Introduction to Matlab

Bart Moelans
Bart.Moelans@luc.ac.be

January 5, 2004

Abstract

This paper gives an introduction to Matlab. A big part of this paper is based on the Matlab basics of Paolo Rapisarde, assistant professor at the University of Maastricht, department of mathematics. Like Paolo once said to me: "Why invent the wheel if it already exists?".

Contents

1 Introduction

In these notes some of the main commands of Matlab are explained in a hands-on way, the idea is that while reading these pages, you will have Matlab running on your computer, and you will type the commands (the words you find in typewriter font) in order to check the results.
What is explained in these notes is about 2% of Matlab, much more is available, but is either too

specialistic to be explained in introductory material, or it pertains toolboxes, the Matlab word for a collection of functions concerning a specific domain such as, for example, neural networks (but also control theory, signal processing, and so on).
Final remark: you can read these notes, but unless you try them out, meaning type what is written in them on your keyboard, curse for a while because of some forgotten special symbol, try some modification, and so on, you will never get the hang of it... (to this purpose, it may be interesting for you to know that Matlab distinguishes between low and capital case letters- for example, a variable named a is different from another one called A).
The main goal of this course is to give you enough basics of Matlab to make the exercises for the course datamining. So these notes have no pretense of originality or particular depth, but there is no reason not to improve them for the students who will make use of them in the next years.
Comments and remarks from you will be gratefully acknowledged.

2 Matrices

The word 'Matlab' is the abreviation for 'MATrix LABoratory'. All calculations in Matlab are based on matrices. You can write the following in a Matlab command window:

```
>>5*3
```

So no matrices are involved, but you can see this as a vector (what is actually a single row matrix) multiplication.
In order to create a (row)vector, enter each element of the vector (separated by a space or komma) between [ ] brackets, and set it equal to a variable. For example, to create the vectors a, b, enter into the Matlab command window:

```
>>a = [1 2 3 4 5 6 9 8 7];
>>b = [1, 1, 2, 3, 5, 8, 5, 9, 7];
```

Notice the semicolumn, this prevents Matlab of writing out the result, example:

```
>> c = 2*3;
>> c
c =
     6
>> d=2*3
d =
     6
```

Now you can calculate $a * b$, but just typing this in Matlab will give an error, because what you (should) actually mean is $a * b^T$, you can write this in Matlab using the apostrophe:

```
>>a*b'
ans =
     260
```

You even can write $a' * b$, then Matlab multiplies each element of a with the rows of b.
Note that you don't have to use variables, you can also write:

```
>>[1 2 3 4 5 6 9 8 7]'*[1, 1, 2, 3, 5, 8, 5, 9, 7]
ans =
     1     1     2     3     5     8     5     9     7
     2     2     4     6    10    16    10    18    14
     3     3     6     9    15    24    15    27    21
     4     4     8    12    20    32    20    36    28
     5     5    10    15    25    40    25    45    35
     6     6    12    18    30    48    30    54    42
     9     9    18    27    45    72    45    81    63
     8     8    16    24    40    64    40    72    56
     7     7    14    21    35    56    35    63    49
```

It is also possible to make a column vector b, you can do this by seperating rows with a semicolumn:

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

Now it's very intuitive how to create a matrix. Lets create a 3-matrix:

```
>>A=[1,2.5; 9,27; 2.8 3.9]
A =
     1.0000  2.5000
     9.0000 27.0000
     2.8000  3.9000
```

Notice that you should use a '.' to create decimals.
Suppose that you want to create a matrix/vector with a logical order in it, for instance $a = [0, 2, 4, 6, 8, 10]$. You can create this in Matlab as follows:

```
>>a=[0:2:10]
```

If we write it symbolical as $a : b : c$, then a indicates the start value, b the stepvalue and c the stop value. When you leave the b value away, $b = 1$ is used as default value. Try the following commands:

```
>>a=[0:10]
>>b=[-3:-3:-9]
>>c=[-3:-3:-10]
```

You will notice that b and c have the same value. The best way to understand this statement is to see it as *for-loop*.
Another handy command is '...'. Suppose you have a very long vector to enter, but you want a maximum of 10 numbers at each line. Then you can type the following; use shift-enter instead of a hard enter:

```
>> a =[1,2,6,8,5,...
       7,8,9,10,12]
```

