2 -1 ; 2 -1 3]
a =
     1     4     2
     4     2    -1
     2    -1     3
```

we can take its transpose `a'`, compute the matrix norm `norm(a)`, the determinant `det(a)` and the matrix inverse `inv(a)`:

```
>> a'
ans =
```

```
     1     4     2
     4     2    -1
     2    -1     3

>> inv(a)
ans =
   -0.0746    0.2090    0.1194
    0.2090    0.0149   -0.1343
    0.1194   -0.1343    0.2090

>> det(a)
ans = -67

>> norm(a)
ans = 5.6866
```

It also easy to solve the linear equation $\mathbf{Ax} = \mathbf{b}$ using the operator \:

```
>> b = [1 ; 2; 3]
b =

     1
     2
     3
>> x = a\b
ans =
    0.7015
   -0.1642
    0.4776
```

Some other very useful commands are `sum` which computes the sum of each column, and `mean` which computes the mean of each column:

```
>> sum(m)
ans =
    12    15    18
>> mean(m)
ans =
     4     5     6
```

How to take the mean of each *row*? One way would be to compute the transpose:

```
>> mean(m')
ans =
     2     5     8
```

Another way would be to read the Help documentation on mean and find that it takes an optional second argument:

```
>> mean(m,2)
ans =
     2
     5
     8
```

Note that here the answer is a column vector.

And `reshape(m,r,c)` reshapes matrix (or vector) `m` to have `r` rows and `c` columns:

```
>> reshape(m, 1, 9)
ans =
     1 4 7 2 5 8 3 6 9
```

2 Programming in MATLAB

You can get lots done in MATLAB using it as an interactive environment, issuing commands one at a time, while keeping data and variables in the workspace. However you will soon want to write *functions* or *scripts*.

A MATLAB function or script is written as an M-file (a text file with the extension `.m`)

Functions A function file should contain a *single* function definition., Function `func` should reside in a file called `func.m`. Functions can take input arguments and return output values, and variables within them are local.

Scripts A MATLAB script file is a sequence of MATLAB commands (and no `function` heading) stored in an M-File. When a script executes (called by typing its name without the `.m`) the commands are executed, and variables mentioned in the script reside in the MATLAB workspace (and are available when the script ends).

2.1 Examples

As an example Matlab function here is a function that computes a one-dimensional Gaussian.

```
function p = gaussian1d(mean, var, x)

% compute a 1D Gaussian

diff = (mean-x);
p = exp(-0.5*diff.*diff/var) / sqrt(2*pi*var);
```

Save it in a file `gaussian1d.m` and call it as follows:

```
>> x = (-4:.1:4);
>> plot(x,gaussian1d(0, 1, x))
```

2.2 Control structures

MATLAB has the usual set of control structures: `for`, `if-else`, etc. A `for` loop requires a vector of loop variable values, eg:

```
for i=1:n
  % do something with i
end
```

or

```
points = [3 5 7 9 11];
for i=points
  % and so on
```

Use the help system to find out more.

2.3 Vectorization

If you write MATLAB code with lots of explicit loops (even worse, lots of nested loops), then your code is unlikely to be taking advantage of the facilities for efficient matrix-vector computation (*vectorization*). Writing your code in terms of matrix-vector operations will result in code that is easier to read and that runs an order of magnitude faster.