Assignment 2 (Compulsory)

MATLAB, short for Matrix Laboratory, is a so-called 4th generation programming language. These are programming languages that are designed with a specific application area in mind. In case of MATLAB, the application area is mathematical analysis, data processing and data visualizations. MATLAB allows:

- matrix manipulations
- plotting of functions and data
- implementation of algorithms
- creation of user interfaces
- interfacing with programs written in other languages, including C, C++, Java, and Fortran.
- interfacing with instruments (instrument control and image acquisition)

Note: You can watch the tutorial videos: https://goo.gl/u8iBrm that will help you to get started with MATLAB.

Some of the functionality is offered standard. Others are added optionally be means of installed toolboxes. There are many toolboxes. Here are just a few examples:

Communications Toolbox	Optimization Toolbox
Control System Toolbox	Parallel Computing Toolbox
Curve Fitting Toolbox	Partial Differential Equation Toolbox
Filter Design Toolbox	Pattern Recognition Tools
Global Optimization Toolbox	Robust Control Toolbox
Image Acquisition Toolbox	Signal Processing Toolbox
Image Processing Toolbox	Statistics Toolbox
Computer Vision Toolbox	Symbolic Math Toolbox
Instrument Control Toolbox	System Identification Toolbox
Neural Network Toolbox	Wavelet Toolbox

The goal of this first assignment is twofold:

1. Getting acquainted with MATLAB and its operating structure
2. Knowing how to visualize data in graphs

Examples of visualization code can be found in Section 6 of the Lecture notes.
Want to know more? Visit:
http://www.mathworks.nl/help/matlab/getting-started-with-matlab.html.

Instructions for this assignment:

- Go through the exercises below to develop your MATLAB skills and understanding of signals. The output will be an m-file with MATLAB code and a filled-out pdf form accompanying this. To fill out the form: save the pdf form to file, answer the questions, save it, upload it to Canvas.
- Use an Adobe pdf reader (don't use a plug-in of a browser).
- **Don't forget to fill in your family name, student id, and email address** on top of the pdf-form.
- **Copy your MATLAB code to the last page of the pdf form.**
- Save your work and submit it on Canvas. Deadline: 20 September 17:30.
- The work must be done individually!

Part I: The first brush with MATLAB

When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains sub-windows. For instance:

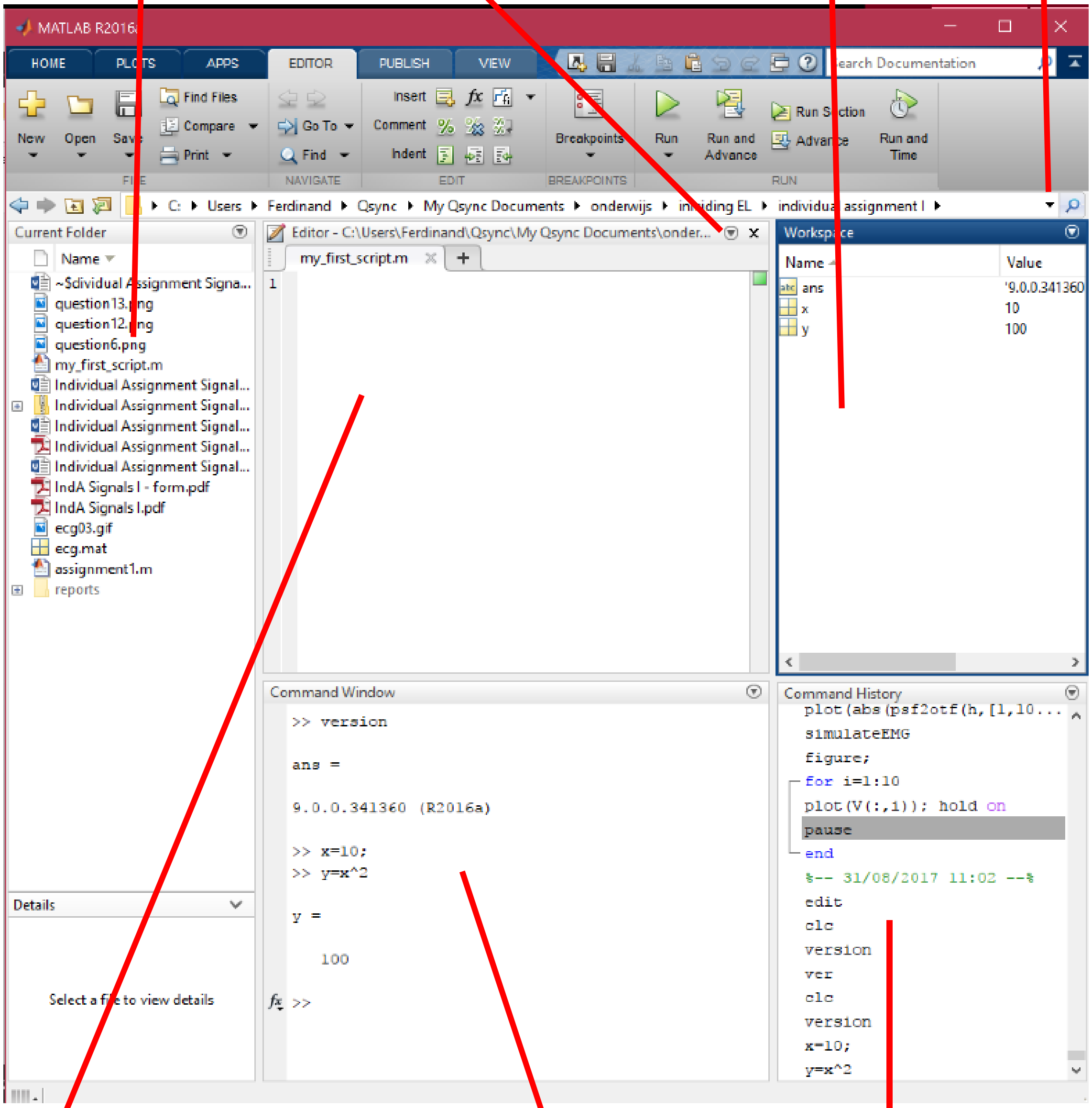
- The Command Window: for entering commands and executing functions
- The Command History: where previously run commands are logged
- The Workspace: where an overview of variables are shown
- The Editor: where new MATLAB code is created