3 Matrix-tools

Matrices in Matlab can be manipulated in many ways.
For one, you can find the transpose of a matrix using the apostrophe key:

```
>>B = [1 2 3 4;5 6 7 8;9 10 11 12]
>>C = B'
C =
     1  5   9
     2  6  10
     3  7  11
     4  8  12
```

It should be noted that when B had been complex, the apostrophe would have given the complex conjugate transpose. To get the transpose, use '.' (the two commands are the same if the matrix is not complex). Now you can multiply the two matrices B and C. Remember that order matters when multiplying matrices.

```
>>D = B * C
D =
     30   70  110
     70  174  278
    110  278  446
>>D = C * B
D =
    107  122  137  152
    122  140  158  176
    137  158  179  200
    152  176  200  224
```

Another option for matrix manipulation is multiplying the corresponding elements of two matrices using the '.*' operator (the matrices must have the same dimension to do this).

```
>>E = [1 2;3 4],F = [2 3;4 5],G = E .* F
E =
    1  2
    3  4
F =
    2  3
    4  5
G =
    2  6
   12 20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
>>E^3
ans =
     37   54
     81  118
```

If wanted to cube each element in the matrix, just use the element-by-element cubing (note the dot in front of the ‘’ sign!).

```
>>E.^3
ans =
     1  8
    27 64
```

You can also find the inverse of a matrix:

```
>>X = inv(E)
X =
    -2.0000   1.0000
     1.5000  -0.5000
```

or its eigenvalues:

```
>>eig(E)
ans =
     -0.3723
      5.3723
```

There is even a function to find the coefficients of the characteristic polynomial of a matrix. The poly function creates a vector that includes the coefficients of the characteristic polynomial.

```
>>p = poly(E)
p =
     1.0000 -5.0000 -2.0000
```

Note:

$$\left| \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right| = \lambda^2 - 5\lambda - 2$$

Remember that the eigenvalues of a matrix are the same as the roots of its characteristic polynomial:

```
>>roots(p)
ans =
     5.3723
    -0.3723
```

You can select each element of a matrix seperately, or whole colums/rows using ‘:’, for instance:

```
>>M = [68 3 5; 7 3 98; 687 65 9];
>>M(2,3)
ans =
     98
>> M(:,2)
ans =
     3
     3
    65
>> M(1,:)
ans =
    68  3  5
>> M(1:2,:)
ans =
    68  3  5
     7  3 98
>> M(2:3,2:3)
ans =
     3 98
    65  9
```

Some other nice functions are *zeros*, *ones* and *eye*, see following examples:

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
>> ones(2,3)
ans =
     1     1     1
     1     1     1
>> eye(3,4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
>>eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

4 Polynomials

In Matlab, a polynomial is represented by a vector. To create a polynomial in Matlab, simply enter each coefficient of the polynomial into the vector in descending order. For instance, lets say you have the following polynomial:

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

To enter this into Matlab, just enter it as a vector in the following manner:

```
>>x = [1 3 -15 -2 9]
x =
     1  3  -15  -2  9
```

Matlab can interprete a vector of length $n+1$ as an n-th order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example:

$$s^4 + 1$$

would be represented in Matlab as:

```
>>y = [1 0 0 0 1]
```

You can find the value of a polynomial for a given value of the variable using the polyval function. For instance, to find the value of the above polynomial at $s = 2$, type:

```
>>z = polyval(y,2)
z =
     17
```

You can also extract the roots of a polynomial. This is useful when you have a high-order polynomial such as

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

Finding the roots is as easy as entering the following command:

```
>>roots([1 3 -15 -2 9])
ans =
     -5.5745
      2.5836
     -0.7951
      0.7860
```

Let's say you want to multiply two polynomials. The product of two polynomials is found by taking the convolution of their coefficients. Matlabs function *conv* will do this for you:

```
>>x = [1 2];
>>y = [1 4 8];
>>z = conv(x,y)
z =
     1  6  16  16
```

Dividing two polynomials is just as easy. The *deconv* function will return the remainder as well as the result. Lets divide z by y and see if we get x.

```
[xx, R] = deconv(z,y)
xx =
     1  2
R =
     0  0  0  0
```

