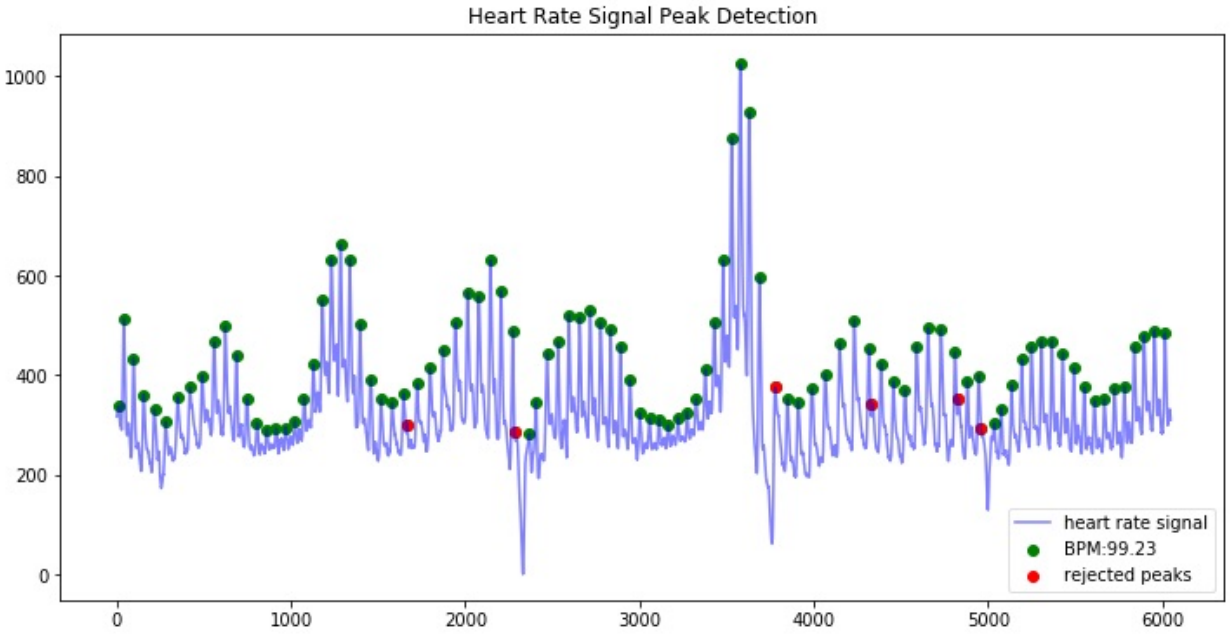
Python Heart Rate Analysis Toolkit Documentation

Release 1.0

Paul van Gent

Nov 06, 2018

.



Welcome to the documentation of the HeartPy, Python Heart Rate Analysis Toolkit. The toolkit is designed to handle (noisy) PPG data collected with either PPG or camera sensors.

- The toolkit was presented at the Humanist 2018 conference in The Hague (see paper here).
- A technical paper about the functionality is available here

Please cite one or both of these papers when using the toolkit in your research!

The documentation will help you get up to speed quickly. Follow the *Quickstart Guide* guide for a general overview of how to use the toolkit in only a few lines of code. For a more in-depth review of the module's functionality you can refer to the papers mentioned above, or the *Heart Rate Analysis* overview.

CHAPTER 1

Note on using it in scientific research

Support is available at P.vanGent@tudelft.nl. When using the toolkit in your scientific work: please include me in the process. I can help you implement the toolkit, and the collaboration will also help improve the toolkit so that it can handle more types of data in the future.

CHAPTER 2

Index

2.1 Quickstart Guide

2.1.1 Installation

pip

```
python -m pip install heartpy
```

github

Download the latest release here

```
python -m pip install <.whl file>
```

Don't forget to replace the <.whl file> with the filename of the .whl file in the release

from source

Clone the repository from this link

```
python setup.py install
```

2.1.2 Basic Example

Import the *HeartPy* module and load a file

```
import heartpy as hp

hrdata = hp.get_data('data.csv')
```

This returns a `numpy.ndarray`.

Analysis requires the sampling rate for your data. If you know this *a priori*, supply it when calling the *process()* function, which returns a *dict{}* object containing all measures:

```
import heartpy as hp

data = hp.get_data('data.csv') #data.csv is sampled at 100Hz
working_data, measures = hp.process(data, 100.0)
```

process(dataset, fs, windowsize=0.75, report_time=False, calc_freq=False, freq_method='welch', interp_clipping=True, clipping_scale=True, interp_threshold=1020, hampel_correct=False, bpmmin=40, bpmmax=180, reject_segmentwise=False, measures = {}, working_data = {})

requires two arguments:

- **dataset:** An 1-dimensional list, numpy array or array-like object containing the heart rate data;
- **fs**: The samplerate of the signal in Hz;

Several optional arguments are available:

- **windowsize:** _optional_ *windowsize* is the window size used for the calculation of the moving average. The windowsize is defined as *windowsize * samplerate*. Default windowsize=0.75.
- **report_time:** _optional_ whether to report total processing time of process() loop.
- **calc_fft:** _optional_ whether to calculate frequency domain measures. Default = false Note: can cause slowdowns in some cases.
- **calc_freq:** _optional_ whether to calculate frequency domain measures. Default = false Note: can cause slowdowns in some cases.
- **freq_method:** _optional_ method used to extract the frequency spectrum. Available: 'fft' (Fourier Analysis), 'periodogram', and 'welch' (Welch's method), Default = 'welch'
- **interp_clipping:** if True, clipping parts of the signal are identified and the implied peak shape is interpolated. Default=True
- **clipping_scale:** whether to scale the data priod to clipping detection. Can correct errors if signal amplitude has been affected after digitization (for example through filtering). Default = false
- **interp_threshold**: the amplitude threshold beyond which will be checked for clipping. Recommended is to take this as the maximum value of the ADC with some margin for signal noise (default 1020, default ADC max 1024)
- **hampel_correct:** whether to reduce noisy segments using large median filter. Disabled by default due to computational complexity, and generally it is not necessary. Default = false.
- **bpmmin:** minimum value to see as likely for BPM when fitting peaks. Default = 40
- **bpmmax:** maximum value to see as likely for BPM when fitting peaks. Default = 180
- **reject_segmentwise:** whether to reject segments with more than 30% rejected beats. By default looks at segments of 10 beats at a time. Default = false.
- **measures:** measures dict in which results are stored. Custom dictionary can be passed, otherwise one is created and returned.
- **working_data:** working_data dict in which results are stored. Custom dictionary can be passed, otherwise one is created and returned.

Two `dict{}` objects are returned: one working data dict, and one containing all measures. Access as such:

```
import heartpy as hp

data = hp.get_data('data.csv')
fs = 100.0 #example file 'data.csv' is sampled at 100.0 Hz

working_data, measures = hp.process(data, fs, report_time=True)

print(measures['bpm']) #returns BPM value
print(measures['rmssd']) # returns RMSSD HRV measure

#You can also use Pandas if you so desire
import pandas as pd
df = pd.read_csv("data.csv", names=['hr'])
#note we need calc_freq if we want frequency-domain measures
working_data, measures = hp.process(df['hr'].values, fs, calc_freq=True)
print(measures['bpm'])
print(measures['lf/hf'])
```

2.1.3 Getting Data From Files

The toolkit has functionality to open and parse delimited .csv and .txt files, as well as matlab .mat files. Opening a file is done by the `get_data()` function:

```
import heartpy as hp

data = hp.get_data('data.csv')
```

This returns a 1-dimensional `numpy.ndarray` containing the heart rate data.

`get_data(filename, delim = ',', column_name = 'None')` requires one argument:

- **filename:** absolute or relative path to a valid (delimited .csv/.txt or matlab .mat) file;

Several optional arguments are available:

- **delim** _optional_: when loading a delimited .csv or .txt file, this specifies the delimiter used. Default delim = ‘,’;
- **column_name** _optional_: In delimited files with header: specifying column_name will return data from that column. Not specifying column_name for delimited files will assume the file contains only numerical data, returning np.nan values where data is not numerical. For matlab files: column_name specifies the table name in the matlab file.