See the files in the current folder. Double click an m-file to open a MATLAB script or function

Move Editor outside desktop (undock)

See which variables are defined in the workspace

Change current folder



Current Folder:

- ~$dividual Assignment Signa...
- question13.png
- question12.png
- question6.png
- my_first_script.m
- Individual Assignment Signal...
- Individual Assignment Signal...
- Individual Assignment Signal...
- Individual Assignment Signal...
- Individual Assignment Signal...
- IndA Signals I - form.pdf
- IndA Signals I.pdf
- ecg03.gif
- ecg.mat
- assignment1.m
- reports

Workspace:

Name	Value
ans	'9.0.0.341360
x	10
y	100

Command Window:

```
>> version

ans =

9.0.0.341360 (R2016a)

>> x=10;
>> y=x^2

y =

   100

>>
```

Command History:

```
plot(abs(psf2otf(h,[1,10...
simulateEMG
figure;
for i=1:10
plot(V(:,1)); hold on
pause
end
%-- 31/08/2017 11:02 --%
edit
clc
version
ver
clc
version
x=10;
y=x^2
```

Use the editor to create and run new MATLAB commands (scripts) and functions

Enter MATLAB commands at the line prompt

View or execute previously commands from the command history window

The Command Window allows you to use MATLAB as a calculator, as shown in the figure above.

