An introduction to MATLAB

by Maarten Vergauwen, modified for H05F2a and H05F4a by Paschalis Tsiaflakis* and Alexander Bertrand

October, 2013

Contents

1 What is Matlab ? 2

2 Getting started, getting stopped 3
2.1 Starting a Matlab session 3
2.2 Stopping a Matlab session 3

3 Setting your Matlab-path 3

4 Getting help 4

5 Use of special keys 4
5.1 Stopping a command 4
5.2 Using arrow-keys 5

6 Entering data 5
6.1 Constants 5
6.2 Creating a vector 5
6.3 Creating a matrix 6
6.4 The size of a matrix 6

7 Selecting data 6

8 Matrix arithmetics and functions 7
8.1 Basic matrix arithmetics 7
8.2 Elementwise functions 7
8.3 Matrix functions, based on svd 8
8.4 Elementary math functions 8

*For questions or remarks: paschalis.tsiaflakis@esat.kuleuven.be

9 Operations on the data stack **8**
9.1 Viewing the current variables 8
9.2 Saving variables into a data file 9
9.3 Loading a data file 9
9.4 Clearing variables 9

10 Graphics **9**
10.1 Making a plot 9
10.2 Zooming 10
10.3 Handling figures 10
10.4 Using subplots 11
10.5 How to print a screen plot 11

11 M-files **12**
11.1 Creating an M-file 12
11.2 Asking for input 12

12 Functions **12**

13 Control structures **13**
13.1 The “for” loop 13
13.2 The “while” loop 14
13.3 The “if” statement 14
13.4 Pre-allocation 14

14 Useful commands **15**

15 Exercises **17**
15.1 Exercise 1: Sum of sines 17
15.2 Exercise 2: Discrete system 18
15.3 Exercise 3: Three sequences 18
15.4 Exercise 4: Square and sawtooth 20
15.5 Exercise 5: Median filter 21
15.6 Exercise 6: The musical scale (Dutch: ‘de toonladder’) 24
15.7 Exercise 7: Half-wave rectification 24

1 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.
Matlab has evolved over a period of years 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.

2 Getting started, getting stopped

2.1 Starting a Matlab session

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

2.2 Stopping a Matlab session

- type `quit` or `exit`.

3 Setting your Matlab-path

- Matlab has access to 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. If you want Matlab to search not only the current directory for files, you can set the **Matlab-path**. This is a variable in Matlab containing a list of directories where Matlab will search for files or commands. You can use the `addpath` command to add directories to the Matlab-path.
- Alternative: in the Matlab directory toolbar, you see the 'current directory'. Here you can type the path you need or you can press the Browse button on the left to select it from a list.

- Create a new directory for this exercise session, and set the path of Matlab to this directory.

4 Getting help

- To get an overview of all packages, type `help`.
- To get help on a specific topic (a command or a package), 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*.
- Similarly you can also consult the reference pages in the Help browser by typing `doc topic`.

5 Use of special keys

5.1 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.

5.2 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.

6 Entering data

6.1 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.

6.2 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];`

6.3 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)]`.

6.4 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.

7 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.

8 Matrix arithmetics and functions

8.1 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)`
- Division: `x=a\b` solves $ax = b$. Notice that this statement is equivalent to `x=inv(a)*b`. 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 (do you remember the formula for the least-squares solution?).
- 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 Hermitian matrix is invariant under this operator. For real numbers, a Hermitian matrix is a symmetric matrix. To transpose a complex matrix, without applying the conjugate operator, use `x=a.'`.
- Exponentiation: `x=a^n`

8.2 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 $x = max(a_{ij})$.

8.3 Matrix functions, based on svd

- 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 `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.

8.4 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.

9 Operations on the data stack

9.1 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.

9.2 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*.

9.3 Loading a data file

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

9.4 Clearing variables

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

10 Graphics

10.1 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`.

10.2 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`.

10.3 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`.

