An introduction to MATLAB

by Maarten Vergauwen, modified for B-KUL-I0D91a by Alexander Bertrand*

Sept, 2014

Contents

*For questions or remarks: alexander.bertrand@esat.kuleuven.be or jorge.plata-chaves@esat.kuleuven.be

1 Important prior remarks

This is a (very) brief Matlab introduction for the course 'Linear Algebra'. For the sake of completeness, this manual also includes Matlab commands for certain linear algebra tools that will be studied only later in the course, i.e., which have not been explained yet (such as determinants, singular value decomposition, etc.). It is recommended to highlight or mark all the commands for which you don't know what they are. Many of these commands will be needed in later sessions. At the start of any Matlab exercise session, go through all highlighted commands and check whether you have arrived at a point in the course where these have been studied.
Some general hints:

- Don't forget the Matlab `help` command in case you forgot how to use a certain command. You can also search for commands in the Matlab help documents.
- Be careful where you save your data! In most PC rooms, the data is lost once the PC is restarted. Remember to save everything on a USB stick, or e-mail the files to your own email account after the session.

2 What is Matlab ?

Matlab is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.
Matlab is an interactive system whose basic data element is an array that does not require dimensioning. This allows to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.
Originally, Matlab was designed for computational linear algebra (Matlab='Matrix Laboratory'). Over the years, Matlab has evolved with input from many users. In university

environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, Matlab is the tool of choice for high-productivity research, development, and analysis.
Matlab features a family of application-specific solutions called toolboxes. Very important to most users of Matlab, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of Matlab functions (M-files) that extend the Matlab environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

3 Getting started, getting stopped

3.1 Starting a Matlab session

- In Linux: Type `matlab` in a terminal to start Matlab.
- In Windows: should be in the programs list.

3.2 Stopping a Matlab session

- type `quit` or `exit`.

4 Setting your Matlab-path

- Matlab knows about all files in the current directory. You can query this directory with the command `pwd` and change it with the `cd` command, like in a UNIX terminal.
- Alternative: above the Matlab terminal, you see 'current directory'. Here you can type the path you need or you can press the '...' button on the right to select it from a list.
- Create a new directory for this exercise session, and set the path of Matlab to this directory. Preferably, take a directory in your own USB storage device (most PC labs remove all data when a PC is restarted).

5 Getting help

- To get an overview of all packages, type `help`.
- To get help on a specific topic, type `help topic`. This is very important and you should do this very frequently. A certain command often has a lot of possible uses and outputs, and can have a different meaning when applied to a vector or a matrix. For instance, the `min` command returns the minimum of a vector, or the minimum of every *column* (not row!) of a matrix. It can also be used to find the indices in the vector/matrix where this minimum occurs. Therefore, the only way to know the right use of `min` for your case, just type `help min`.
- The `lookfor` command allows you to search for functions based on a keyword. It searches through the first line of help text for each matlab function. For example: if you try `help inverse`, matlab will tell you that there is no file called "inverse.m". If you type in `lookfor inverse`, you will get over a dozen matches.

- If you have some time left at the end of this session and you want to see some matlab examples, type `demo`. From the menu displayed, run the demos that interest you, and follow the instructions on the screen.
- The help pages in Matlab are very good. If you have problems with the syntax or the behavior of a certain function, check the help page of this function first, *before asking.*

6 Use of special keys

6.1 Stopping a command

- To interrupt the execution of a command, type <CONTROL>–C. Example: type `i=1:0.001:1000` and press <CONTROL>–C to interrupt.

6.2 Using arrow-keys

- To get former commands back, use the up-arrow. This is very interesting for editing long commands. Example: Type `a=[1:0.1:10;5:0.1:14)`. This will raise an error. Then press the up-arrow and change the '`)`' into '`]`'.
- If it has been a while since you entered the command you want to review, type in the first characters of that command. Pressing the up-arrow will then only browse through the commands which start with those characters.

7 Entering data

7.1 Constants

- Some important constants exist in Matlab.
  - `pi` is 3.14159265...Example: check that `cos(pi)` $=-1$.
  - `i` or `j` is the imaginary unit $\sqrt{-1}$. It is a good idea to never use `i` or `j` as variables. However, Matlab allows this, so pay attention.
  - `Inf` is infinity.
  - `NaN` is Not-a-number. Example: check that Inf/Inf equals NaN.
  - `5e7` is the number $5 * 10^7$

7.2 Creating a vector

Vectors are one-dimensional matrices and are often used in Matlab. They are created as follows.