An expression, like: $$\frac{1}{2+\sin^2(\frac{1}{2}\pi)}+e^{-\frac{4}{5}}$$

is entered as (**try this**):

```
>> 1/(2+sin(pi/2)^2) + exp(-4/5)
```

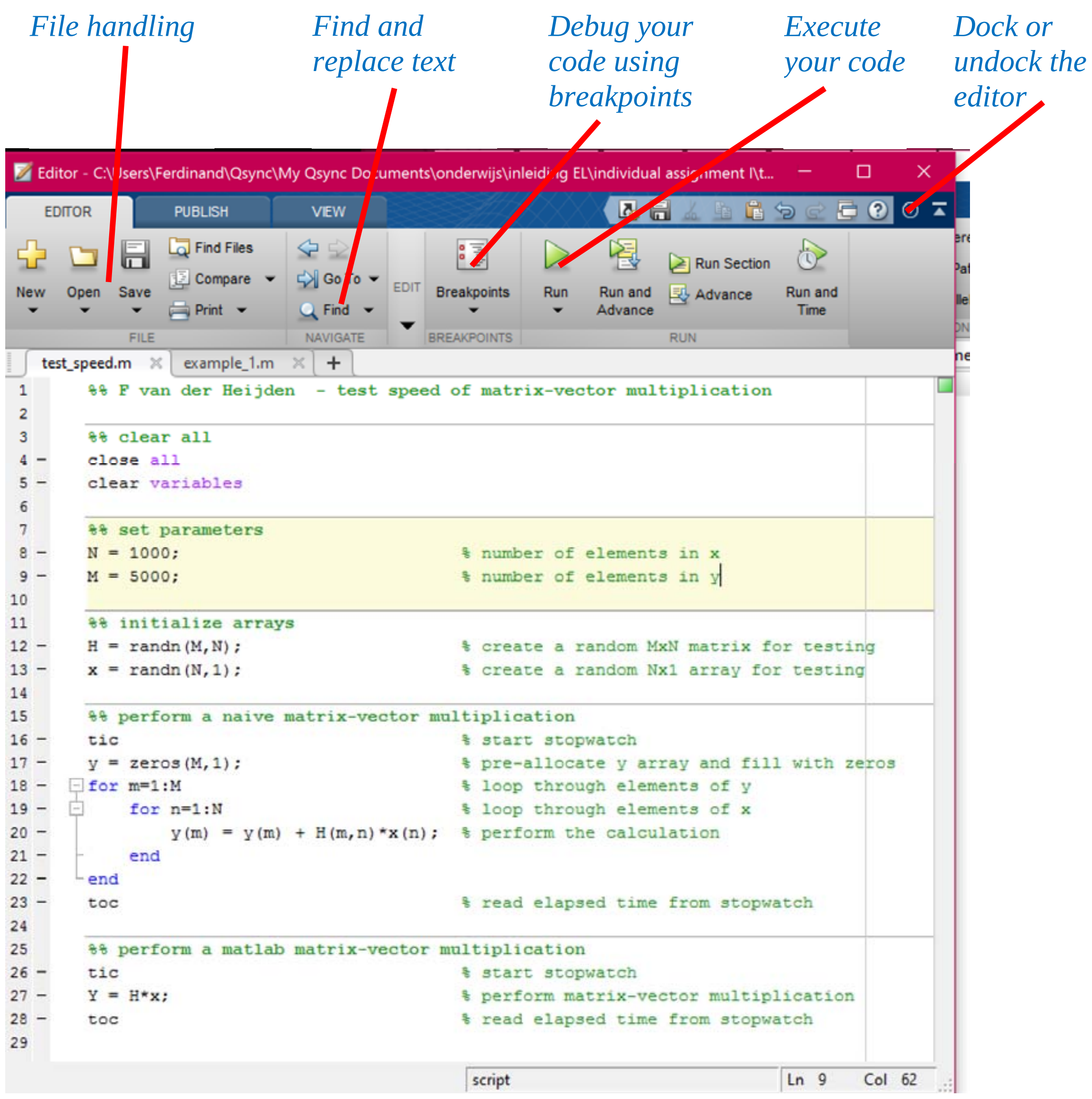
which will produce something like:

```
ans =

    0.7827
>>
```

Using the editor

It is unusual to work with the command window as we did in the example. MATLAB code is almost always stored and retrieved in so-called m-files. These m-files are created and manipulated with the MATLAB editor. If you undock the Editor window, you get:



```
%% F van der Heijden  - test speed of matrix-vector multiplication

%% clear all
close all
clear variables

%% set parameters
N = 1000;                           % number of elements in x
M = 5000;                           % number of elements in y

%% initialize arrays
H = randn(M,N);                     % create a random MxN matrix for testing
x = randn(N,1);                     % create a random Nx1 array for testing

%% perform a naive matrix-vector multiplication
tic                                 % start stopwatch
y = zeros(M,1);                     % pre-allocate y array and fill with zeros
for m=1:M                           % loop through elements of y
    for n=1:N                       % loop through elements of x
        y(m) = y(m) + H(m,n)*x(n);  % perform the calculation
    end
end
toc                                 % read elapsed time from stopwatch

%% perform a matlab matrix-vector multiplication
tic                                 % start stopwatch
Y = H*x;                            % perform matrix-vector multiplication
toc                                 % read elapsed time from stopwatch
```

When you run MATLAB the first time, the editor is often docked in the main window. You can easily undock it with the small button on the right top corner.

