Digital Systems

EEE4084F

Practical 1 – Octave

[30 Marks]

The focus of this task is on using OCTAVE (the free sort-of MATLAB program) and doing some statistical operations. In later assignments you will make further use of these statistical functions, and perhaps reuse this code, in comparing and discussing results obtained in other practicals and projects (for example, correlation can be used in analysing gold standard results to higher-speed approximation result).

Please don't use MATLAB for this assignment – the command syntax is slightly different and part of the point is getting you to use open-source software.

For this assignment you are free to choose where you want to work: a lab computer, on your laptop, or at home. *If working on your own PC* I advise obtaining a recent version of OCTAVE.

What to hand in for Practical 1?

Note that Parts 1 and 2, as well as Part 3 steps #1 and #3 are entirely optional. You can skip these items.

Hand in a short report (aim for two pages: the page limit for this assignment is 3) briefly describing your solutions for Part 3 step #4, as well as Part 4. The **callouts** show a submission requirement.

This is an example callout

Format your report as if it is an article (i.e. don't follow the same chronology as the prac-sheet – format it as "Introduction – Method – Results & Discussion – Conclusion"). In the "Method" section, theorise about what you expect and how you plan on testing said theory. In the "Results" section, confirm that you obtained what you expected (or explain why you obtained something unexpected).
Hint: You are trying to answer two questions: 1) "Is my `mycorr` function better suited to run correlation tests, in comparison to the built-in equivalent?" and 2) "Are my theories relating to the correlation of time-shifted sinusoids correct?"

NB: Please hand in your report as per the due date set for Practical 1 in Vula.

A hint: Calculating speed-up

Speed-up = T_{p1}/T_{p2}

Where T_{p1} = Run-time of original / non-optimal program

T_{p2} = Run-time of optimised program

For obtaining a repeatable timing value, run each version (i.e. initial version and optimized version) of your programs more than once and discard the first measured time. You can, if you want to be complete, indicate what the initial speed up was and then the average speed up.

Part 1: Starting out in OCTAVE

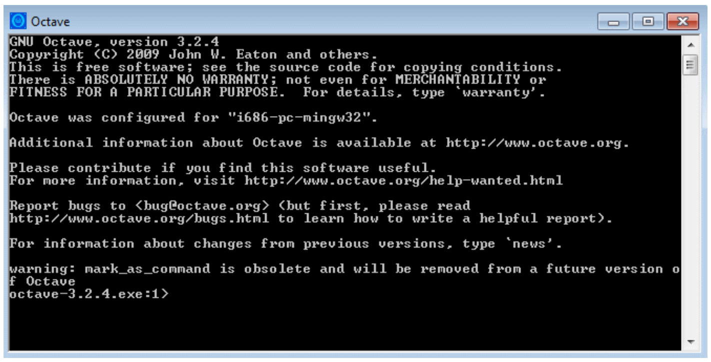
An introduction to OCTAVE is available at the official GNU home site for the project at:

`http://www.gnu.org/software/octave/doc/interpreter/Introduction.html`

But let's make a simple start if you haven't used the program before. . .

#1 If you are using a Lab computer, you might as well run Octave under Linux (the examples in this Prac are from Windows; but the interface is exactly the same in both versions). Start Octave by typing `octave` at the Linux shell prompt, or in Windows search for the octave icon.

When Octave starts, you simply get a terminal interface, as per the image on the right.

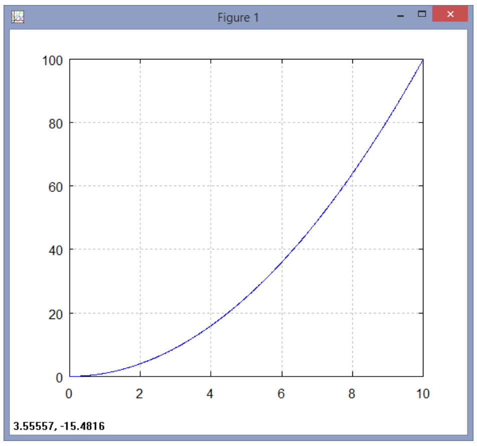


```
GNU Octave, version 3.2.4
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type `warranty'.

Octave was configured for "i686-pc-mingw32".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

For information about changes from previous versions, type `news'.

warning: mark_as_command is obsolete and will be removed from a future version o
f Octave
octave-3.2.4.exe:1>
```

