

# Basic Introduction to Matlab: exercise session 1

## Course HL13 (URKS)

### 1. Introduction.

Matlab was originally built to handle the algebra of vectors and matrices, always numerically. That's why the entries in Matlab will basically be numbers.

### 2. Start and Quit Matlab

To start, type **matlab** in your console (or simply click the icon in a Windows environment). To quit it, type in the Matlab window

```
>> quit
```

or

```
>> exit
```

Matlab uses commands to perform the calculations at any stage. Type

```
>> help <command>
```

to know how to use a command.

### 3. Vectors in Matlab.

To define a vector  $v$

$$v = (1 \ 2 \ 3 \ 4 \ 5) \quad (1)$$

we use square brackets

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

To have the vector written in columns we can use the **transpose** operator ( $\dots'$ )

```
>> v'
```

which yields

$$v' = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \quad (2)$$

To select specific components of the vector we use the **colon** notation

```
>> v(1:3)
```

This gives the components from the 1st to the 3rd, that is, 1 2 3

$$\text{ans} = (1 \ 2 \ 3) \quad (3)$$

The command line

```
>> v(1:2:5)
```

gives the components from the 1st to the 5th, but in jumps of 2, 1 3 5

$$\text{ans} = (1 \ 3 \ 5) \quad (4)$$

To obtain only one component we just have to type

```
>> v(2)
```

We can also use this notation to define a vector

```
>> v=[1:1:5]
```

or

```
>> v=[1:5],
```

which defines the previous vector

$$v = (1 \ 2 \ 3 \ 4 \ 5) \quad (5)$$

#### 4. Matrices in Matlab.

To define matrices in Matlab we just have to add more dimensions to our array. We define matrices row by row

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

This will generate the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix} \quad (6)$$

but we can also define a matrix column by column

```
>> B=[[1 2 3]' [2 4 5]' [6 7 8]']
```

which gives

$$B = \begin{pmatrix} 1 & 2 & 6 \\ 2 & 4 & 7 \\ 3 & 5 & 8 \end{pmatrix}, \quad (7)$$

the transposed of A,  $B = A^T$ . To select components we proceed in the same way as before

```
>> A(1:3,2:3),
```

giving

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}, \quad (8)$$

which corresponds to the first 3 rows and the last 2 columns of A. The command

```
>> A(1,3)
```

returns the component (1, 3) of  $A$  (3). To transpose we will use

```
>> A'
```

Apart from the definitions made by the user, Matlab has a list of built-in commands to generate special matrices:

- **rand(p)** generates a  $p \times p$  matrix with uniformly distributed random elements in the interval (0, 1).
- **eye(p)** generates the  $p \times p$  identity matrix.
- **zeros(p)** generates  $p \times p$  matrix of zeros.
- **ones(p)** generates  $p \times p$  matrix of ones.

## 5. Calculations with vectors and matrices.

Matlab was designed to make as easy as possible the manipulation of arrays. We multiply vectors and matrices in the following way (in the following, we will denote vectors with lower case and matrices with capital letters)

```
>> C=[1 3 4; 5 2 1; 3 1 2]
>> b=[3 1 4 2 6]
>> d=[2 3 7]
>> v*b'
>> v'*b
>> A*C
>> C*A
>> A*d'
```

For square matrices we have some specific built-in functions

- **inv(A)** gives the inverse of  $A$ .
- **det(A)** gives the determinant of  $A$ .
- **diag(A)** gives the diagonal.
- **[v,e]=eig(A)** gives 2 matrices of the dimension of  $A$ .  $v$  contains, per columns, the eigenvectors of  $A$ , and  $e$  contains, in the diagonal, its eigenvalues.

Matlab is also very handy for working with inverse matrices. For instance, in the case of a linear system of equations,  $A \cdot x = d'$ , the solution is given by  $x = A^{-1} \cdot d'$ . This is easily done with

```
>> inv(A)*d'
```

or

```
>> A\d'
```

or in inverse order

```
>> d*inv(A)
```

or

```
>> d/A
```

But how can we work with the elements of a matrix or a vector? For instance,  $v * b$  is not well defined, but suppose we need to multiply the components of one vector by the components of the other. Then we use

```
>> v.*b
```

or

```
>> b.*v
```

which gives

$$(1 \ 2 \ 3 \ 4 \ 5) \star (3 \ 1 \ 4 \ 2 \ 6) = (3 \ 2 \ 12 \ 8 \ 30) \quad (9)$$

Notice how the dot is used. The same operation holds for matrices

```
>> A.*C
>> b./v
>> C./A
>> A.^2
```

Other functions, which are not directly defined for matrices, are then applied element by element, such that the dot is not necessary

```
>> sin(v)
>> log(C)
>> exp(A)
>> 0.5-C
```

This way of working might be sometimes convenient for the user, but doesn't allow us to compute the actual function applied to the matrix itself, for instance,  $\exp(A) = 1 + A + (1/2) * A^2 + \dots$ . Other functions strictly defined for matrices are

- $[m,n]=\text{size}(A)$ , where  $m$  gives the number of rows and  $n$  the number of columns. Also for vectors.
- $\text{min}(v)$  gives the smallest element in  $v$ .
- $\text{max}(A)$  gives a vector with the biggest element of each column of  $A$ .