Examples:

```
import heartpy as hp

#load data from a delimited file without header info
headerless_data = hp.get_data('data.csv')

#load data from column labeles 'hr' in a delimited file with header info
headered_data = hp.get_data('data2.csv', column_name = 'hr')

#load matlab file
matlabdata = hp.get_data('data2.mat', column_name = 'hr')
#note that the column_name here represents the table name in the matlab file
```

2.1.4 Estimating Sample Rate

The toolkit has a simple built-in sample-rate detection. It can handle ms-based timers and datetime-based timers.

```
import heartpy as hp

#if you have a ms-based timer:
    mstimer_data = hp.get_data('data2.csv', column_name='timer')
fs = hp.get_samplerate_mstimer(mstimer_data)
    print(fs)

#if you have a datetime-based timer:
    datetime_data = hp.get_data('data3.csv', column_name='datetime')
fs = hp.get_samplerate_datetime(datetime_data, timeformat='%Y-%m-%d %H:%M:%S.%f')
    print(fs)
```

`get_samplerate_mstimer(timerdata)` requires one argument:

- **timerdata:** a list, numpy array or array-like object containing ms-based timestamps (float or int).

`get_samplerate_datetime(datetimedata, timeformat = '%H:%M:%S.f')` requires one argument:

- **datetimedata:** a list, numpy array or array-like object containing datetime-based timestamps (string);

One optional argument is available:

- **timeformat** _optional_: the format of the datetime-strings in your dataset. Default timeformat='%H:%M:%S.f', 24-hour based time including ms: 21:43:12.569.

2.1.5 Plotting Results

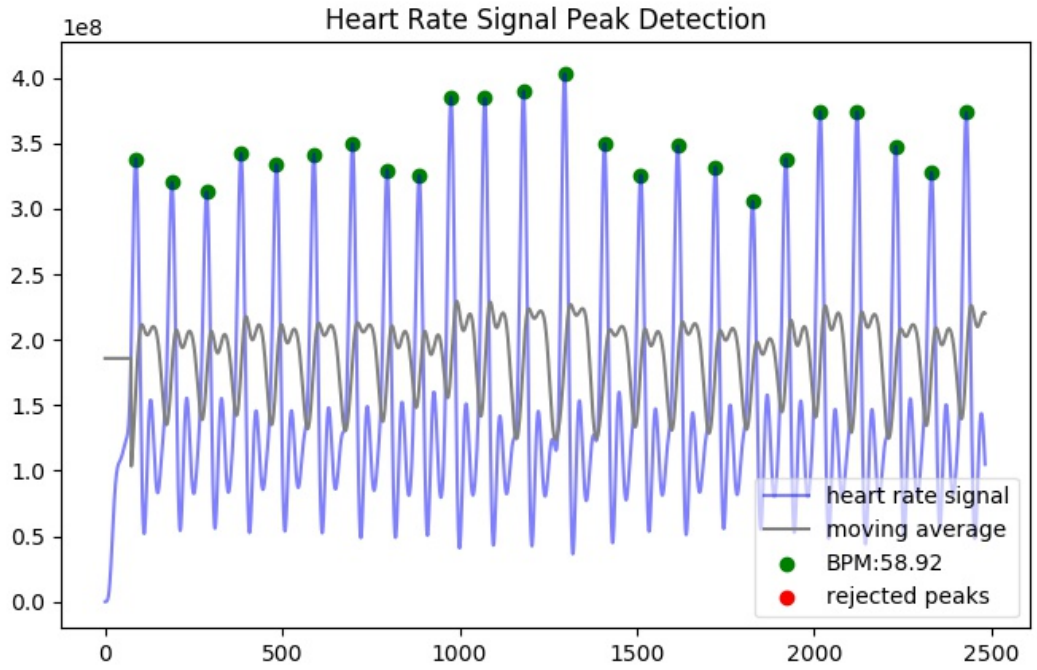
A plotting function is included. It plots the original signal and overlays the detected peaks and the rejected peaks (if any were rejected).

Example with the included *data.csv* example file (recorded at 100.0Hz):

```
import heartpy as hp

data = hp.get_data('data.csv')
working_data, measures = hp.process(data, 100.0)
hp.plotter(working_data, measures)
```

This returns:



`plotter(working_data, measures, show = True, title = 'Heart Rate Signal Peak Detection')` has two required arguments:

- **working_data** The working data `dict{}` container returned by the `process()` function.
- **measures** The measures `dict{}` container returned by the `process()` function.

Several optional arguments are available:

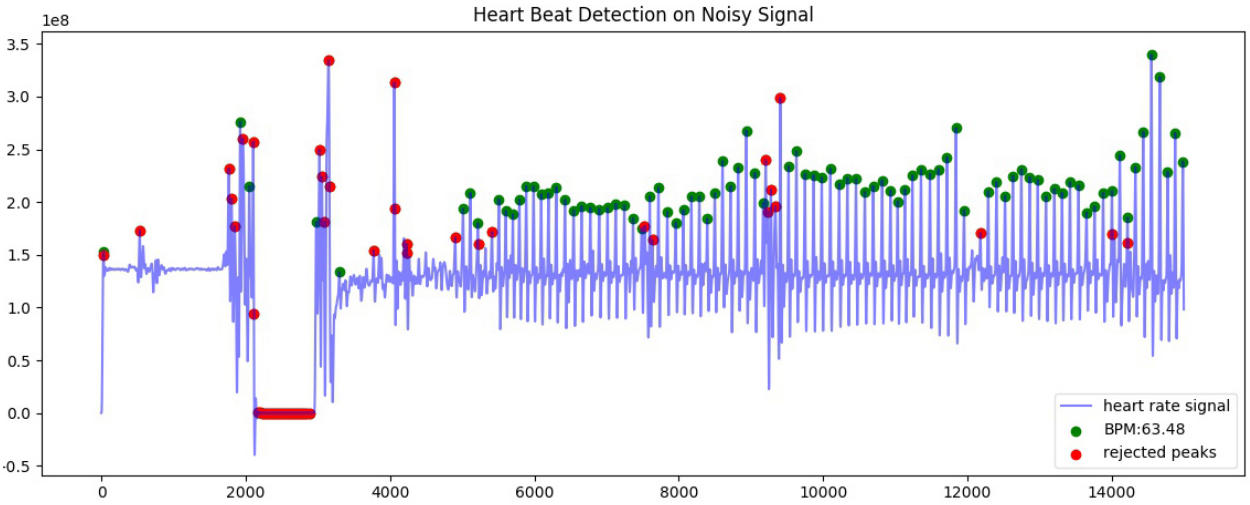
- **show** _optional_: if set to True a plot is visualised, if set to False a matplotlib.pyplot object is returned. Default show = True;
- **title** _optional_: Sets the title of the plot. If not specified, default title is used.

Examples:

```
import heartpy as hp
hrdata = hp.get_data('data2.csv', column_name='hr')
timerdata = hp.get_data('data2.csv', column_name='timer')

working_data, measures = hp.process(hrdata, hp.get_samplerate_mstimer(timerdata))

#plot with different title
hp.plotter(working_data, measures, title='Heart Beat Detection on Noisy Signal')
```



Measures are only calculated for non-rejected peaks and intervals between two non-rejected peaks. Rejected detections do not influence the calculated measures.

By default a plot is visualised when plotter() is called. The function returns a matplotlib.pyplot object if the argument show=False is passed:

```
working_data, measures = hp.process(hrdata, hp.get_samplerate_mstimer(timerdata))
plot_object = hp.plotter(working_data, measures, show=False)
```

This returns:

```
<module 'matplotlib.pyplot' [...]>
```

Object can then be saved, appended to, or visualised:

```
working_data, measures = hp.process(hrdata, hp.get_samplerate_mstimer(timerdata))
plot_object = hp.plotter(working_data, measures, show=False)

plot_object.savefig('plot_1.jpg') #saves the plot as JPEG image.

plot_object.show() #displays plot
```

2.2 Heart Rate Analysis

A complete description of the algorithm can be found in: <ref embedded paper>.