Looking at the m-code given in the editor above:

A. The first line of the m-file is always a short explanation of what the m-file should do.
B. The fourth and fifth lines of the code, shown above, are not mandatory. They reset the state of MATLAB. Each time when the code is executed, MATLAB has a well-defined starting condition.
C. As you can see, the code is well structured into sections (in MATLAB jargon: cells). Each cell is preceded by a line starting with `%%` followed by a short explanation what the cell is supposed to do. For instance:
   `%% clear all` and `%% set parameters`
D. Also, most lines (as a rule of thumb: more than 50%) are terminated with comment that explains what the line is supposed to do. Comments start with `%`.

For readability of your code, these four points above are essential, and you must always stick to these.

MATLAB coding: an example

MATLAB is able to execute mathematical expressions. As an example, we consider the matrix-vector multiplication:

$$\mathbf{y} = \mathbf{H}\mathbf{x} \tag{1}$$

where $\mathbf{x}$ is a N -dimensional vector, $\mathbf{H}$ is a $M \times N$ dimensional matrix, and $\mathbf{y}$ a M-dimensional vector. The multiplication is defined by:

$$y_m = \sum_{n=1}^{N} h_{m,n} x_n \qquad \text{for } m = 1, \cdots, M \tag{2}$$

A **C-programmer** might implement this as follows:

```
for (m = 0; m < M; m++)
{ y[m] = 0;
  for (n = 0; n < N; n++)
  {
     y[m] = y[m] + h[m][n]*x[n];
  }
}
```

A **naïve MATLAB-programmer** would translate this into the following MATLAB script:

```
y = zeros(M,1);                  % pre-allocate y array and fill
with zeros
for m=1:M                        % loop through elements of y
    for n=1:N                    % loop through elements of x
        y(m) = y(m) + H(m,n)*x(n); % perform the calculation
    end
end
```

A **not-so-naïve MATLAB-programmer** will do this:

```
y = H*x;                         % use matrix-vector multiplication
```

This last approach has several advantages above the naïve MATLAB implementation:

a) The readability of the code is much improved (see the alikeness with eq (1)).
b) It takes much less time to create the code, and the chance on mistakes is much smaller.
c) The code executes much faster

1. The default MATLAB working folder is `%USERPROFILE\documents\matlab` which will be created to first time that you use MATLAB. So, start up MATLAB and see whether this is really the case. Navigate with Windows Explorer (MacBook users: Finder) to this folder. Create a sub-folder `IEEE_Sign_Assigment2`. To organize your work neatly: unpack the zip-file `Assignment2.zip` and put all these files in the newly created sub-folder. In MATLAB: navigate to this sub-folder. Open the file `test_speed.m`. This file

contains the code as shown on page 61. However, the last four lines are missing. Add these missing lines to the code. Execute this script by pressing the Run button. In the command window you see the results (in seconds) of the stopwatch.

2. Execute this script once again and see whether there is a difference in execution time.

Insert your results in the pdf form `IndA2_form.pdf`.

You might have observed the large difference between the executing times of the naïve approach, and the not-naïve approach, especially in the second run. The reason is that MATLAB is vector oriented. It is optimized to process vectorised data, i.e. data arrays rather than scalars (single variables). In the naïve approach, MATLAB uses an interpreter. In the second approach, an optimized pre-compiled code is used. The first time that MATLAB's interpreter encounters the statement, it has to load this code once, but after that it is cached in memory and the second call runs even faster.

For efficiency of your code: always try to avoid for-loops, and try to vectorise the processing.

There are many techniques to avoid for-loops. You will encounter some of them in the sequel. For now, it suffices to get to know the *colon* operator. That is the operator '`:`'

3. a) In the command window, execute the following: `D=1:6,` and see what happens.
   b) Execute: `E = 0:.1:.5` and see what happens.
   Fill in the form. You can find help about the colon operator by typing `doc colon`.

4. The colon operator is also useful to select columns or rows from 2D arrays. As an example, execute the following code:
   ```
   A = magic(5)
   A(3,:)
   A(:,3)
   ```
   The function `magic(5)` produces a 5x5 array with some specific properties (that are not relevant for now). Fill in the form.