## 6. About variables.

Up to now we have been using some variables. To see which variables we have defined since the beginning of the session, type

```
>> who
```

and to know more about them

```
>> whos
```

To remove all the variables

```
>> clear
```

Other variables we can use are

```
>> pi
```

or

```
>> ans
```

which corresponds to the last Matlab output.

## 7. Loops.

The **for** and **while** commands are very useful for iterative works. Suppose, for instance, that we want to subtract to all the components of a vector the first one

```
>> for i=2:5,
v(i)=v(i)-v(1),
end
```

or we want to perform the Gaussian elimination in a matrix

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

where the notation  $B(i, :)$  gives the row  $i$  of the matrix  $B$ . Analogously,  $B(:, j)$  returns the column  $j$ .

Another nice example is the numerical calculation of the solution of differential equations. Suppose we want to find the numerical solution  $y(x)$  such that

$$\frac{dy}{dx} = x^2 - y^2 \quad (10)$$

with  $y(0) = 1$ . To do so, we discretize this differential equation and proceed

```
>> n=0.1
>> x=[0:n:2];
```

Notice that the addition of “;” at the end of a command gives no output. We continue initializing an array  $y$  of the same size of  $x$ :

```
>> y=0*x;
>> y(1)=1
>> size(x)
```

The last command will give 21 as a result. We then use a loop

```
>> for i=2:21,
y(i)=y(i-1)+n*(x(i-1)^2-y(i-1)^2);
end
```

(the coma at the end of the first line is not necessary). We can then plot our result (fig. 1)

```
>> plot(x,y)
```

The command **while** is almost the same. We could have done

```
>> i=2
>> y(1)=1
>> while(i<=21),
```

```

y(i)=y(i-1)+n*(x(i-1)^2-y(i-1)^2);
i=i+1
end

```

## 8. Plotting.

We can easily plot results in Matlab. Just type

```
>> help plot
```

to see the possible options you have. Example: to plot a function we must first create an array of arguments,  $x$ , and then define the function (fig. 2):

```

>> x=[0:0.1:100];
>> y=sin(x)./(1+cos(x));
>> plot(x,y)
>> plot(x,y,'rx',x,y)

```

Note that we need the dots in the second expression because  $x$  and  $y$  are arrays. Let's go back to the Euler example for the differential equation

$$\frac{dy}{dx} = \frac{1}{y} \quad (11)$$

with  $y(1) = 1$ . The implementation would be

```

>> h=1/16
>> x=[0:h:1];
>> y=0*x
>> size(x)
>> y(1)=1
>> for i=2:17,
y(i)=y(i-1)+h/y(i-1);
end

```

We know that the exact solution for that equation is (we define a function called *true* for it)

```
>> true=sqrt(2*x+1)
```

Now we can compare both

```
>> plot(x,y,'go',x,true)
```

We can also plot the error

```
>> plot(x,abs(true-y),'mx')
```

both shown in fig. 3. We can include them in the same plot using subplots (fig. 4):

```

>> subplot(1,2,1)
>> plot(x,y,'go',x,true)
>> subplot(1,2,2)
>> plot(x,abs(true-y),'mx')

```

The first argument of subplot corresponds to the desired number of rows of subplots. The second argument stands for the desired columns of subplots. In our case we will have 2 plots in a row. The third argument

determines in which subplot are we working at the moment. You can also give labels to the plots with the following commands

```
>> xlabel('x')
>> ylabel('y')
>> title('error')
```

To convert your picture to a ps file type

```
>> print -dps name.ps
```

For 3d pictures, we only add a new variable (fig. 5):

```
>> [x,y]=meshgrid(-2:0.2:2,-2:.2:2);
>> z=x.*exp(-x.^2-y.^2)
>> mesh(z)
```

## 9. Executable files.

Suppose we want to execute several times the same basic Matlab commands but changing at each time some of the parameters. A good way to do it is creating an executable file. We will follow the example of Euler's approximation for  $y' = 1/y$  (the prime stands for a derivative with respect to the argument of  $y$ ). We first need an editor to write our file, which in Matlab must have the extension **.m**. We will call it **SimpleEuler.m**. Open the editor of Matlab or any other. Then type

```
% file: SimpleEuler.m
% To find and approximation of dy/dx=1/y
% y(0)=starty
% you need first to specify h and starty
x=[0:h:1];
y=0*x;
y(1)=starty
for i=2:max(size(y)),
y(i)=y(i-1)+h/y(i-1);
end
```

The “%” symbol should be at the beginning of comment lines. Save the file with the name specified before. Then type

```
>> help SimpleEuler
```

You see how “%” is used to add comments. Before execution, we need to give values to  $h$  and  $starty$

```
>> h=0.01;
>> starty=1;
>> SimpleEuler
>> plot(x,y)
```

The result is shown in fig. 6.