2.2.1 Background

The Python Heart Rate Analysis Toolkit has been designed mainly with PPG signals in mind. The Raspberry Pi and the Arduino platforms have enabled more diverse data collection methods by providing affordable open hardware platforms. This is great for researchers, especially because traditional ECG may be considered to invasive or too disruptive for experiments.

2.2.2 Measuring the heart rate signal

Two often used ways of measuring the heart rate are the electrocardiogram (ECG) and the Photoplethysmogram (PPG). Many of the online available algorithms are designed for ECG measurements. Applying an ECG algorithm (like the famous Pan-Tompkins one[1]) to PPG data does not necessarily make sense. Although both the ECG and PPG are measures for cardiac activity, they measure very different constructs to estimate it.

The ECG measures the electrical activations that lead to the contraction of the heart muscle, using electrodes attached to the body, usually at the chest. The PPG uses a small optical sensor in conjunction with a light source to measure the discoloration of the skin as blood perfuses through it after each heartbeat. This measuring of electrical activation and pressure waves respectively, leads to very different signal and noise properties, that require specialised tools to process. This toolkit specialises in PPG data.

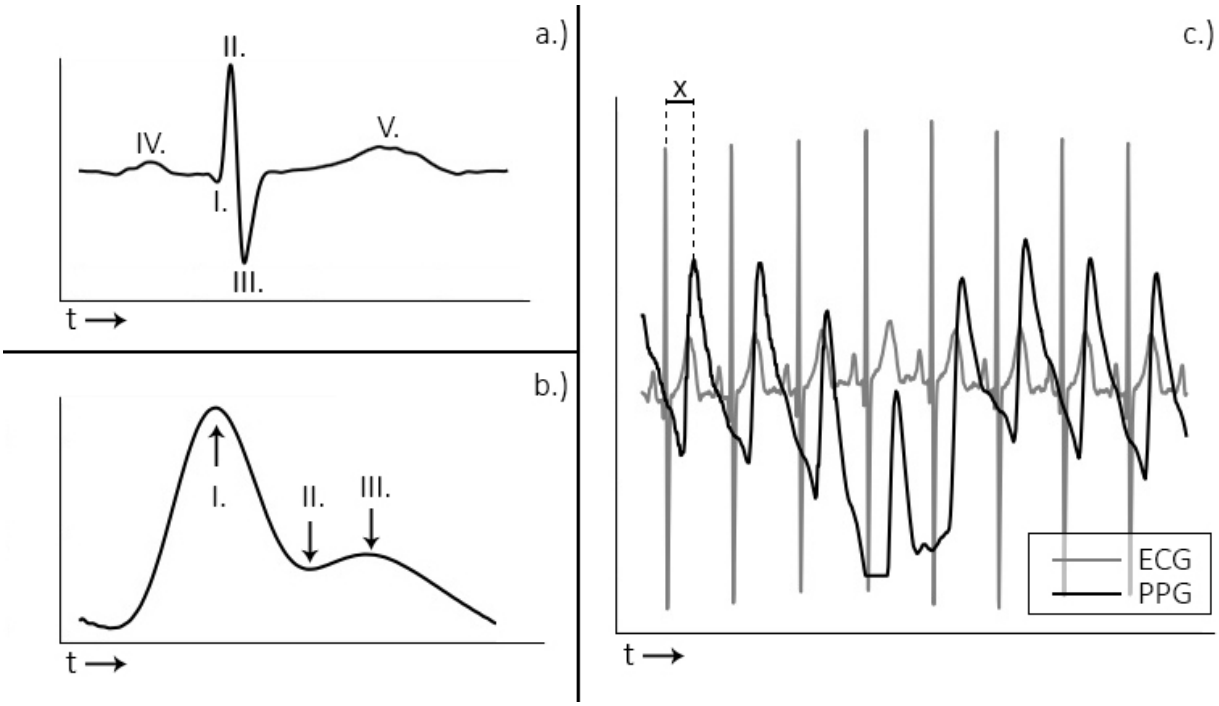


Figure 1: a. and b. display the ECG and PPG waveform morphology, respectively. The ECG is divided into distinct waves (a, I-V), of which the R-wave (a, II) is used for heart beat extraction. With the PPG wave, the systolic peak (b, I) is used. The plot in c. shows the relationship between ECG and PPG signals.

Most notably in the ECG is the QRS-complex (Fig 1a, I-III), which represents the electrical activation that leads to the ventricles contracting and expelling blood from the heart muscle. The R-peak is the point of largest amplitude in the signal. When extracting heart beats, these peaks are marked in the ECG. Advantages of the ECG are that it provides a good signal/noise ratio, and the R-peak that is of interest generally has a large amplitude compared to the surrounding data points (Fig 1c). The main disadvantage is that the measurement of the ECG is invasive. It requires the attachment of wired electrodes to the chest of the participant, which can interfere with experimental tasks such as driving.

The PPG measures the discoloration of the skin as blood perfuses through the capillaries and arteries after each heartbeat. The signal consists of the systolic peak (Fig 1-b, I), dicrotic notch (II), and the diastolic peak (III). When extracting heart beats, the systolic peaks (I) are used. PPG sensors offer a less invasive way of measuring heart rate

[1] Pan, J., & Tompkins, W. J. A simple real-time QRS detection algorithm. IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, BME-32(3), 230–236, 1985. https://doi.org/10.1109/IEMBS.1996.647473

data, which is one of their main advantages. Usually the sensors are placed at the fingertip, earlobe, or on the wrist using a bracelet. Contactless camera-based systems have recently been demonstrated[2],[3],[4]. These offer non-intrusive ways of acquiring the PPG signal. PPG signals have the disadvantages of showing more noise, large amplitude variations, and the morphology of the peaks displays broader variation (Figure 2b, c). This complicates analysis of the signal, especially when using software designed for ECG, which the available open source tools generally are.

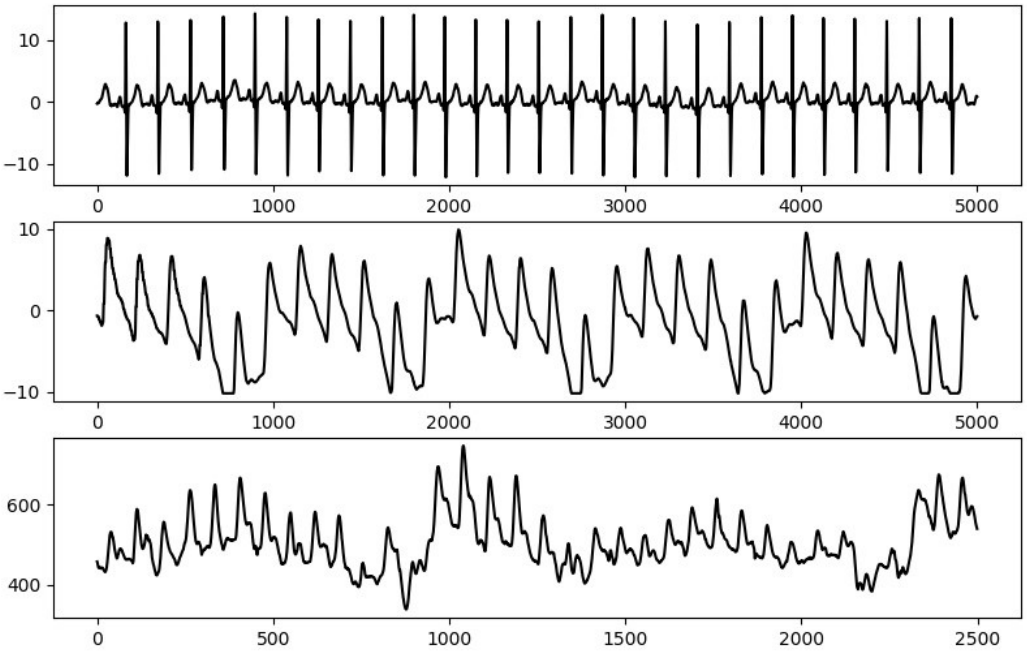


Figure 2 – The ECG signal (a.) shows a strong QRS complex together with little amplitude variation. The PPG signal measured simultaneously while the patient is at rest in a hospital bed (b.) shows some amplitude variation but relatively stable morphology. When measuring PPG in a driving simulator using low-cost sensors (c.), strong amplitude and waveform morphology variation is visible.