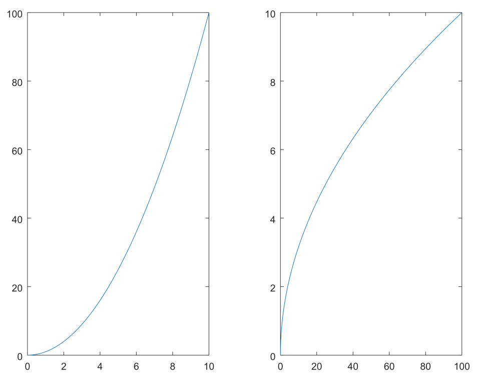
#2 I always think it's a good idea to have a particular project directory to keep project files. So, let's immediately go ahead and create a project directory. You can simply use Linux commands (in either Linux or Windows) to do this. You could mount a flash disk in your computer to keep your work in one place (although things tend to run faster from the hard-drive). The example below shows how to create a project directory:

```
# this is a comment - Oh, and incase you wanted to know,
# hash is for comments in OCTAVE

cd e:        # assuming E: your flash disk. In Linux
             # you'll need to do cd /media/flash0 or something
mkdir Prac0  # you'll see it says "ans = 1" if things worked OK
cd Prac0     # Now you're in the right directory
pwd          # this checks what directory you are in,
             # should say "ans = e:\Prac0"
ls           # let's double check that it is indeed an empty directory
```

#3 Now let's create a signal... type these commands to do so:



```
# x going from 0 to 10 in 0.1 increments
x = [0:0.1:10];
# y[i] is set to x[i]*x[i]
y = x .* x
# plot the graph (should get result
# shown on right)
plot(x,y); grid on;
```



#4 Multiple plots can be added to a single figure. This is useful for when you want to do side-by-side plotting to compare the graphs. The code below shows you how...

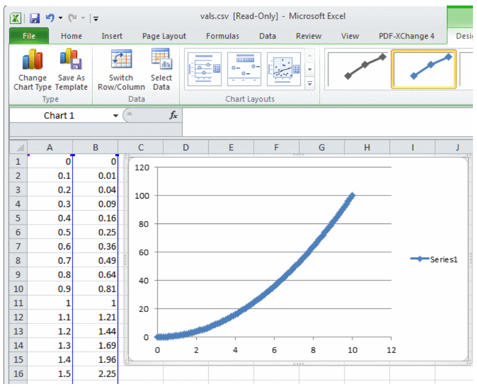
```
subplot (1,2,1);
#two separate plots horizontally together
plot(x,y);
subplot (1,2,2);
#select the second plot
plot(y,x);
```

#5 This is how we can go about saving the vectors. Save the vectors x and y to a CSV file:

```
save xvals.mat x  # save variable x to file called xvals.mat
save yvals.mat y  # save variable y to file called yvals.mat

# Also save both x and y to a .csv file as follows:
B = [x' y'];             # combine transposed variables
                         # (else it saves as row vectors)
csvwrite("vals.csv",B)   # save the B variable
clear B                  # free the temporary variable
```

#6 Load up the XLS file in OpenOffice or Microsoft Excel and graph the data.

You should be able to get a similar graph displayed in the spreadsheet application. It's probably a good idea to disable the markers so as not to display such thick lines.



	A	B
1	0	0
2	0.1	0.01
3	0.2	0.04
4	0.3	0.09
5	0.4	0.16
6	0.5	0.25
7	0.6	0.36
8	0.7	0.49
9	0.8	0.64
10	0.9	0.81
11	1	1
12	1.1	1.21
13	1.2	1.44
14	1.3	1.69
15	1.4	1.96
16	1.5	2.25

Part 2: Working with files

#1 Now access the Practical 1 assignment on Vula and download the `example_write.m` attachment. Place the file into your `Prac0` directory you made earlier.

#2 Open the `example_write.m` file in a text editor. You can do this in OCTAVE by typing in:

```
edit example_write.m
```

All that this program does, as you can see in the file, write the text `100 + 50/2` and its answer to a file.

#3 Run the program:

```
example_write.m
```

View the resultant `test1.txt` file.