- Type `v=[1 2 3]` or `v=[1,2,3]` to create a horizontal vector.
- Type `v=[1;2;3]` to create a vertical vector.
- Type `v=[1:10]` to create a vector with elements 1 to 10.
- Type `v=[1:0.5:10]` to create a vector with elements from 1 to 10 with a step of 0.5.
- Typing a semicolon (;) after a command will suppress the print-out of the result to the screen. See the difference between the following statements by trying them out: `v=[1:5]` and `v=[1:5];`

7.3 Creating a matrix

Matrices are the basic features in matlab (in fact everything, even numbers or vectors, is stored as a matrix. That's where the name "Matlab" comes from: MATrix LABoratory).

- Type `m=[0 1 2;1 2 3]` to create a matrix with two rows and three columns.
- Type `m=[0:2:20;1:0.1:1]` to see that all rows (and columns) must have the same number of elements. Type `m=[0:2:20;1:0.1:2]` to get the correct matrix (don't forget the arrow-keys ;-))
- To create a matrix with 10 rows and 2 columns which is filled with zeros, type `m=zeros(10,2)`.
- To create a matrix with 10 rows and 2 columns which is filled with ones, type `m=ones(10,2)`.
- To create a unity matrix of rank 5, type `m=eye(5)`.
- Everything together: Create the following matrix:

$$m = \begin{pmatrix} 5 & 1 & 0 & 1 & 0 \\ 3 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

  A short expression for this is
  `m=[[5 1 0;3 1 0] eye(2);ones(1,5);zeros(1,5)]`.

7.4 The size of a matrix

- Type `size(m)` to see the numbers of rows and columns of the matrix m.
- Type `length(v)` to see the number of elements in the vector v.

8 Selecting data

- Type `m` to see the current value of the variable m.
- Type `m(i,j)` to select the element on the i-th row and the j-th column of m.
- Type `m(i,:)` to select the i-th row of m.
- Type `m(:,j)` to select the j-th column of m.
- Type `m([1 3],:)` to get a new matrix containing the first and third row of m.
- Type `diag(m)` to get a vector that holds the diagonal of m.

9 Matrix arithmetics and functions

9.1 Basic matrix arithmetics

Always keep in mind that most operators are matrix operators, e.g. multiplication performs a matrix multiplication.

- Addition: `x=a+b`
- Subtraction: `x=a-b`
- Multiplication: `x=a*b`
- Inversion: `x=inv(a)`
- Scalar division: `x=a/b`
- General (matrix) division: `x=a\b` solves $ax = b$ where a and b can be matrices. Notice that this statement is equivalent to `x=inv(a)*b` if a is a square matrix. If a is not invertible or ill-conditioned, a Warning message will appear. If a has more rows than columns, the solution is given in least-squares sense (will be explained later in the course).
- Transposition: `x=a'`. Always keep in mind that for complex numbers, this will also apply a complex conjugate together with the transposition. In that way, this command is equivalent to taking the conjugate transpose of a matrix. A complex-valued Hermitian matrix or a real-valued symmetric matrix is invariant under this operator. To transpose a complex matrix, without applying the conjugate operator, use `x=a.'`.
- Exponentiation: `x=a^n`

9.2 Elementwise functions

Adding a dot in front of the operator makes it element-wise:

- Multiplication: `x=a.*b` yields $x_{ij} = a_{ij} * b_{ij}$.
- Division: `x=a./b` yields $x_{ij} = \frac{a_{ij}}{b_{ij}}$.
- Exponentiation: `x=a.^n` yields $x_{ij} = a_{ij}^n$.
- Absolute value (or modulus for complex numbers): `x=abs(a)` yields $x_{ij} = |a_{ij}|$.
- Minimum:

  `x=min(a)` yields a vector with the minimum of each column.

  `x=min(min(a))` yields $x = min(a_{ij})$.
- Maximum:

  `x=min(a)` yields a vector with the maximum of each column.

  `x=max(max(a))` yields $x = max(a_{ij})$.