2

25. Sun, S. Hu, V. Azorin-Peris, R. Kalawsky, and S. Greenwald, "Noncontact imaging photoplethysmography to effectively access pulse rate variability," J. Biomed. Opt., vol. 18, no. 6, p. 61205, 2012.

3

13. Lewandowska, J. Ruminsky, T. Kocejko, and J. Nowak, "Measuring Pulse Rate with a Webcam - a Non-contact Method for Evaluating Cardiac Activity," in Proceedings of the Federated Conference on Computer Science and Information Systems, 2011, no. January, pp. 405–410.

4

6. Bousefsaf, C. Maaoui, and a. Pruski, "Remote detection of mental workload changes using cardiac parameters assessed with a low-cost webcam," Comput. Biol. Med., vol. 53, pp. 1–10, 2014.

2.2.3 On the Accuracy of Peak Position

When analysing heart rate, the main crux lies in the accuracy of the peak position labeling being used. When extracting instantaneous heart rate (BPM), accurate peak placement is not crucial. The BPM is an aggregate measure, which is calculated as the average beat-beat interval across the entire analysed signal (segment). This makes it quite robust to outliers.

However, when extracting heart rate variability (HRV) measures, the peak positions are crucial. Take as an example two often used variability measures, the RMSSD (root mean square of successive differences) and the SDSD (standard deviation of successive differences). Given a segment of heart rate data as displayed in the figure below, the RMSSD is calculated as shown. The SDSD is the standard deviation between successive differences.

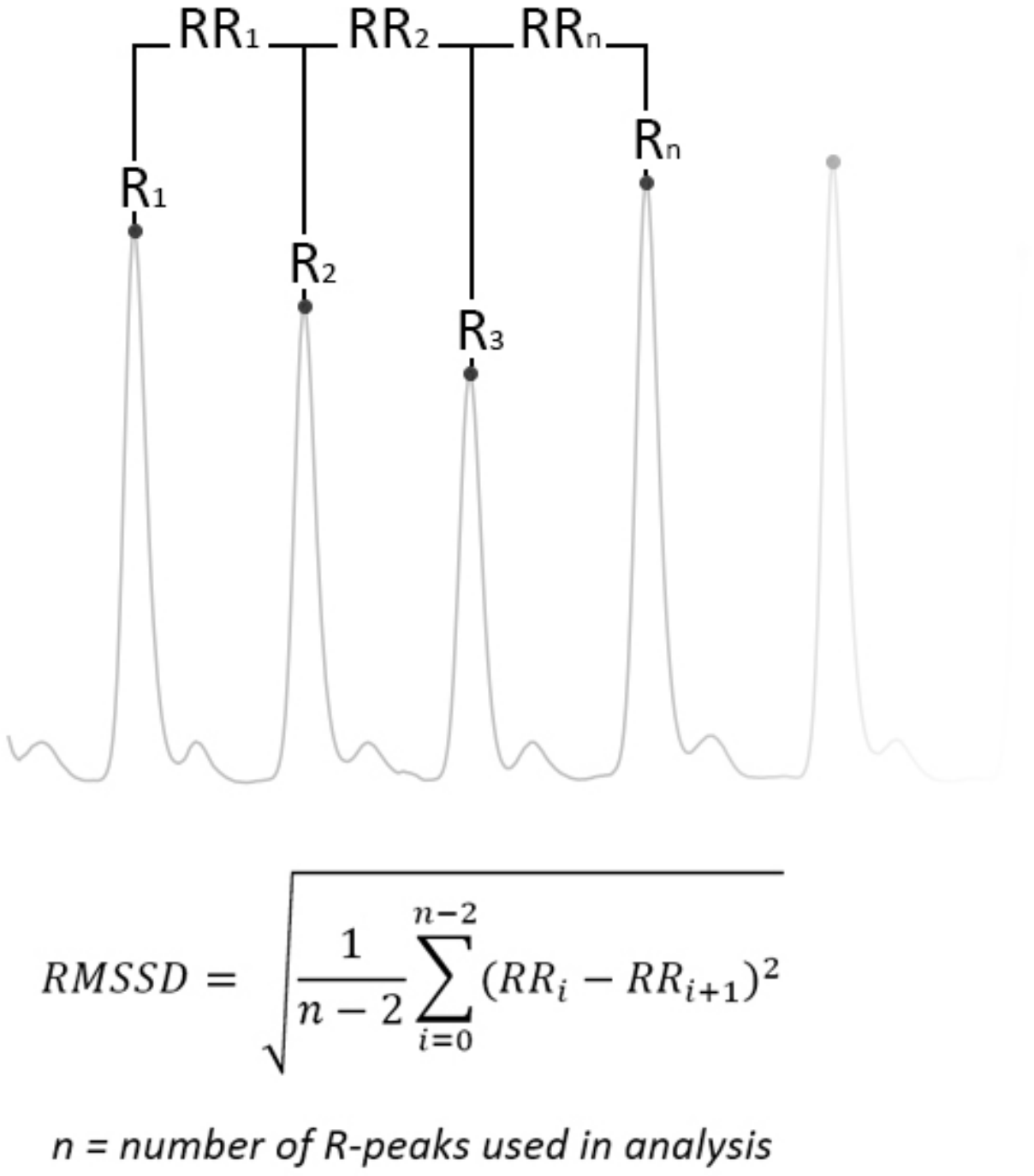


Figure 3 - Image displaying the desired peak detection result, as well as the calculation of the RMSSD measure. The SDSD measure is the standard deviation between successive differences

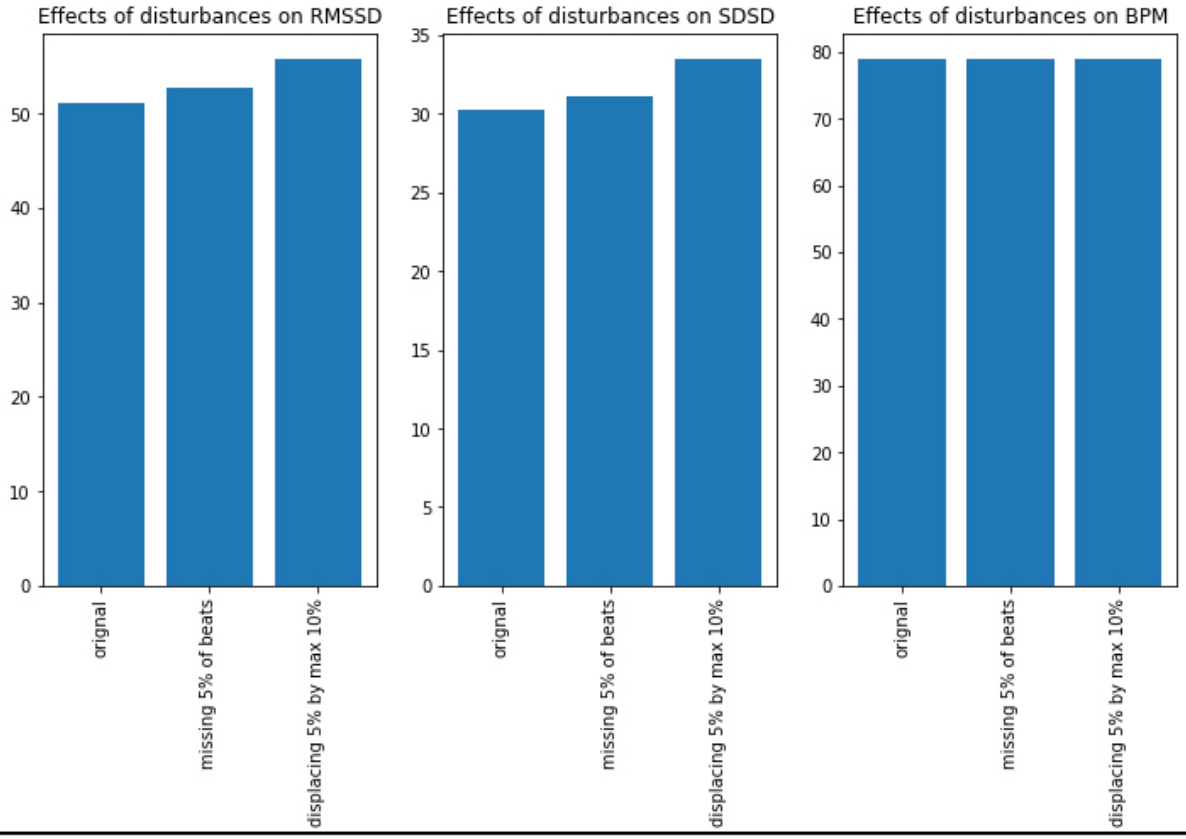
Now consider that two mistakes are possible: either a beat is not detected at all (missed), or a beat is placed at an incorrect time position (incorrectly placed). These will have an effect on the calculated HRV output measures, which are highly sensitive to outliers as they are designed to capture the slight natural variation between peak-peak intervals in the heart rate signal!