#4 Copy the `example_write.m` to a new file called `pretty_table.m`. Implement the new program `pretty_table` so that it accepts a parameter `X` that is a 2-D array and writes it to a text file so that each value shows 2-digit precision and looks pretty. Note you don't need to hand anything in for this part – this is just to familiarize yourself with OCTAVE. An example pretty output is shown below:

```
| 1.00 | 0.00 | 0.00 |
| 0.00 | 1.00 | 0.00 |
| 0.00 | 0.00 | 1.00 |
```

Part 3: Measuring execution time with tic and toc

#1 White noise is often generated with GNU Octave's random number generator `rand()`, which generates uniformly distributed random values in the interval `[0,1)`. To create a sound wave of the white noise, the `wavwrite()` function in Octave is used. The `wavwrite()` function expects values in `[-1.0, 1.0)`, so the `rand()` output must be multiplied by 2 and shifted down by 1. To generate 10 seconds white noise sampled at 48 kHz, the following instruction are called:

```
white = rand(48000*10,1)*2-1;
```

And to generate a wave file for this noise, we use the `wavwrite()` function as follows:

```
wavwrite(white, 48000, 16, 'white_noise_sound.wav');
```

Alternatively one could plot the noise using Octave `plot(...)` or `hist(...)`, but first let's test this noise creation technique by doing the following:

Generate a 100 and 1000 seconds white noise sampled at 48 kHz. Play it back using Ubuntu's Movie Player or any audio player of your choice. Notice the sound.

#2 Write a function in a new script called `createwhiten.m` that implements a function with a `for` loop that generates a white noise signal comprising `N` duration in seconds. Assume that `N` will always be positive and a multiple of 10. The white noise must be sampled at either 48 kHz or 8 kHz. Name your function `createwhiten(...)`. You need to use the `rand()` function without arguments so that it will generate a single random value, and the main task is figuring out how to scale so that you create a suitable input to `wavwrite(...)` as explained above. Call the function and check output size as follows:

```
whiten = createwhiten(1000);
size(whiten);
```

should return:

```
ans = 48000000    1
```

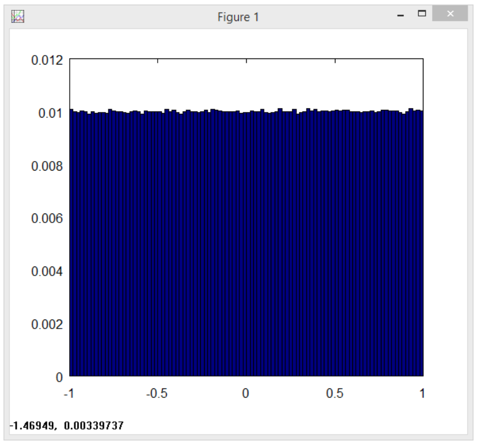
Check that the resulting wave / sound file gives the same sound as the `white_noise_sound.wav` generated above by generating the new sound file named `white_noise_sound2.wav` and playing it back.

```
wavwrite(whiten, 48000, 16, 'white_noise_sound2.wav');
```

#3 Confirm that you've created the sample correctly. Since it's a big signal, let's just look at the first 100 samples by plotting using a histogram function as follows:

```
hist(whiten, 100, 1);
```

should give image as shown on right.



#4 Time how long it took to execute the script. Use the tic and toc functions as follows. Note that I always put the call to tic on the same line just before the function I want to time because this tends to have less delay between the start of the timer and starting the function. Of course, it is good practice to put it all in a script file.

This goes in your report
[5 marks]

```
tic; white = rand(48000*1000, 1)*2 - 1; runtime = toc();
disp(strcat("It took: ", num2str(runtime*1000), " ms to run"));
```

I got the result:

```
It took:  1963.2 ms to run
```

Call the `createwhiten(...)` you created in step #2 that does the same thing as the `white = rand(48000*1000,1)*2 - 1;` statement that generate the white noise.

Measure the time it took for your `createwhiten(...)` function to run and show the timing difference (in milliseconds) and discuss the speed-up you have achieved (if any).

Part 4: Implementing a Correlation routine

Correlation is a useful statistical function for comparing two datasets to judge how similar or different they are. The correlation function returns a correlation coefficient, r, between -1 and 1. A value of 1 for r implies perfect positive correlation, i.e. the two datasets are the same. Correlation of 0 implies there is no correlation (the two datasets behave totally differently). A correlation of -1 indicates a total opposite - for example if you compare vectors x to $-x$ you get a correlation of -1. Generally if $|r| >=$ 0.8 there is strong correlation, between 0.5 and 0.8 moderate weak, less that 0.5 is weak (towards no) correlation.

