Assignment 2, Digital Signal processing: FIR filters

Bernd Porr

2018

The task of this assignment is to filter an ECG with FIR filters. Before you record from yourself please read the info-sheet and sign the consent form.

1 ECG filtering

Form groups of 2 students. Submit one report. There are different amplifiers with different gains and converter boards available. When you record the ECG make sure that you take a note of the gain, bits, etc. If you record with a one channel amplifier record Einthoven II (google it what it means). Upload the files onto the ECG forum on moodle or e-mail it to yourself. Give the file a name which anonymises the ECG. Use only numbers, characters and the underscore for your filename.

Every group of two has to work on a different ECG. If your initial recording has been too noisy just record another one. You can minimise artefacts by lying down and/or placing the electrodes on the shoulders / hips instead of wrists/ankles. In Python load the file with the command *numpy.loadtxt('myecg.dat'*. You'll get a numpy matrix with time-stamps in ms in the first column and 3 columns of ecg with raw unsigned integer numbers. Choose the channel with the highest *amplitude*. Also note that the ECG devices will be around all Oct / Nov so there is plenty of time to record your ECG. You can use one of the example ECGs from moodle to kick start your work.

The Python code must be *object oriented* (i.e classes). Decide which Python classes to create to structure your code and make it readable.

Make sure you scale the ECG properly so that it's displayed at the right amplitude.

1. Create a Python FIR filter class which implements an FIR filter which has a method of the form `value dofilter(value)` where both the value argument and return value are *scalars* and not vectors so that it could be used in a realtime system. The constructor of the class takes the coefficients as its input:

```
class FIR_filter:
```

```
def __init__(self,_coefficients):
# your code here

def dofilter(self,v):
# your code here
    return result
```

[30%]

2. Create a Python class which inherits or has an instance of the FIR class above from 1 but is also able to calculate its own coefficients *numerically* (=ifft) were the user can specicy the frequency response in a user friendly way. Think of methods which can manipulate the frequency response so that the user can specify a high/low/band/stopband filter. Then Filter the ECG by removing the 50Hz interference and the DC. Let the Python class automatically decide how many taps are needed. Also this filter class needs to be able to perform realtime processing (i.e. scalar in and scalar out for the filter operation). [30%].

2 ECG heartrate detection

The task is to detect the *momentary* heart rate $r(t)$ over a longer period of time. For example, after exercise you should see a slow decay of the heart rate to the baseline of perhaps 60 beats per minute. It is not the average heart rate but the frequency derived from the times *between adjacent heartbeats*.

Record a separate long ECG over a couple of minutes. Ask the person to breathe in quite deeply which will change the heart rate or anything which changes the heart rate for example sending the person up to level 8 and back.

1. Create a matched filter Python class which detects the individual heartbeats in the ECG. Think of the optimum length of the filter and give a reason. Optimise the detection by pre-filtering both the ECG and the template. Remember the detection works best when both the signal and the templates are completely DC free. Use your own FIR filter class for it and square the output of the matched filter to increase the SNR.

   Instead of taking your own ECG you can also use functions which look like an ECG. Very popular are the so called Daubechies wavelets which look like an ECG r-peak. Python can generate these for you.

   Test your detector with an ECG which is clean and one where there are a lot of artefacts. [20%]

2. Implement a Python class which uses the output of the matched filter to calculate the *momentary* heart rate $r(t)$ over time (not the average!) by measuring the intervals between the detected hearbeats over the whole

period of the ECG. Detect the heartbeats by employing a threshold (adaptive, if required). Generally use any heuristics to weed our wrong detection, for example that a heartbeat is usually below 200bpm and above 30bpm. [20%]

Every report must be based on different ECG recordings. Please keep it short but it should make clear what you have done and why you have done it. Include the Python code (object oriented) as well as plots of the ECGs (timedomain) and their frequency representation (with proper lables). If necessary enhance the plots with InkScape and remember to use vector based image formats, for example EPS, SVG, PDF or EMF and not pixel based formats for the report. These figures need to be in the report not attached separately. Also, show zoomed in ECG traces of one heartbeat so that it is possible to identify the different parts of a single heartbeat (see Fig. 1) and that it's possible to check if it's still intact.

No high level Python functions except of FFT/IFFT and the window functions are allowed. Any use of "lfilter" or "firwin" will result in zero marks for the whole report. If the same ECG data is spotted in two different reports this will result in zero marks.

Hand in your report at the teaching office (Room 620 James Watt Building South). Deadline is 12th November 3pm.

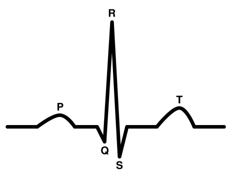

Figure 1: A normal ECG with P,Q,R,S,T waves (taken from Wikipedia)