9.3 Matrix functions

- Type `rref(m)` to compute the reduced row echelon form of m.
- Type `rank(m)` to compute the rank of matrix m. Remember, however, that this is not a very robust way to compute the rank. It is better to check the singular values.
- Type `det(m)` to compute the determinant of matrix m.

- Type `[V,D]=eig(m)` to get the eigenvalues of matrix m (stored in D) and the corresponding eigenvectors (stored in V).
- Type `[U,S,V]=svd(m)` to get the singular values of matrix m (stored in S) and the corresponding left and right singular vectors (respectively stored in U and V).
- Type `orth(m)` perform Gram-Schmidt orthogonalization on columns of m.

9.4 Elementary math functions

- Type `cos(m)` to get the cosine of all elements of matrix m.
- In the same manner, `sin, tan, exp, asin, acos, atan, log, sqrt, ...` can be used. Type `help elfun` to see all possible elementary functions.

10 Operations on the data stack

10.1 Viewing the current variables

- Type `who` to see the names of all defined variables.
- Type `whos` to see the names and sizes of all the variables.

10.2 Saving variables into a data file

- Type `save data` to save all the variables in the file *data.mat*.
- Type `save data m v` to save the variables m and v in the file *data.mat*.

10.3 Loading a data file

- Type `load data` to load the file *data.mat*.

10.4 Clearing variables

- Type `clear m v` to clear variables m and v.
- Type `clear` to clear all variables.

11 Graphics

11.1 Making a plot

- First make a vector x and a vector y with the same length. For example, type `x=[0:0.1:10];` and `y=sin(x);`
  Remark the typical Matlab procedure here: one can not define "continuous functions" in matlab but one has to create a discrete x-vector with a certain number of values. For each of these values one can compute the value of the function.
- Type `plot(x,y)` to plot the y-values against the x-values.

- Type `hold on` after a plot if you want the next plot be plotted together with the current plot.
  Type `hold off` to turn this off. Example: plot $\sin(x)$ and $\sin(2x)$ on the same plot by typing the following commands: `plot(x,sin(x)); hold on ; plot(x,sin(2*x))`.
- Type `hold off; plot(x,sin(x),'y'); hold on; plot(x,sin(2*x),'r');` to plot $\sin(x)$ in yellow and $\sin(2x)$ in red.
- Type `title('drawing')` to put the title *drawing* above your plot.
- Type `xlabel('position')` to indicate that each x-value corresponds to a *position*.
- Type `ylabel('sine-wave')` to indicate that the y-values correspond to a *sine–wave*.
- Type `grid` to make a grid on your plot.
- Type `axis([xmin,xmax,ymin,ymax])` to set the x-axis limits to $[xmin, xmax]$ and the y-axis limits to $[ymin, ymax]$.
- Type `semilogx(x,y)` to plot y against x. The only difference with `plot` is that a logarithmic scale (base 10) is used for the X-axis.
- Type `semilogy(x,y)` for a plot with a logarithmic scale for the Y-axis.
- Type `loglog(x,y)` for a plot with logarithmic scales for both the X- and Y-axes.
- `clf` clears the current figure.
- `[x,y]=ginput` gathers the x- and y-coordinates of data points that are entered by pressing a mouse button in the current plot until the return key is pressed.
- For more information and other plot commands, type `help plot`.

11.2 Zooming

- You can zoom in and out in the current figure. Type `zoom on` to enable zooming. Now, go to the current figure and press the left mouse button to zoom. The right mouse button is used to zoom out again.
- You can also zoom in on a rectangular part of the figure. To accomplish this, move the mouse while pressing the left mouse button.
- To turn off the zoom-feature, just type `zoom off`.

11.3 Handling figures

- If you want a new figure, type `figure`.
- When more than one figure is present and you want to change one of them, you have to tell Matlab which figure you want to work on. You can do so by typing `figure(`n`)` with the n the number of the figure you want to work on.
- If you want to delete a figure, type `close(`n`)` with n the number of the figure you want to delete.
- To delete all figures, type `close all`.