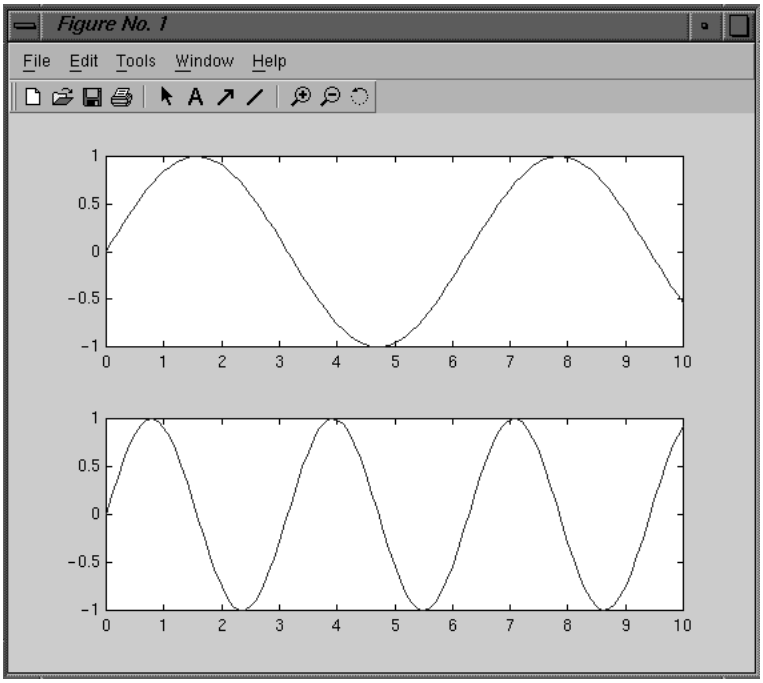

Figure 1: Two plots in one window, using `subplot`

10.4 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.

10.5 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.

11 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 12.

11.1 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.

11.2 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.

12 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.

13 Control structures

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

13.1 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
```

13.2 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
```

13.3 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.

13.4 Pre-allocation

If you use a matrix or vector variable, and do not know its content beforehand, it is often a good idea to pre-allocate this matrix as an all-zero matrix

(using the command `zeros`). Often it is tempting to fill a matrix like in this example:

```
received_bits=[];
for k=1:number_of_segments
received_bits=[received_bits bitsegment(k,:)];
end
```

This is easy since you don't have to think about what size `received_bits` will have after all bitsegments are added, and the code is simple. However, this kind of programming significantly slows down the execution time of the code, because Matlab must copy the current version of `received_bits` to a new version of `received_bits` at another place in the memory. The following code does the same and has faster execution time:

```
segmentlength=size(bitsegment,2);
received_bits=zeros(1,number_of_segments*segmentlength);
for k=1:number_of_segments
received_bits((k-1)*segmentlength+1:k*segmentlength)=bitsegment(k,:);
end
```

14 Useful commands

Here is a list of some useful, not yet mentioned commands that you might need during the project. Use the `help` command to know how they work. Re-read this list regularly to fresh up your memory. It can save you a lot of implementation time!

- `rem` (rest after division, i.e. modulo operator)
- `real`, `imag` (real part and imaginary part of complex numbers)
- `conj` (takes conjugate of complex numbers)
- `angle` (return phase angles of complex numbers in radians)
- `conv` (convolution)
- `toeplitz` (create a toeplitz-matrix, handy to represent convolutions as matrix multiplications)
- `fft`, `ifft` (FFT and inverse FFT)
- `fftfilt` (filter a signal with a given filter in an efficient way)
- `rand`,`randn` (create uniform random noise, create Gaussian random noise). Note: this function always uses the same seed when Matlab

starts up. This means that the same noise string will be produced when you restart Matlab. If you want a time dependent seed, use the `help` command for more information.

- `find` (find all elements in a matrix that satisfy a certain logic expression, e.g. $x > 0$)
- `round, ceil, floor` (make integer value out of non-integer value)
- `reshape` (change the shape of a matrix or vector)
- `spectrogram` (create a spectrogram of a given input signal. A spectrogram shows information on the frequency content of the signal over time. Carefully consider the parameters you use in this command!)
- `break` (go out of a loop immediately)
- `norm` (calculate any norm of a vector or matrix)
- `image` (display a matrix as an image, where the matrix values define the color of each pixel)
- `imagesc` (ditto, but now the matrix elements are scaled first to use the full color map. See also `help colormap`)
- `colorbar` (add a colorbar to the image to visualize the mapping between colors and numerical values)
- `soundsc` (plays an audio signal, and scales it to use full dynamic range)
- `tic,toc` (determining the execution time of Matlab-code inbetween tic ... toc)
- `(:)` (this matrix argument transforms a matrix into a vector formed by stacking all matrix columns)
- `fliplr,flipud` (flip vector or matrix in left/right or up/down direction)
- `repmat` (creates a large matrix by tiling copies of a given matrix)
- `buffer` (buffer a signal vector into a matrix of data frames)

