Heartbeat Detection

Practical Assignment 2

Informatics 1 for Biomedical Engineers

Graz University of Technology

Abstract

Data collected via any means frequently suffers from inconsistencies and errors. Cleaning the data is thus an important step during the analysis process. One common data error is to detect outliers—entries that are far from the average. There are numerous ways to detect these. For data series one simple approach is to look at means and standard deviations. Yet, as the data on the PTBDB is already sanitised we use this method to detect heartbeats.

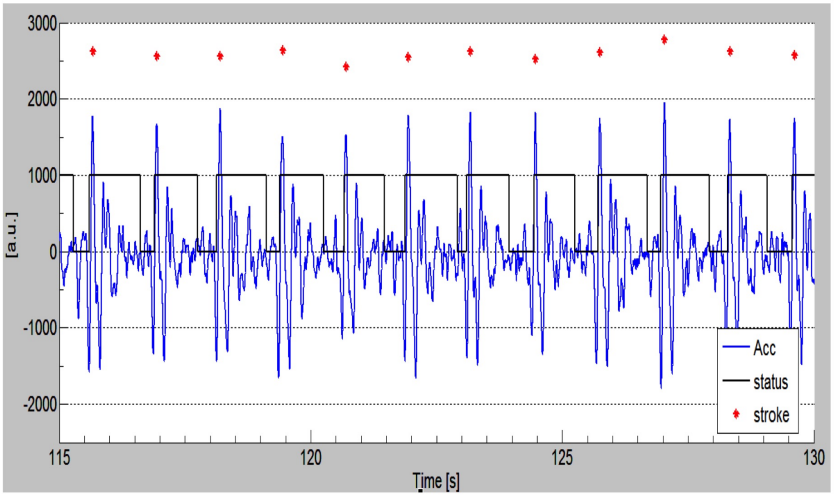


Image from: -
http://www.electronicsweekly.com/market-sectors/medical-electronics/mems-device-can-detect-volume-heart-beat-2015-05/

1. Tasks

This assignment builds on-top of the previous task. We will use the output of assignment 0 as input for assignment 1. If you are unsure your code was correct feel free to use the reference solution to generate your data[1].

Again, the data provided for this course are from the The PTB Diagnostic ECG Database[2].

Our program should detect heartbeats. To do so we define a threshold. Whenever the signal exceeds this value we mark the frame index as detected heartbeat. Then, we need to wait for the signal value to drop below the threshold. Once the values are back to normal levels we wait for the next spike. The threshold (θ) is defined as the period mean ($\bar{x}$) plus two times the standard deviation (σ) (see Equation 1). Whilst this approach has flaws it is very straight forward and sufficient for the task at hand.

$$\theta = \bar{x} + 2 * \sigma \tag{1}$$

To calculate the threshold value two functions need to be defined. First, to calculate the mean and second, to calculate the standard deviation. Note that to calculate σ you need the period mean. Call your own mean function here. Do **NOT** write the same code twice.

To keep track we store all detected heartbeats (their indexes) in a simple list. At the end we save these as CSV.

1.1. Open a data package (1 pt)

Open the pickle you created in assignment 1. Again, if you are unsure your implementation was correct, then create this data with the reference solution. The name of the data file your program is supposed to open is defined via the first (and only) command line parameter!

1.2. Select sensor (2pt)

The heartbeat detection should run on only a single signal. It can be interesting to run the program with the other sensors too. This way you might detect the problems our detection algorithm has. The name of the signal is passed as the second command line parameter. You will need to look

[1] `https://palme.iicm.tugraz.at/wiki/Info1BM/Assignment_1`

[2] `https://www.physionet.org/physiobank/database/ptbdb/`

through your loaded data structure and check which index has the wanted name.

1.3. Functions for mean and std (3+3pt)

Functions help you to keep your code tidy. To detect outliers we need both the mean and the standard deviation of our values. But to calculate the std you already need the mean. Thus, it would be smart to create a function to calculate this for you. So if something goes wrong you only need to edit your code at one point instead of two.

Please note that the definition and usage of functions is mandatory for this task! However, how you name them is up to you. (Coding standard still applies!)

1.4. Heartbeat detection (5pt)

Once you have calculated the threshold value (using the Equation 1) you can start to detect heartbeats. Whenever your signal value exceeds this value you store the index of this event in a list. After that you need to wait until the value drops below it before you start checking again. Here is an example:

```
heartbeats = []
threshold = 2
sensor_values = [1,2,3,4,5,4,3,4,5,3,1,2,3]

### DO THE CHECKS ###

# heartbeats = [2,12]
```

Note that the stored index is the one where the value (3) is greater than the threshold (2). Your check should thus use a $>$ and not a $\geqslant$.