To illustrate the problem we have run a few simulations. We took a sample of a heart rate signal which was annotated manually, and introduced two types of errors:

- We randomly dropped n% of peaks from the signal, than re-ran the analysis considering only intervals between two peaks where no missing value occurred in between.
- We introduced a random position error (0.1% - 10% of peak position, meaning between about 1ms and 100ms deviation) in n% of peaks.
- The simulation ran bootstrapped for 10,000 iterations, with values n=[5, 10, 20].

Results show that the effect of incorrect beat placements **far outweigh** those of missing values. As described earlier, the instantaneous heart rate (BPM) is not sensitive to outliers, as is shown in the plots as well, where almost no discernible deviation is visible.

n=5%


Effects of disturbances on RMSSD
Effects of disturbances on SDSD
Effects of disturbances on BPM
orignal
missing 5% of beats
displacing 5% by max 10%

n=10%

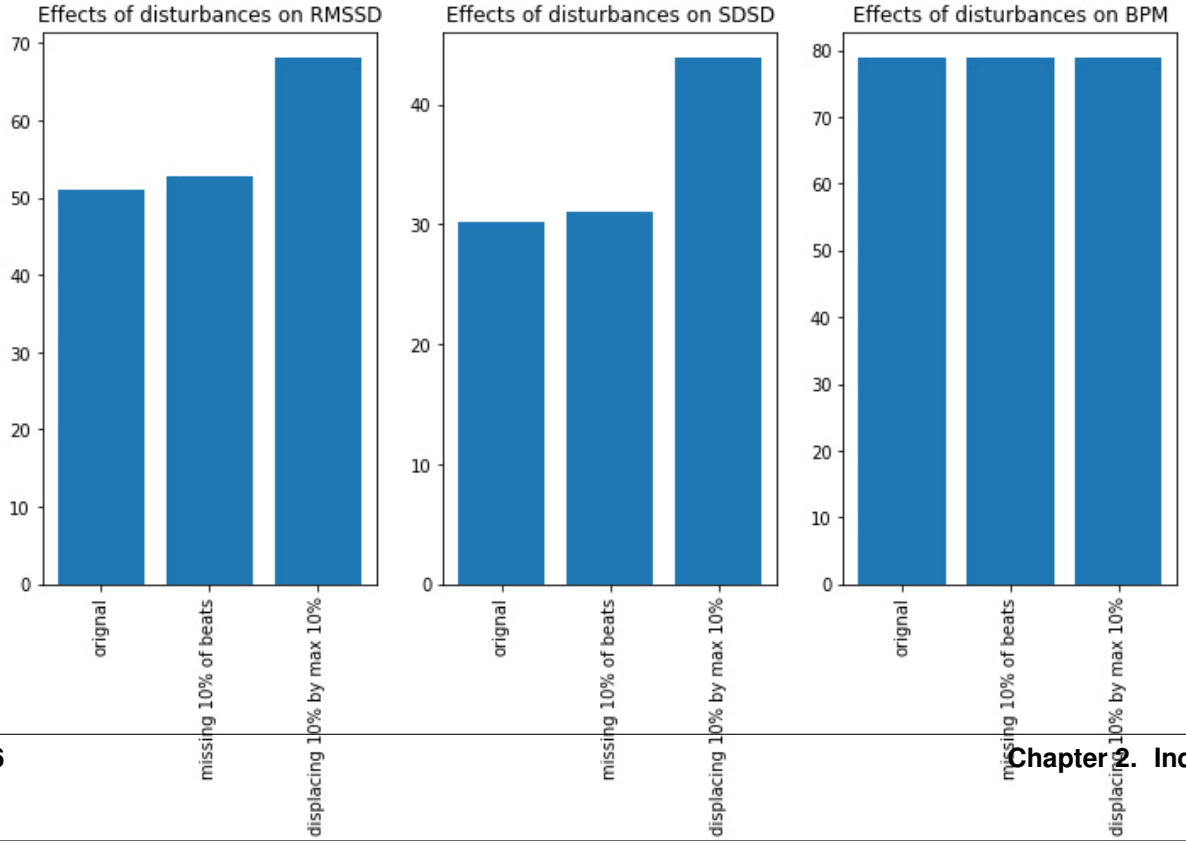

Effects of disturbances on RMSSD
Effects of disturbances on SDSD
Effects of disturbances on BPM
orignal
missing 10% of beats
displacing 10% by max 10%

Figure 4 - Results for manually anotated measures (ground truth), and error induction of n% missed beats, as well as error induction on the detected position of n% beats (random error 0.1% - 10%, or 1-100ms).

Take into consideration that the scale for RMSSD doesn't typically exceed +/- 130, SDSD doesn't differ by much. This means that even a few incorrectly detected peaks are already introducing large measurement errors into the output variables. The algorithm described here is specifically designed to handle noisy PPG data from cheap sensors. The main design criteria was to minimise the number of incorrectly placed peaks as to minimise the error introduced into the output measures.

More information on the functioning can be found in the rest of the documentation, as well as the **embedded paper**. Information on the valiation can be found in[5].

2.2.4 References

2.3 Algorithm functioning

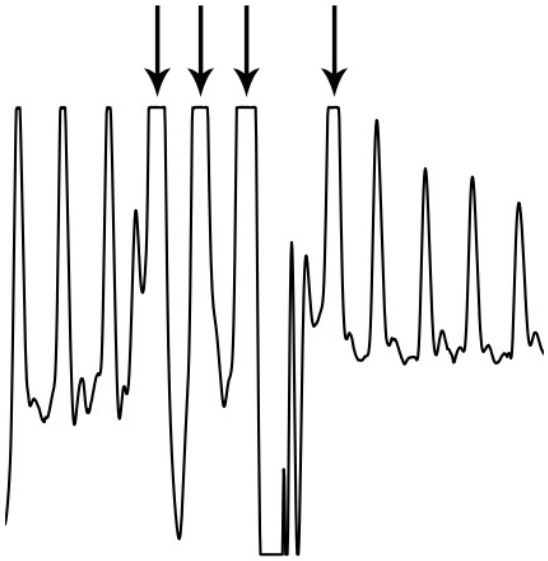
This section describes the details of the algorithm functionality.