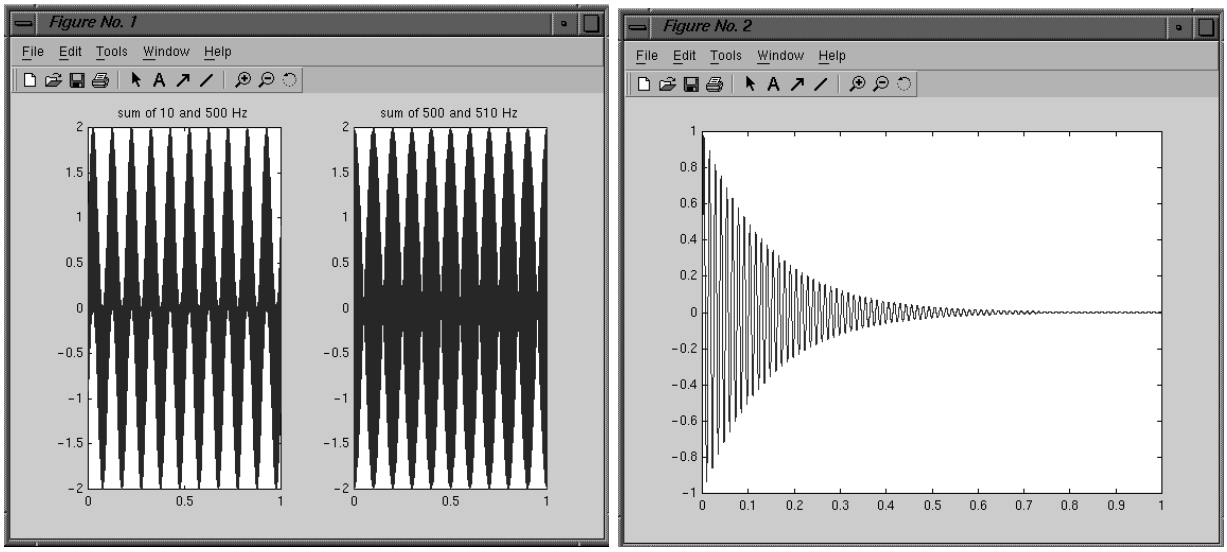

Figure 2: Results of exercise 1

15 Exercises

15.1 Exercise 1: Sum of sines

- Make a time vector t which starts at 0, runs until 1 sec with a sampling frequency of 10000 Hz.
- Create three sine-waves with frequencies 10, 500 and 510 Hz.
- Create y and z which respectively are the sum of the sine-waves of 10 and 500 Hz. and 500 and 510 Hz.
- Make a plot of y and of z (make sure you can read the time on the x-axis, and not the vector index of the signals). Give both an appropriate title and save them. Zoom in if necessary.
- Plot the function $f(t)$ defined as:

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

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

15.2 Exercise 2: Discrete system

- A certain discrete system has an input-output function described by

$$y[n] = 0.5\left(y[n-1] + \frac{x[n]}{y[n-1]}\right), n > 0$$

- Find what the output y converges to for $n \to \infty$ if $y[0] = 1$ and $x[n] = \alpha$. Do this by simulating y for different values of *alpha* and plotting y. Of course it's impossible to simulate the function for all values of n up to ∞; just choose a maximum value of n and see if the function has converged. If it hasn't, increase your maximum value of n. You can then read the value the function converges to from the plot with the command `ginput`.

- Determine the settling-time, i.e. the sample-index after which the output signal is always less than 2% from the equilibrium point. Make sure this happens automatically (i.e. without manually checking on a plot).