   You might also want to know what happens with the following constructions:
   ```
   A(1:3,:)
   A(1:2:3,:)
   A(2:3,2:3)
   A(:)
   A(1:2:end)
   ```
   Try this, and try to understand the logic behind this.

Part II: Plotting x-y data

In this part, you will learn the basics of visualization of data with MATLAB. The most often used MATLAB function to create XY graphs is the function `plot`. To demonstrate its usage, we first have to create data points along the horizontal x-axis with associated data points along the vertical y-axis. As a simple example, execute the following code in your command window:

```
x = 0:5;
y = [2,3,4,3,1,0];
```

Our purpose is to plot the y-data against the x-data. For that, we first have to create a so-called *figure window*. Type and execute:

```
        figure(1);
```
This call creates a new window on your screen. This window is identified here by the number 1, and this number is called the *figure handle*. The figure handle enables us to change figure properties like size, position, figure title, default fonts and font size, etc. The next step is to create an axes graphic object. This is done as follows:
```
        ha = axes;
```
Execute this. You will see now x- and y axes within the figure window. By default, the limits are between 0 and 1. The variable `ha` is the axes handle, which identifies this graphic object. Type:
```
        ha
```
to see all kind of default axes properties. The handle can be used to change these properties.

Now we are ready to plot the data:
```
        hp = plot(x,y);
```
You will see a graph of the x- and y-data shown as coloured line segments that connects each (x,y) point in the order that were given in the `x` and `y` array. This graph is a graphic object of the type *line*. The variable `hp` is the handle to this object. You can see some of the properties by typing `hp<enter>`. You can change the properties of the object using this handle. For instance, type the following text in the command window, and see what happens:

5. Inspect after each line what happens with the graph and its properties, and describe this as comment in the pdf form:
```
hp.Color = 'r';             % add comment that describes what happens
hp.LineWidth = 1.5;         % add comment that describes what happens
hp.LineStyle = '--';        % add comment that describes what happens
hp.Marker='o';              % add comment that describes what happens
hp.LineStyle='none';        % add comment that describes what happens
```

   Graphs are only complete if they are provided with at least axis labels. In addition, you may want to add a title and perhaps a legend. These can be added by using the following functions:

`xlabel:`	add a label at the x axis
`ylabel:`	add a label at the y axis
`title:`	add a title
`legend:`	add a legend

   Use these functions to add appropriate labels, title and legend. (For now the actual text that you add is not important). Type ‘`doc xlabel`’, etc, to find the explanation how to use these functions.

6. Apply the function to the current graph, and save the figure to file (use: either `print –r600 –dpng question6.png` or `print –r600 –djpeg question6.jpg`)[*]. Add the png-file or jpg-file to the pdf form.

Part III: Plotting functions of time

In this part, we consider a given function $s(t)$ that could be the impulse response of an electric circuit:

[*] Don’t copy-paste this, as pdfs use other characters than the editor; you have to retype it

$$s(t) = A\exp\left(-\frac{t}{\tau}\right)\cos(2\pi ft) \qquad \text{for } t \geq 0 \tag{3}$$

The purpose of this part is to create a nice plot of this function. That is, we have to create an xy-graph in which the time t forms the x-axis, and $s(t)$ is plotted in the y direction against t. To do so, we have to sample the function $s(t)$. The plot is presented in Figure.1.

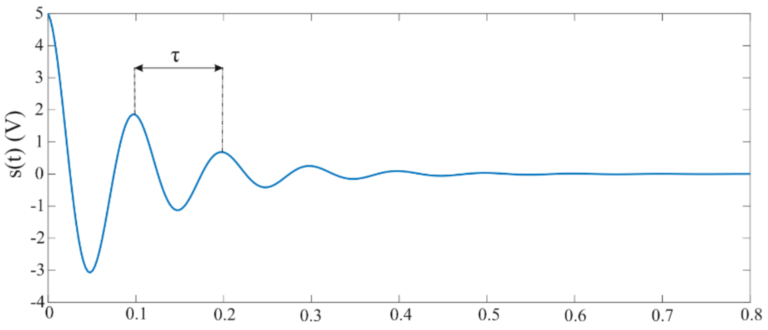


Figure 1: Plot of s(t) where τ is the time constant.

