Introduction to Matlab

dr. G. Englebienne

September 7, 2012

All the lab exercises of the *Machine Learning: Pattern Recognition* course are done in Matlab. This provides us with a convenient programming language and an interactive environment in which to experiment with matrices and vectors. The language is geared towards the easy manipulation of mathematical vectors and matrices, and the command prompt allows us to manipulate data interactively. Moreover, it provides a large number of functions that, amongst other things, allow for convenient plotting. Matlab's plotting capabilities are probably its main strength, certainly within the context of this course.

In this document, we will have a brief look at what are, from the course's perspective, the most important aspects of Matlab. This is far from being an exhaustive list of the environments capabilities, but should help you get started if you have programmed before, only not in Matlab.

1 Getting started

The first thing is, of course, to get Matlab to run. The linux machines in the lab provide many different versions of the program, but for our purposes these are all identical and provide the same functionality (albeit with slightly different interfaces). The latest version is (at the time of this writing) in `/opt/linux-x86_64/matlab-edu_r2010b/bin`. Alternatively, you can use the free software *octave*, which provides a largely Matlab-compatible environment.

Once the environment is running, you can list files, view and change the working directory using `ls`, `pwd` and `cd` respectively. This is important when you need to load data or execute your own scripts.

2 First steps

The best way to learn Matlab is to play with it. Here are a few examples, together with the resulting output, which should give you a feel for the language. But do remember to play with them yourself!

In Matlab, variables are not defined explicitly, and are instantiated at assignment. For example, the following commands create a column vector v, with three elements, and a row vector v2, with four:

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

    1
    2
    3

>> v2 = [1 2 3 4]
    v2 =

    1 2 3 4
```

Notice how the system displays the value of the variable after the assignment. To prevent this behaviour, end the line with a semicolon:

```
>> v3 = v;
```

The transpose of a vector is most easily obtained with the prime, as in:

```
>> v'

ans =

     1     2     3
```

while standard mathematical operators are defined on vectors and matrices:

```
>> v3 = [v2;v2]'
v3 =
     1     1
     2     2
     3     3
     4     4

>> v2 * v3

ans =
    30    30

>> v4 = v2 + 5

v4 =
     6     7     8     9
```

Operators can be applied element-wise, by prefixing them with a dot:

```
>> v2 .* v4

ans =
     6    14    24    36
```

Binary operators are defined for matrices and work elementwise:

```
>> v2

v2 =
     1     2     3     4

>> v2.^3

ans =
     1     8    27    64

>> v2 == v2.^3

ans =
     1     0     0     0
```

For more structured output, use the “fprintf” function.

2.1 Getting help

The matlab environment is self-documenting. For information about a function, type “help <function>”, as in:

```
>> help plot
 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, disconnected
    line objects are created and plotted as discrete points vertically at
    X.
    ...
```

3 Program flow

For Matlab to be any use, it is of course important to be able to write control structures in our scripts. In the following, we briefly look at how to write for-loops, if-statements and functions.

3.1 For loops

For loops cycle over the elements of a vector, or the columns of a matrix, and execute the body of the statement (which is delimited by the `end` keyword) for each assignment of the control variable. For example:

```
>> M = [v 2*v 3*v]
M =
     1     2     3
     2     4     6
     3     6     9
>> for i = M
i'
end

ans =
     1     2     3
ans =
     2     4     6
ans =
     3     6     9
>> v2
v2 =
     1     2     3     4

>> for i = v2
fprintf('Value of i=%d\n',i);
fprintf('i squared =%d\n',i^2);
end
Value of i=1
i squared =1
Value of i=2
i squared =4
Value of i=3
i squared =9
Value of i=4
i squared =16
```

3.2 If statements

The if-statements are similar to the for statements:

```
>> if length(v2) == length(v)+1
disp('yes!')
end
yes!
```

4 Scripts and functions

Functions are defined with the keyword `function`, and can return zero, one or more values. They must be defined in a file, however, and cannot be created at the command prompt. They can, of course, be called from the prompt. Matlab will look for functions by searching its search path for files named after the script you're calling. For example, if you write the following code in a file called 'square.m':

```
function [s c f]=powers(x)
  s = x^2;
  c = x^3;
  f = x^4;
end
```

you can then call the function from the command prompt. Not all return values need to be used, and by default Matlab will only consider the first value. For example:

```
>> powers(3)
ans =
     9

>> [p1,p2,p3]=powers(3)
p1 =
     9


p2 =
    27


p3 =
    81
```

The search path can be checked and modified using the `path` and `addpath` commands, but it always includes the current directory.

Multiple functions can be defined within a single file, but only the function that is named after the file can be called from outside the file. The other functions are only accessible from within the file.

Finally, remember that built-in functions are not defined as `.m` files. They are compiled functions, which are much more efficient than the interpreted Matlab functions. So try to avoid doing things yourself — use the built-in functions and operators as much as possible!

5 Matrix manipulation

You can't have "nested" matrices. For example, if you type

```
>> M= [1 2 [ 3 4 ] 5]
M =

   1   2   3   4   5
```

As a consequence, you can easily grow matrices and vectors. For example

```
>> M = [];
>> V = [1 2 3];
>> for i = 1:5
>>   M = [ M; v ];
>> end
>> M
M =

   1   2   3
   1   2   3
   1   2   3
   1   2   3
   1   2   3
```

If you just want to make a matrix containing a repetition of some matrix, however, it is more efficient to use the `repmat` function.

And as a last point for today: you can also apply Boolean operations to matrices, as in:

```
>> M==2
ans =

   0   1   0
   0   1   0
   0   1   0
   0   1   0
   0   1   0
>> N=M.^2
N =

   1   4   9
   1   4   9
   1   4   9
   1   4   9
   1   4   9

>> M~=N
ans =

   0   1   1
   0   1   1
   0   1   1
   0   1   1
   0   1   1
```