15.3 Exercise 3: Three sequences

- Write a Matlab program (called *sequences.m*) to generate the following sequences.

  1. The unit sample sequence $\delta[n]$
  2. The unit step sequence $\mu[n]$
  3. The ramp sequence $n\mu[n]$

- The input parameter specified by the user is the desired length. Be careful: half the samples should lie before 0 and the other half after 0. Write the program only for odd lengths. (You can check if a number if odd or even with the function `rem`. Read the help page for an explanation on this function).

- Use this program to generate the first 101 samples of the three functions.

- Plot these samples with the command `stem`. Type `help stem` to see what this function does.

- Now plot the same data, but leave out all odd sample-times. Note that this can be done easily with only one (plot)-command, so you don't need commands like `rem` and such.

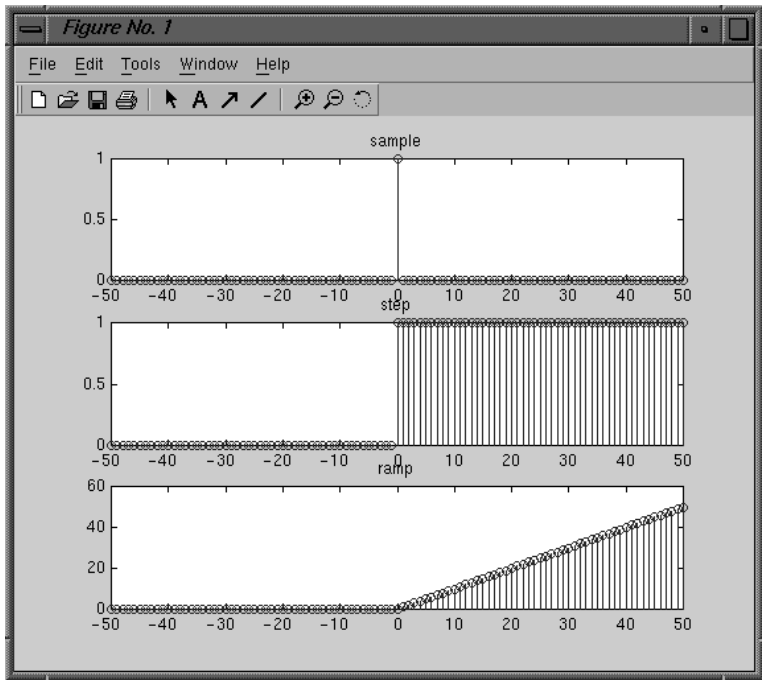

Figure 3: Results of exercise 3

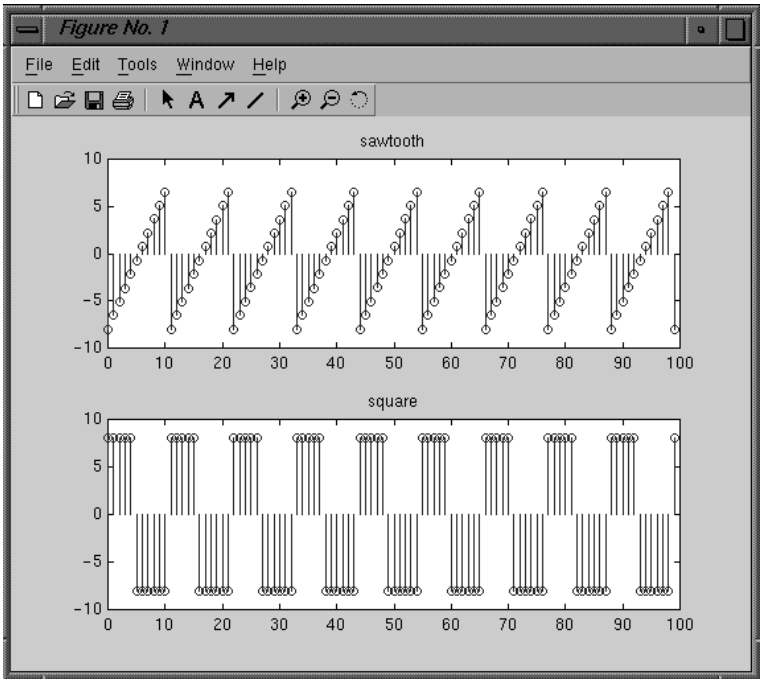

Figure 4: Results of exercise 4

15.4 Exercise 4: Square and sawtooth

- The square wave and the sawtooth are two periodic sequences.
- Write two Matlab programs to generate the two sequences displayed and plot them using the function stem.
- The input parameters specified by the user are the desired length L of the sequence, the peak value A and the period N. For the square wave sequence there's one extra parameter: the duty-cycle which is the percent of the period for which the signal is positive.
- Use this program to generate the first 100 samples of the two sequences with a peak value of 8, a period of 11 and a duty-cycle of 40%.

15.5 Exercise 5: Median filter

- The median filter is a non-linear filter that is often used in image-processing to smooth images while at the same time preserving the edges. For usage in image-processing one needs the 2D-version of the filter. We will implement the 1D-version.
- The median filter has only one parameter: the length. We will only worry about filters with odd lengths (i.e. length mod 2 is 1).
- To get the result of the filter we slide a window with the given length over the vector we want to filter. We order the values of this window in ascending order and take the middle one. The original value of the element in the middle of the window is replaced by this "median" value. Then we slide the window one place further and repeat the process. The first and last $(length - 1)/2$ values are not changed. Figure 5 will make things clear.
- For the sorting the function `sort` can be useful.
- Your task is to write a function "median" `function out = median(length,in)`. The result (stored in out) should be the median filtered version of the vector "in" with a median filter of length "length". Obviously, you may not use the pre-defined command `median`.
- Check the filter on the vector v as given in the file `medianvector.mat` on Toledo. The original and filtered vector with a median filter of length 5 look like figure 6.