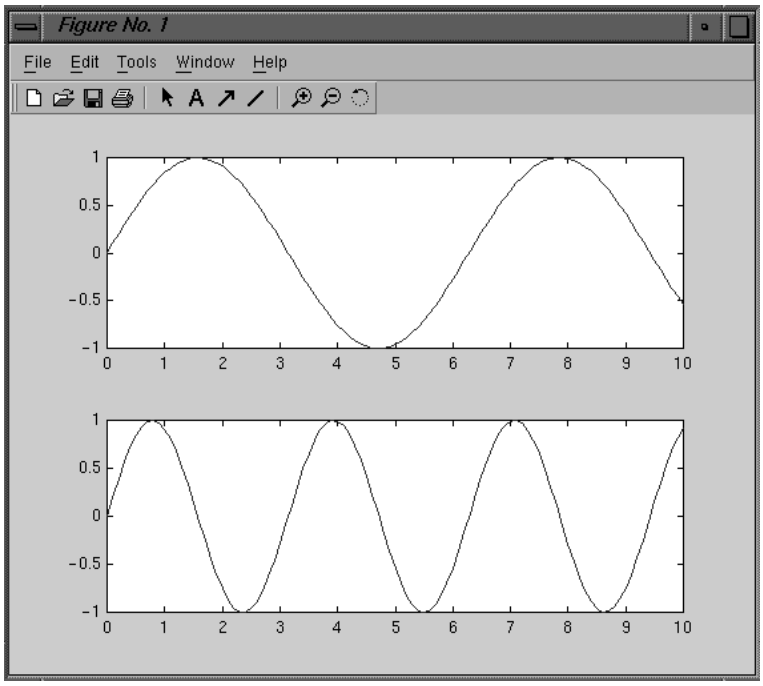

Figure 1: Two plots in one window, using `subplot`

11.4 Using subplots

- You can put different drawings in one plot with the command `subplot`.
- Type `subplot(2,1,1)` to indicate that you want two plots vertically, one plot horizontally and select the first of the two. Type `plot(x,sin(x))` to plot $\sin(x)$ there.
- Type `subplot(2,1,2)` to indicate that you want to select the second of the two. Type `plot(x,sin(2*x))` to plot $\sin(2x)$ there. The result should look like figure 1.

11.5 How to print a screen plot

- Draw a plot as described above.
- If you have more than one figure on your screen, type `figure(n)` to select the n-th figure.
- Type `print -dps plotname.ps` to make a postscript file of your plot.

12 M-files

A very important aspect of Matlab is the ability to create **M-files**. These are files, written by the user, enabling him to execute a number of commands sequentially without the need of typing them in at the Matlab-prompt. It is also possible to write functions. More details about this feature can be found in paragraph 13.

12.1 Creating an M-file

- Copy the file *example.m* from Toledo to your directory for this session and open it. Or create a new m-file *example.m* and copy the following code:
```
x=[0:0.01:10];
y=sin(5*x);
plot(x,y);
title('sine wave');
```
- start the M-file by typing `example` at the Matlab prompt.
- At this point, if you want to change the scale of x, the frequency of the sine-wave, the title, ..., you only have to change it in the M-file, save it and run it again. Example: Change x into `x=[0:0.01:3]`. Save and run it.
- Remark the difference between a **M-file** and a **Mat-file**. The former is a file which holds a "program" which can be executed in Matlab. The latter contains "data" which can be loaded and saved.

12.2 Asking for input

- You can ask the user for input with the command `input`;
- Type `n = input('Give in an odd number');` in an m-file. This will ask for an odd number and assign the result to n.

13 Functions

The biggest drawback in using M-files is the fact that you can't parametrize your code except by asking for input which stops the process until the user responds. To alleviate this problem, you can use **functions**.

- Copy the file *examplefunc.m* from Toledo to your directory.
- Open the file.
- This is an example of a function file for Matlab. It contains the following code:
```
function [t,y] = examplefunc(freq);
freq_sample=10*freq;
t=[0:1/freq_sample:0.1];
y=sin(freq*2*pi*t);
plot(t,y);
title('sine wave');
```
- Call the function by typing `examplefunc(100)`. The function will then execute. First it computes the sampling frequency as 5 times the Nyquist frequency to get a nice plotting result. Then t and y are generated and the plot is made.
- The function also gives a return value. This can be used to call the function in a slightly different way: Type `[T Y] = examplefunc(40)`. The t and y variables are then stored in the (global) variables T and Y.

14 Control structures

Matlab knows about different control structures. You can compare them to their counterparts in any programming language (like C, C++, Pascal, Basic, ...).