2.3.1 Pre-processing

Various options are available for pre-processing. These are described below

Clipping detection and interpolation

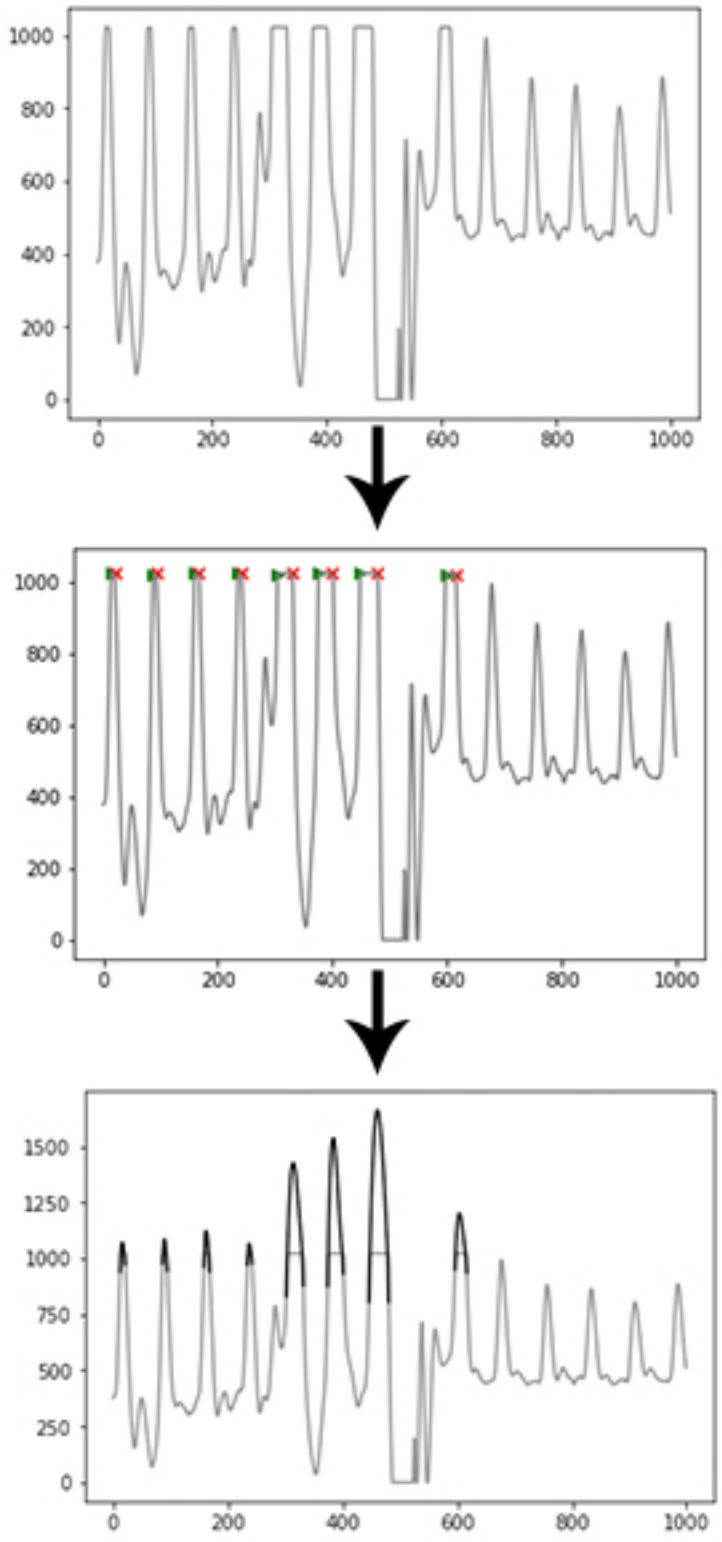
Whenever a measured property exceeds a sensor's sensitivity range, or when digitising an analog signal, clipping can occur. Clipping in this case means the peaks are flattened off because the signal continues outside the boundaries of the sensor you're using:



Clipping functions by detecting (almost) flat parts of the signal near its maximum, preceded and followed by a steep angle on both ends. The 'missing' signal peak is interpolated using a cubic spline, which takes into account 100ms of

[5] van Gent, P., Farah, H., van Nes, N., & van Arem, B. (2018). "Heart Rate Analysis for Human Factors: Development and Validation of an Open Source Toolkit for Noisy Naturalistic Heart Rate Data." In proceedings of the Humanist 2018 conference, 2018, pp.173-17

data on both ends of the clipping portion of the signal. The reconstructed R-peak is overlaid on the original signal and used for further analysis.

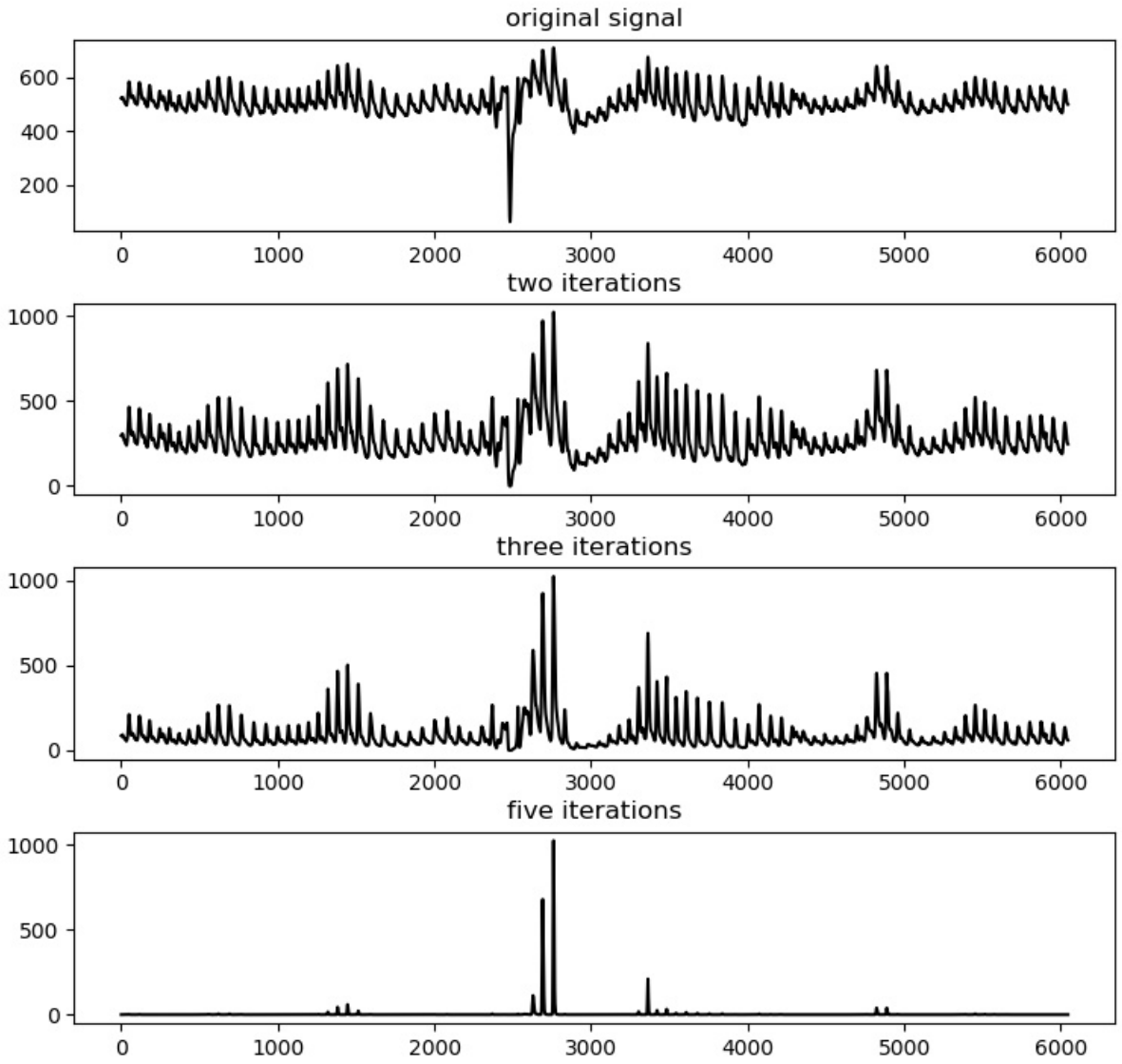


Peak enhancement

A peak enhancement function is available that attempts to normalise the amplitude, then increase R-peak amplitude relative to the rest of the signal. It only uses linear transformations, meaning absolute peak positions are not disturbed (in contrast to FIR filters). It runs a predefined number of iterations. Generally two iterations are sufficient. Be cautious not to over-iterate as this will start to suppress peaks of interest as well.

```
import heartpy as hp

enhanced = hp.enhance_peaks(data, iterations=2)
```