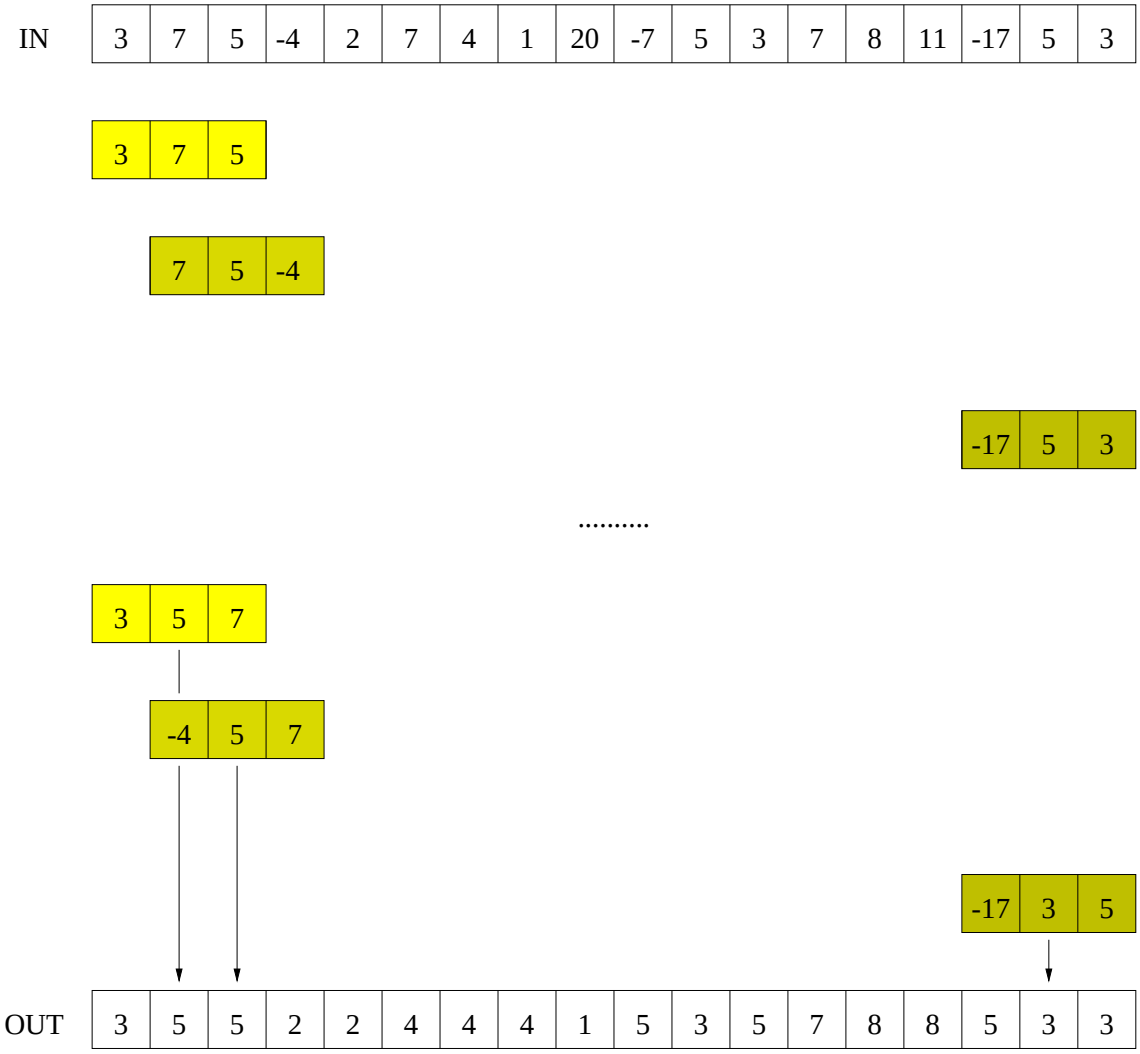

Figure 5: the median filter

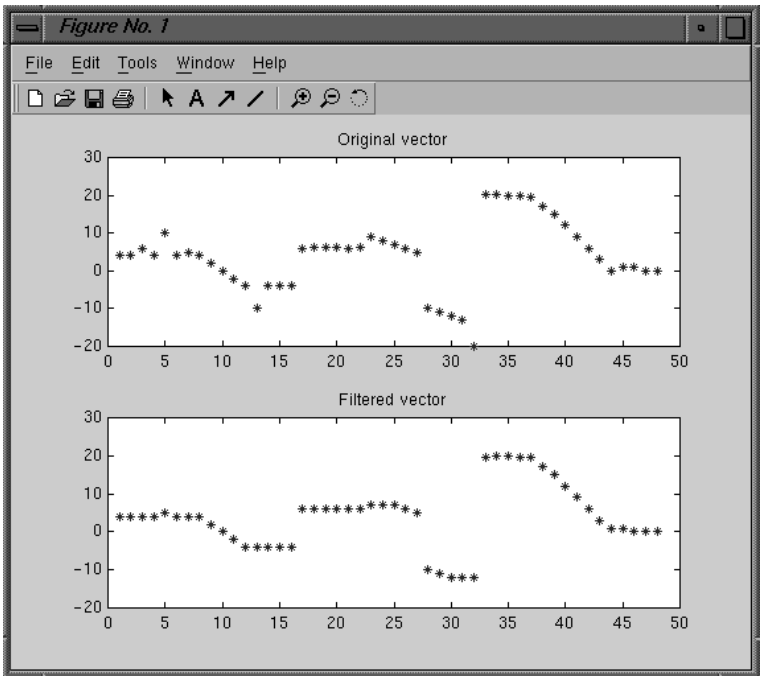

Figure 6: Results of exercise 5

15.6 Exercise 6: The musical scale (Dutch: 'de toonladder')

- Play the complete musical scale on your computer loudspeakers using Matlab, or compose your own song. All tones are given below (starting with 'do'=C):

C	262 Hz
D	294 Hz
E	330 Hz
F	349 Hz
G	392 Hz
A	440 Hz
B	494 Hz
C	523 Hz

- Generate a colored image of the spectrogram of this signal. The vertical axis should indicate the frequencies in Hertz, the horizontal axis should indicate the time in seconds. Your colleagues should be able to interpret the spectrogram and write down the 'music' you generated.

15.7 Exercise 7: Half-wave rectification

- Generate a white-noise signal that produces approximately the same number of negative and positive values.

- Half-wave rectify this signal, *without* using a control structure (=for, while, if,...). (If you don't know what half-rectification is, google is your friend...)