## 10. Subroutines.

We want to generalize the previous case for any general function. We will use subroutines for that. As before, open an editor and type

```
function [x,y]=EulerApprox(startx,h,endx,starty,func)
% file: EulerApprox.m
% given y'=f(x,y)
% y(startx)=starty
%
% you need to give startx,h ,endx, starty, func
%
% func: routine name to calculate the right hand side of the dif.eq
%
% ex: [x,y]=EulerApprox(0,1/16,1,1,'f');
x=[startx:h:endx];
y=0*x;
y(1)=starty;
for i=2:size(y),
y(i)=y(i-1)+h*feval(func,x(i-1),y(i-1));
end
```

where `feval` stands for *function evaluation*. Now we create the file with the definition of the function, `f.m`

```
function [f]=f(x,y)
% Evaluation of the right hand side of a differential equation
f=1/y;
```

We can now execute it (fig. 6)

```
>> [x,y]=EulerApprox(0,1/16,1,1,'f');
>> plot(x,y)
```

## 11. Toolboxes.

The toolboxes are just a bunch of subroutines performing common calculations for very specific environments (neural networks, chemistry, robotics...).

## 12. Graphs

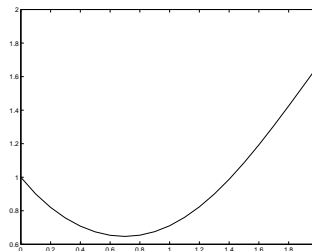


Figure 1. `plot(x,y)`

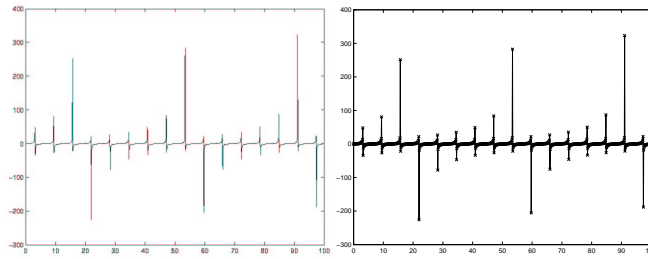


Figure 2. `plot(x,y)` and `plot(x,y,'rx','x,y)` respectively

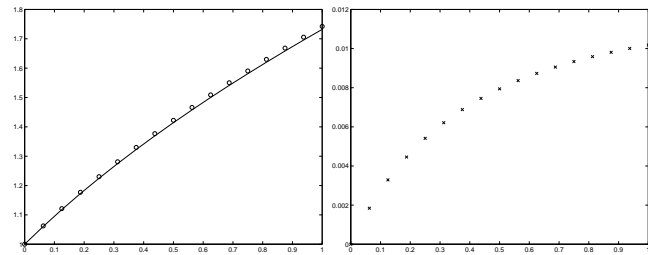


Figure 3. `plot(x,y,'go','x,true)` and `plot(x,abs(true-y),'mx')`

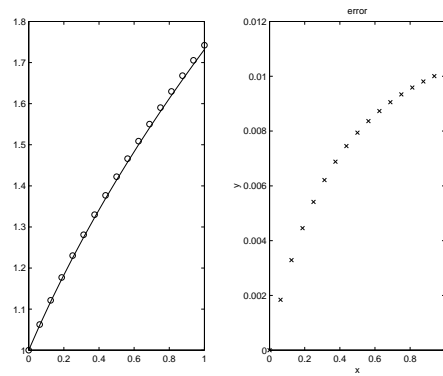


Figure 4. The two subplots. Notice the labels in the second one

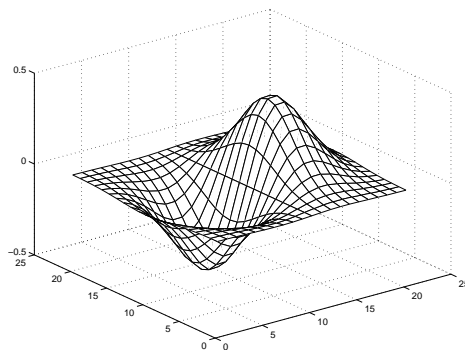


Figure 5. The 3d plot

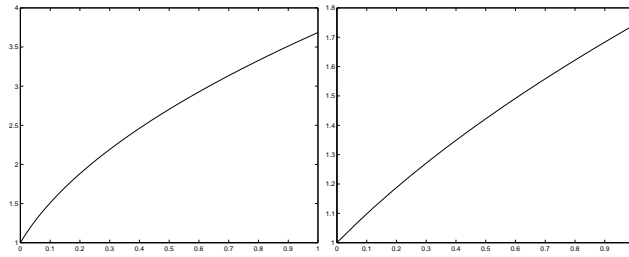


Figure 6. plot(x,y) in pages 7 and 8 respectively