First, we define a finite number of points in time:

$$t_n \quad \text{with} \quad n = 1, 2, \cdots, N \tag{4}$$

Then we create the samples of $s(t)$ by computing$_s$ for every value of t_n:

$$s_n = s(t_n) \tag{5}$$

It is common practice to *equidistantly* sample the signal. That is, any two successive points in time differ always by the same period of time:

$$t_{n+1} - t_n = \Delta \qquad \text{for any } n \tag{6}$$

$\angle$ is called the *sampling period*. Because the time starts at zero, and because of eq. (6), we can generate the sequence t_n as follows:

$$t_n = (n-1)\Delta \qquad \text{for } n = 1, \cdots, N \tag{7}$$

To generate the sequence t_n, we have to choose the sampling period Δ and the number N of points in time. The criteria to select these two variables are:

a) The sampling of the data should not be visible in the graph.
b) The sequence t_n should cover the interesting part of the time interval. Therefore, the last point $t_N = (N-1)\Delta$ should be selected appropriate. Once t_N and N are chosen, $\angle$ is also fixed.

7. Suppose that it is given that:

$$A = 5\ V \qquad \tau = 10\ ms \qquad f = 100\ Hz$$

The `exp()` function in eq (3) forces the signal to approach to zero as time increases.

a) Determine the last point in time t_N such that $s(t_{N-1})$ is almost zero.

b) Assuming $N=300$, calculate Δ.

(Fill in the results in the pdf form.)

8. Create a new m-file, `ass2.m`, start a new section, and add code that generates an array t that corresponds to t_n with $n = 1, \cdots, N$.
9. Add code that generates the array `s` that corresponds to $s(t_n)$. **Hint**: see footnote[†].
10. Plot the function; add appropriate x- and y labels, and a title.
11. The function `min` calculates the minimum value of an array. Use this function to calculate the minimum of $s(t)$. Add this to the graph as a horizontal line that extends from $t_1 = 0$ to

[†] By default the `*` operator is a matrix-vector multiplication as shown in eq. (1) and (2). The expression `exp(-t/tau)` will yield an array with a size equal to the one of `t`. The same holds for an expression like `cos(2*pi*f*t)`. Multiplication of the two expression is not a compatible matrix-vector multiplication. Thus, an error message will result (**try this**). The operator "`.*`" is an element-by-element multiplication. This is what we need.

t_N (Matlab: `t(end)`). Include a legend. Hint: see footnote[‡]. Add the figure to file as a jpg or png file, and insert it in the pdf form.

Part IV &V: Plotting signals

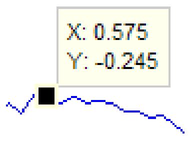


Figure 2 a data tip

A signal is often a time-varying voltage that, after AD conversion (analogue to digital), becomes a sequences of numbers: the *samples*. Each sample is associated with a point in time. In this part, we consider an electrocardiogram (ECG) which is a record of the voltages between electrodes that are attached to the surface of the skin of the chest. These voltages represent the electrical activity of the heart. A typical ECG is more or less periodic.

An ECG record is available in the MATLAB datafile `ecg.mat`. After loading this file (MATLAB command: `load ecg`), the samples of the ECG data are available in a MATLAB array `s`. The corresponding time sequence of these samples is in the array `t`.

12. Load the signal and plot it. With the function `axis([xmin xmax ymin ymax])` you can set the limits of the graph (horizontally between `xmin` and `xmax`; vertically between `ymin` and `ymax`). Set the horizontal limits exactly to 4 times the period of the signal, and the vertical axis to -1 and +1.

    Hint: MATLAB's figure window contains a toolbar. On this toolbar, pressing the icon invokes the data cursor tool. With that you can interactively place a data tip on the graph. This enables you to interactively inspect your data. Under control of the keyboard arrows, the data tip glides from left to right or vice versa. Selecting the data tip with the right mouse button invokes a menu to further process the data tip.

13. Use a data tip to find the period of the signal (chose a strategy to find this period). Also, find an interval of the time axis such that exactly 4 periods of the signal are within this interval. Print the figure to file and insert in the pdf form.

The report:

- Just fill out the pdf form. Don't forget your name, email and student id. Copy and paste your code from the m-file `ass2.m` to the last page of the form.
- Deadline: 20/09/2018 at 17.30 strictly.
- Grading: This assignment will be graded (10 points).
- Submission via Canvas "Assignments".

[‡] By default, an axes replaces a graphic object if a new one is invoked. By executing `hold on` before plotting the second object, this second object is added without deleting the already existing one.