The Pearson's correlation is calculated as shown below. For more details about this formula read up on `http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient`.

Put your code in the report
[10 marks]

$$r = \frac{\sum(XY) - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum(X^2) - \frac{\left(\sum X\right)^2}{N}\right)\left(\sum(Y^2) - \frac{\left(\sum Y\right)^2}{N}\right)}}$$

#1 Implement the above formula in a new `m` file, call it `mycorr.m`. Provide your code in your report. Marks will be awarded on elegance of your code.

#2 The cor function in OCTAVE performs a correlation operation. Compare your mycorr results to the build-in cor function. First read the noise wave file using wavread(...), then you could, for example, use the following code to test your mycorr(...) function as compared to Octave's cor(...):

```
x    = wavread('white_noise_sound.wav');
y    = x;
r1   = mycorr(x,y)
r2   = cor(x,y) # note that in some versions, this is called "corr"
disp(r2 - r1);

y(1) = 2; y(5) = -4; # i.e. fudge some of the value
r1   = mycorr(x,y)
r2   = cor(x,y)
disp(r2 - r1);

x    = rand(1,10); y = rand(1,10);
r1   = mycorr(x,y)
r2   = cor(x,y)
disp(r2 - r1);
```

Generate sample sizes of varying sizes: 100, 1000 and 10000 samples. Do a table listing sample sizes vs. mycorr speed vs. corr speed. Indicate the average speed-up of corr to mycorr.

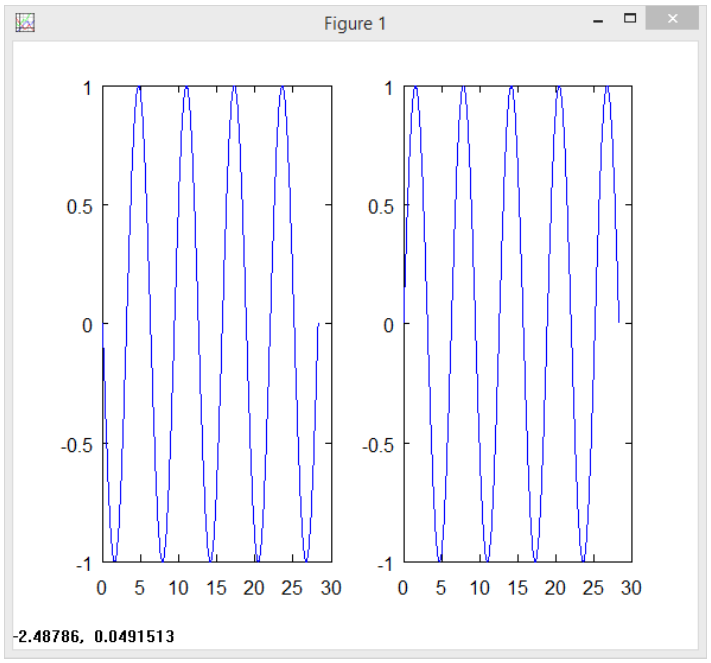
Do table of sample size vs time etc. requested
[5 marks]

#3 For the last experiment, we want to compare signals shifted in time. Generate sin curves of varying frequency and sampling sizes (again sample sizes 100, 1000 and 10000 samples). Compare samples of the same sizes that are shifted in time, e.g. if A uses x[1:100] then B might use x[11:110], in which case B is shifted in time by 10 samples.

For this step only use cor to save time. In your report, discuss what you expect the correlation of the identical but shifted signals would be. Run tests to confirm / verify your hypothesis. Provide screen shot plots of some signals you compared.

Show some code excerpts, correlation values and screen-shots to indicate what you did.
[10 marks]

Here's an example plot of waveforms you might try. . .



END OF ASSIGNMENT

OCTAVE Libraries to install

When installing on Windows you'll be asked for certain libraries to include in the installation. If you have lots of disk space, just install the whole Octave Forge collection. Otherwise, make sure you install the following ones, otherwise you may have difficulties in plotting or in running certain functions mentioned in howework tasks and pracs.

Libraries needed from the Octave Forge collection:

audio ; control ; data-smoothing ; fixed ; ga ; gnuplot ; image ; integration ; oct2mat ; plot ; signal ; sockets ; specfun ; splines ; statistics ; strings