1.5. Store CSV (1pt)

Once you have detected your heartbeats you need to store them to the disk. Create a file called *<dataset>.csv* (so for instance *s0010_re.csv*). Afterwards write your list to this file in a CSV. The delimiter should be a single comma (','). Make sure you do not escape your values as strings and do not add any other data to this file. Here is a simplified example:

```
### CORRECT CSV DATA:
1,100,300,500,710,899
```

```
### INCORRECT CSV DATA:
"1","100","300","500","710","899"
```

As comma separated value (CSV) files are plaintext you can open and check them with any plaintext editor!

2. Additional Notes

Remember, the signal data is stored in the form of 16 bit integers. When you read the data make sure to not read too many bytes as integers are commonly 32 or 64 bit. The *struct* package is meant to deal with these issues. Though, we do not enforce the usage as there are other options to solve this problem without the use of any package.

2.1. Command Line Parameters

Your program should work with command line parameters. Execute it with the command *python assignment_2.py <pickle_file_name> <sensor_name>*. Your submission should expect the files in the same directory as the python source!

2.2. Output

Make sure the file your script creates has the same identifier as the input (as defined by the command line parameter). Calling your program with *assignment_2.py s0010_re.p* should create a file called *s0010_re.csv*! (Hint: String slicing)

2.3. Debug Output

During development and testing debug outputs can be quite helpful. For finished programs on the other hand think about what information is vital to the user. In most cases it is only important to report when something went wrong. This means as long as everything goes as planned your submissions should **not** create any output to the console (no print statements). If you detect an error, you can broadcast it and exit the program. But please note that error handling is not part of this assignment.

3. File Headers

All python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please feel free to copy and paste the example and change its contents to your data. (Any comments will not be subject for plagiarism checks)! Note: Depending on your reader you may have to manually fix the whitespaces and indentation!
Example:

```
##############################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Assignment only has 1 file...
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
##############################################################################
```

4. Allowed packages

The following packages are allowed to use in this assignment:

- sys
- pickle

5. Restrictions

- Do not add any bloat to your submission
  - No useless code
  - No debug output
  - No extra files
  - Try to avoid *.__MACOSX* or *.DS_STORE* folders in your submission archives.
- Use built-in functions where possible
  - except when explicitly stated—such as Task 1.3
- Do NOT load extra packages (no imports) except those listed in section 4
- Do NOT require any extra command line arguments!

6. Coding Standard

For this lecture and the practicals we use PEP 8 [3] as a coding standard guideline. Please note that whilst we do not strictly enforce all these rules we will not tolerate messy or unreadable code. Please focus especially on the following three aspects.

Language Programming is generally done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. Whilst grammatical or typographical errors won't be penalised in any way we ask you to double-check your code before submitting.

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in python 3 (whilst allowing more freedom in python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

[3] `https://www.python.org/dev/peps/pep-0008/`

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all use a single underscore (_) for its name.

Max 72 characters. Do not have lines longer than 72 characters. Whilst PEP 8 allows 79 in some cases we ask you to use the lower cap of 72 characters per line. Please note that this includes indentation.

Not following this coding standard will result in a deduction from your achieved points!

7. Self testing

Naturally, you may want to test your program prior to submitting it. Instructions regarding how to do that can be found on the Wiki [4].

[4] `https://palme.iicm.tugraz.at/wiki/Info1BM#Assignment_Self_Tests`

8. Automated Tests

We will test your submissions automatically. The test machines will have the latest anaconda version on the 5th of October 2016. Your file will be executed with the following command:

```
>python assignment_2.py s0010_re.p i
```

Please make sure your submission follows all the restrictions defined in this document. Your program will have a limited runtime of 2 minutes. If your submission exceeds this threshold then it will be considered non-executable. Please note that this is a very generous amount of time for any of the tasks in this course.
If your submission fails any of the automated tests we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

We will be testing the following aspects:

- General coding errors (Does the program even run?)
- Do the files contain the correct comment headers?
- Your output with the provided input (*s0010_re*.p)
- Your output with different input

9. Submission

9.1. Deadline

6^{th} of December 2016 at 23:59.

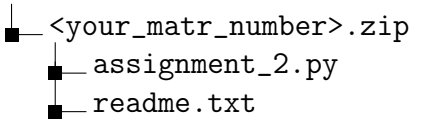
Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will automatically be pushed back for 24 hours.
If for whatever reason you are unable to submit your submission prior to the deadline please contact your tutor BEFORE the deadline.

9.2. Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your python source files please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. This is important to us as immediate feedback so we can adjust the tutor sessions. Please note that all submissions will be anonymised prior to being read. Hence please feel free to be honest (but do stay polite).
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every source file
- Coding standard upheld
- You do not hard-code the file name and the frame amount!

9.3. Your submission file



Do not submit any additional files. You do not need to submit the CSV file (.csv) your program creates!