14.1 The "for" loop

- This control structure is ideal to execute a certain command or set of commands a fixed number of times.
- If we want to make the following vector for instance

$$v = \begin{pmatrix} 1 \\ 1/2 \\ 1/4 \\ 1/8 \end{pmatrix}$$

we type

```
for i=0:3,
v(i+1) = (1/2)^i;
end
```

14.2 The "while" loop

- This control structure is ideal to execute a certain command or set of commands until a condition is fulfilled.
- If we want to create the vector v of the previous example but keep adding elements until they are smaller than 10^{-4} we type

```
a = 1;
i = 0; while(a > 1e-4),
a = (1/2)^i;
v(i+1) = a;
i = i+1;
end
```

14.3 The "if" statement

- There are times when you want your code to make a decision. For example you want to accept only an odd number and reject an even number. For this the **if** statement is used.
- The syntax is easy:

```
if(statement),
command to be executed if statement is non-nil;
else
command to be executed if statement is nil;
end
```

- For the above mentioned example the code would be:

```
n = input('Give in an odd number:  ');
```

```
if (rem(n,2))
disp('Thank you');
else
disp('This is not an odd number !!!');
end
```

- use ==, ~=, >=, <=, >, <, &&, || to denote the logic operators $=, \neq, \geq, \leq, >, <$, and, or.

15 Exercise

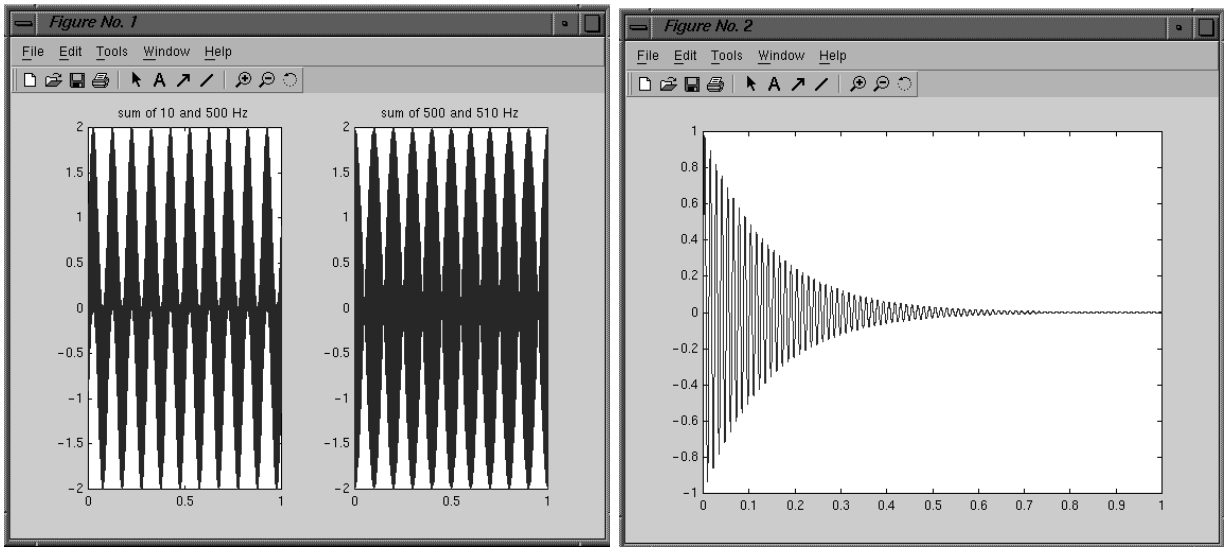

Figure 2: Results of exercise1

- Make a vector t which starts at 0, runs until 1 in time steps of 1e-4 (=0.0001).
- Create three sine-waves with frequencies $sin(2\pi 10t)$, $sin(2\pi 500t)$ and $sin(2\pi 510t)$.
- Create $y = sin(2\pi 10t) + sin(2\pi 500t)$ and $z = sin(2\pi 500t) + sin(2\pi 510t)$.
- Make a plot of y and of z (make sure you can read the values of t on the x-axis, and not the vector index). Give both an appropriate title and save them. Zoom in if necessary.
- Plot the function $f(t)$ defined as:

$$f(t) = sin(2\pi 500t)e^{-7t}$$

- Save the variables t, y and z in a file called *exc1.mat*