As you can see, this is just the polynomial/vector x from before. If y had not gone into z evenly, the remainder vector would have been something other than zero. If you want to add two polynomials together whit the same order, a simple $z = x + y$ will work (the vectors x and y must have the same length). In general, a user-defined function, *polyadd* can be used. Such function adds two polynomials together even if they do not have the same length. It makes use of the programming construct *if... else* which I renounce to explain in detail because it works just as any *if* statement you have seen until now.
The function *polyadd* looks as follows:

```
function[poly]=polyadd(poly1,poly2)
   if length(poly1)<length(poly2)
      short=poly1;
      long=poly2;
   else
      short=poly2;
      long=poly1;
   end
   mz=length(long)-length(short);
   if mz>0
      poly=[zeros(1,mz),short]+long;
   else
      poly=long+short;
   end
```

To use *polyadd*, copy the function into an m-file, and then use it just as you would any other function in the Matlab toolbox. Assuming you had the *polyadd* function stored as a m-file, and you wanted to add the two uneven polynomials, x and y, you could accomplish this by entering the command:

```
z = polyadd(x,y)
x = 1 2
y = 1 4 8
z = 1 5 10
```

5 M-files

You can type everything you want in a Matlab command window, but this is rather impractical for large exercises. Therefore you can use m-files, the nice thing is that you can save a m-file as an

ASCII-file.
Creating an m-file goes as follows:
File->new->M-file
Before you can excecute an M-file, you need to save it using the same name as the function. In this M-file you can type a list of commands and run them by pressing the Run-button (or press F5). You can even debug an M-file.

6 Functions

To make programming easier, Matlab includes many standard functions, a function being a block of code that accomplishes a specific task. Matlab contains the standard functions of most programming languages such as *sin*, *cos*, *log*, *exp*, *sqrt*, as well as many others. Commonly used constants such as π, and i as the square root of -1, are also incorporated into Matlab.

```
>>sin(pi/4)
ans =
     0.7071
```

Note: if you want more decimals use 'format', more info:

```
help format
```

To determine the usage of any function, type help [function name] at the Matlab command window. Matlab allows you to write your own functions in order to support the construction of large and modular computer programs. Functions are defined with the function command which we will now discuss in some detail.
When entering a command such as *roots*, *plot*, or *init* into Matlab what you are really doing is running an m-file with inputs and outputs that has been written to accomplish a specific task. These types of m-files are similar to subroutines in programming languages in that they have inputs (parameters which are passed to the m-file), outputs (values which are returned from the m-file), and a body of commands which can contain local variables. Matlab calls these m-files functions. A new function must be given a filename with a '.m' extension. This file should be saved in the same directory as the Matlab software, or in a directory which is contained in Matlabs search path (depending on the version of Matlab you are using, setting the path can be done either modifying the variable directory with addpath, or using the Set path command of the pull-down File menu). The first line of the m-file should contain the syntax for this function in the form:

```
function [output1,output2] = filename(input1,input2,input3)
```

A function can input or output as many variables as needed. The next few lines contain the text that will appear when the help filename command is evoked. These lines are optional, but must be entered using include comments in an ordinary m-file. Finally, below the help text, the actual text of the function with all of the commands is included. One suggestion would be to start with the line:

```
error(nargchk(x,y,nargin));
```

The x and y represent the smallest and largest number of inputs that can be accepted by the function, *nargin* represent the number of arguments. If more or less inputs are entered, an error is triggered. Below you see a simple example of what the function, *add.m* summing two numbers, might look like.

```
function [var3] = add(var1,var2)
% add is a function that adds two numbers
var3 = var1+var2;
```

Notice that the % -sign is used to write comment.
If you save these three lines in a file called *add.m* in the Matlab directory, then you can use it by typing at the command line:

```
y = add(3,8)
```

Obviously, most functions will be more complex than the one demonstrated here. This example just shows what the basic form looks like. Also see at example of the function *polyadd*. For more sophisticated examples, look at the Matlab manuals.
Also look at *rand* and *randn*, these functions will probably be helpfull during the exercises.

7 Plotting

It is also easy to create plots in Matlab. Suppose you want to plot a sine wave as a function of time. First make a time vector and then compute the sin value for each time.

```
>>t=0:0.25:7;
>>y=sin(t);
>>plot(t,y);
```

Variations of such instructions include specifying the color and the pattern of each graph, adding titles and labels to the axes, etc.... See the help function on this. You can also plot 3D figures using *plot3*, example:

```
>>t = 0:pi/50:10*pi;
>>plot3(sin(t),cos(t),t),grid on,axis square
```

You can also use 'subplot' to have several plots in 1 window. See the Matlab manual.