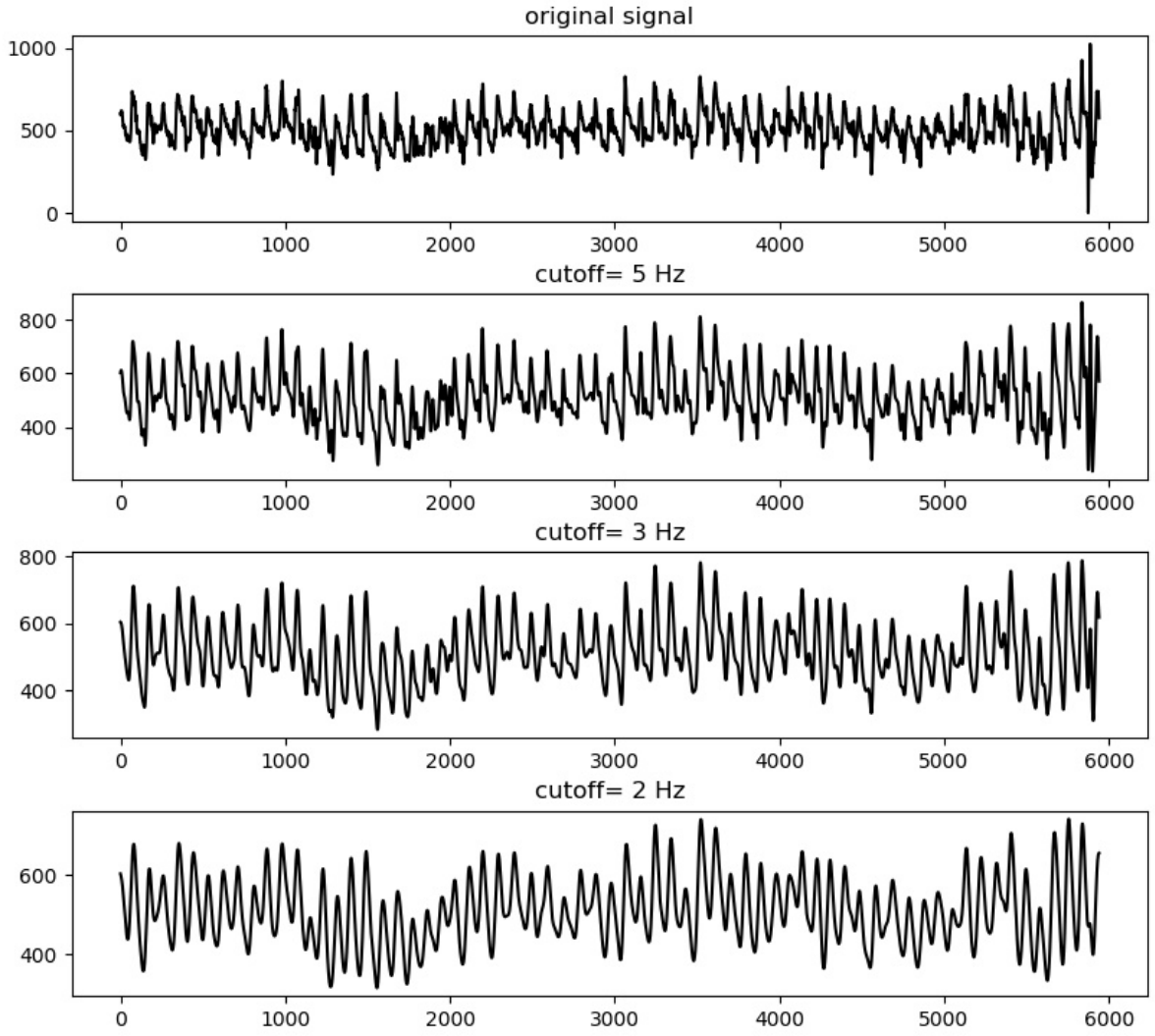


Butterworth filter

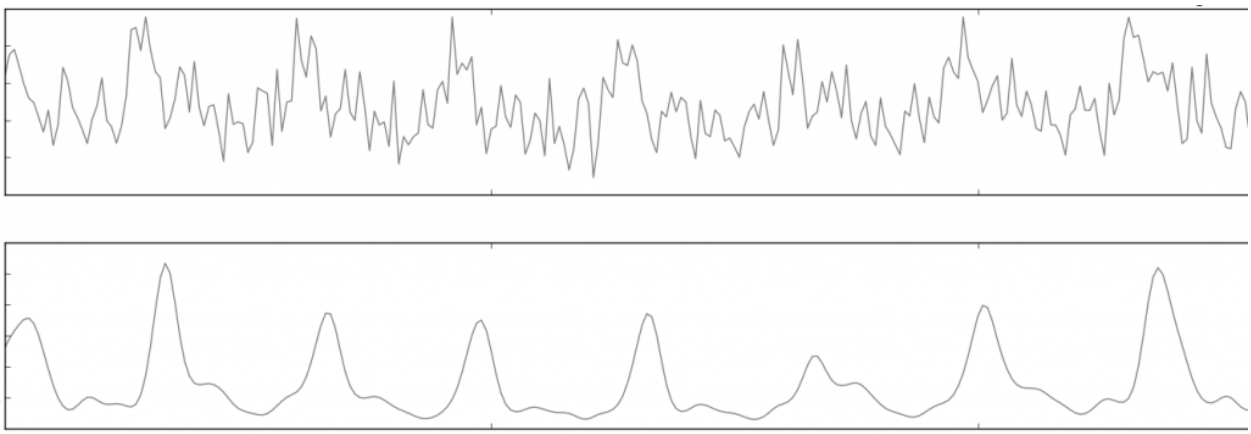
A Butterworth filter implementation is available to remove high frequency noise. Note that this will disturb the absolute peak positions slightly, influencing the output measures. However, in cases of heavy noise this is the only way useful information can be extracted from the signal.

```
import heartpy as hp

filtered = hp.butter_lowpass_filter(data, cutoff=5, sample_rate=100.0, order=3)
```