8 Iterations

In this section we demonstrate how the *for* and the *while* loops are used. First, the for loop is discussed with examples for row operations on matrices. Next a demonstration of the while loop is given.
The for loop allows us to repeat certain commands. If you want to repeat some action in a predetermined way, you can use the for loop. All of the loop structures in Matlab start with a keyword such as *for*, or *while* and they all end with the word *end*.

The for loop will loop around some statement, and you must tell Matlab where to start and where to end. Basically, you give a vector in the for statement, and Matlab will loop through for each value in the vector. For example, a simple loop that will go around four times:

```
>>for j=1:4,j,end
j =
    1
j =
    2
j =
    3
j =
    4
```

Once Matlab has read the *end* statement, it will loop through and print out j each time. Another example, if we define a vector and later want to change the entries, we can step through and change each individual entry:

```
v = [1:3:10]
v =
    1  4  7 10
for j=1:4,v(j) = j;end;v
v =
    1  2  3  4
```

Note, that this is a simple example and is a nice demonstration to show you how a for loop works. However, using such construction in such case misses the whole point of Matlab. Matlab is an interpreted language and looping through a vector like this is the slowest possible way to change a vector. The notation used in the first statement is much faster than the loop, moreover, it is clearer (one of the many reasons why Matlab is so much used for fast prototyping and design in engineering firms is the fact that few lines correspond to many more operations than in any other language, and that such lines correspond to the way a human, and not a computer, would solve the problem at hand).
A better example is one in which we want to perform operations on the rows of a matrix. If you want to start at the second row of a matrix and substract the previous row of the matrix and then repeat this operation on the following rows, a for loop can do this in short order:

```
>>A = [ [1 2 3]', [3 2 1]', [2 1 3]']
A =
    1  3  2
    2  2  1
    3  1  3
>>B = A;
>>for j=2:3,A(j,:) = A(j,:) - A(j-1,:),end
A =
```

```
    1  3  2
    1 -1 -1
    3  1  3
A =
    1  3  2
    1 -1 -1
    2  2  4
```

And now a more realistic example, since we can now use loops and perform row operations on a matrix. The Gaussian elimination (remember? pivots and elementary row operations, and so on?) can be performed using only two loops and one statement:

```
>>for j=2:3,for i=j:3,B(i,:) = B(i,:) - B(j-1,:)*B(i,j-1)/B(j-1,j-1),end,end
B =
    1  3  2
    0 -4 -3
    3 1 3
B =
    1  3 2
    0 -4 -3
    0 -8 -3
B =
    1  3  2
    0 -4 -3
    0  0  3
```

If you don't like the for loop, you can also use a while loop. The while loop repeats a sequence of commands as long as some condition is met. In this example the algorithm of Euclides is worked out with a nice plot included:

```
% E U C L I D E S
% ===============
%
% Description:
% Find the greatest common divider
%
% Parameters:
% a,b real values
% PLOT if equal to 1 something is plotted
%
% Return:
% y gcd of a en b
%
function y = euclides(a,b,PLOT)
```

```
clc; %clean command window


error(nargchk(2,3,nargin));

if (nargin == 2)
    PLOT = 0;
end


A = [a]; %Evolution of the variable a
B = [b]; %Evolution of the variable b

while (a ~= b)
    if (a > b)
        a = a-b;
        A = [A a];
    else
        b = b-a;
        B = [B b];
    end
end

y=a;

if (PLOT == 1)
    clf; %clean figure
    if(size(A)>size(B)) %first plot the greatest matrix
        plot(0:size(A,2)-1,A,'*-r');
        hold on; %plot next plots in same windows
        plot(0:size(B,2)-1,B,'+-g');
        hold off;
    else
        plot(0:size(B,2)-1,B,'*-r');
        hold on;
        plot(0:size(A,2)-1,A,'+-g');
        hold off;
    end
end
```

Now type help euclides in de command window.
You wrote a help-file :-).

9 Input-output files

You can also write and read files in Matlab, I only give a small example, but you can find further information in the Matlab help. This is how you can save a file:

```
x= 25*randn(100,1);
y= 25*rand(100,1);
DAM4data = [x,y];
save c:\data\DAMdataset4.mat DAM4data -ascii;
```

And this how to read it:

```
load c:\data\DAMdataset4.mat DAM4data -ascii;
plot(DAM4data(:,1),DAM4data(:,2),.);
```

10 Questions?

If you have any further question about Matlab post them at `news://lumumba.luc.ac.be/courses.`