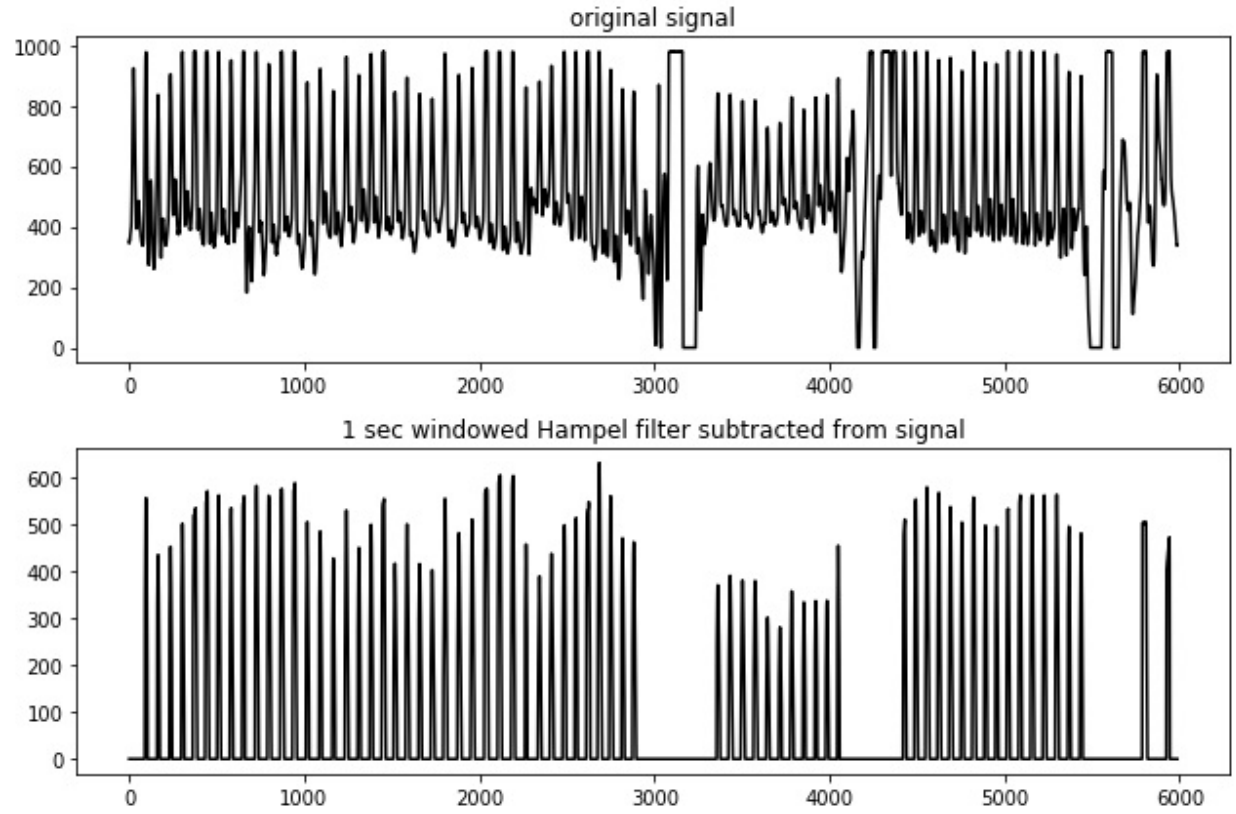
Filtering is generally not recommended unless there is high noise present in the signal. An extreme example is displayed below:



Hampel Correction

The Hampel Correction functions as an extended version of a Hampel Filter, with a larger window size than the standard datapoint + 3 datapoints on each side (=7). The downside is that it (the current implementation at least) takes significant processing time since a window median and median absolute deviation needs to be computed for each datapoint.

In the current implementation, if called, a median filter is taken over a 1-sec window of the heart rate signal. The filter output is subsequently subtracted from the original signal. When doing so, the property of noise suppression arises:



Note that the absolute peak positions will shift slightly when using this type of filter. With it, the output measures will start to deviate as error is induced. Generally the error is not high, but by default hampel filtering is disabled. It should

only be used when encountering segments of heavy noise that the algorithm cannot handle properly.

2.3.2 Peak detection

The peak detection phase attempts to accommodate amplitude variation and morphology changes of the PPG complexes by using an adaptive peak detection threshold (Fig 3, III), followed by several steps of outlier detection and rejection. To identify heartbeats, a moving average is calculated using a window of 0.75 seconds on both sides of each data point. The first and last 0.75 seconds of the signal are populated with the signal's mean, no moving average is generated for these sections. Regions of interest (ROI) are marked between two points of intersection where the signal amplitude is larger than the moving average (Fig 3, I-II), which is a standard way of detecting peaks. R-peaks are marked at the maximum of each ROI.

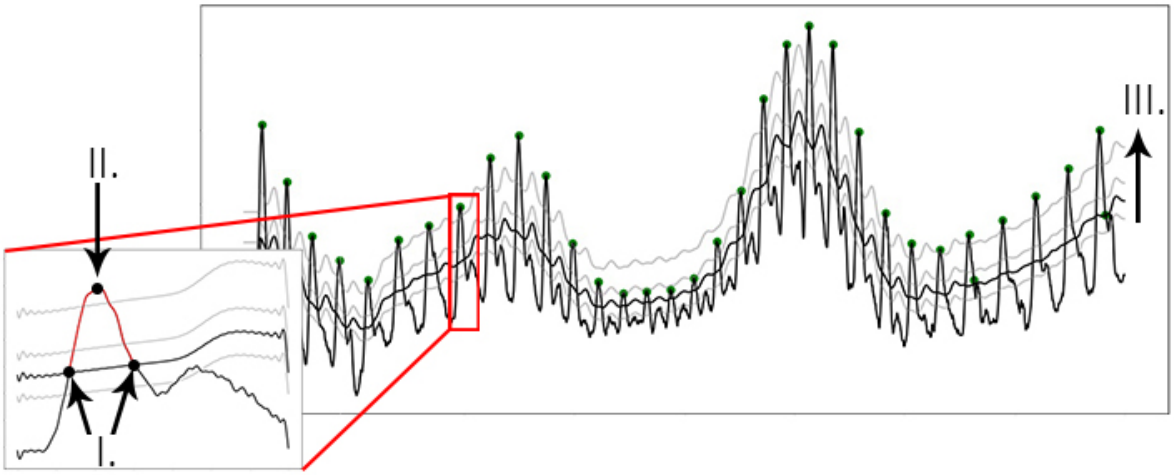


Figure showing the process of peak extraction. A moving average is used as an intersection threshold (II). Candidate peaks are marked at the maximum between intersections (III). The moving average is adjusted stepwise to compensate for varying PPG waveform morphology (I).

During the peak detection phase, the algorithm adjusts the amplitude of the calculated threshold stepwise. To find the best fit, the standard deviation between successive differences (SDSD, see also 2.2) is minimised and the signal's BPM is checked. This represents a fast method of approximating the optimal threshold by exploiting the relative regularity of the heart rate signal. As shown in the figure below, missing one R-peak (III.) already leads to a substantial increase in SDSD compared to the optimal fit (II.). Marking incorrect R-peaks also leads to an increase in SDSD (I.). The lowest SDSD value that is not zero, in combination with a likely BPM value, is selected as the best fit. The BPM must lie within a predetermined range (default: 40 <= BPM <= 180, range settable by user).

The figure below displays how the SDSD relates to peak fitting. In essence the fitting function exploits the strong regularity expected in the heart rate signal.

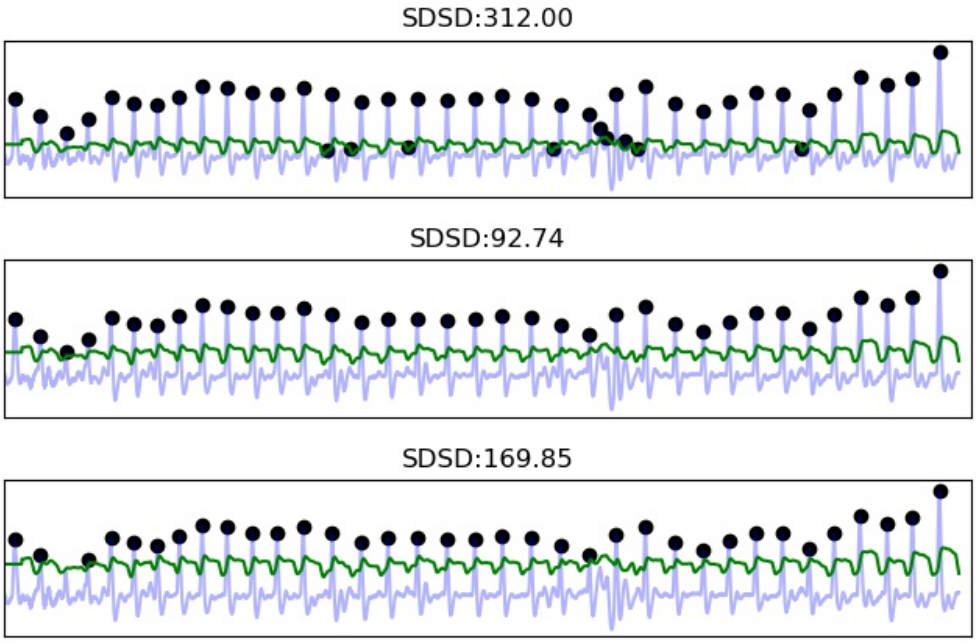
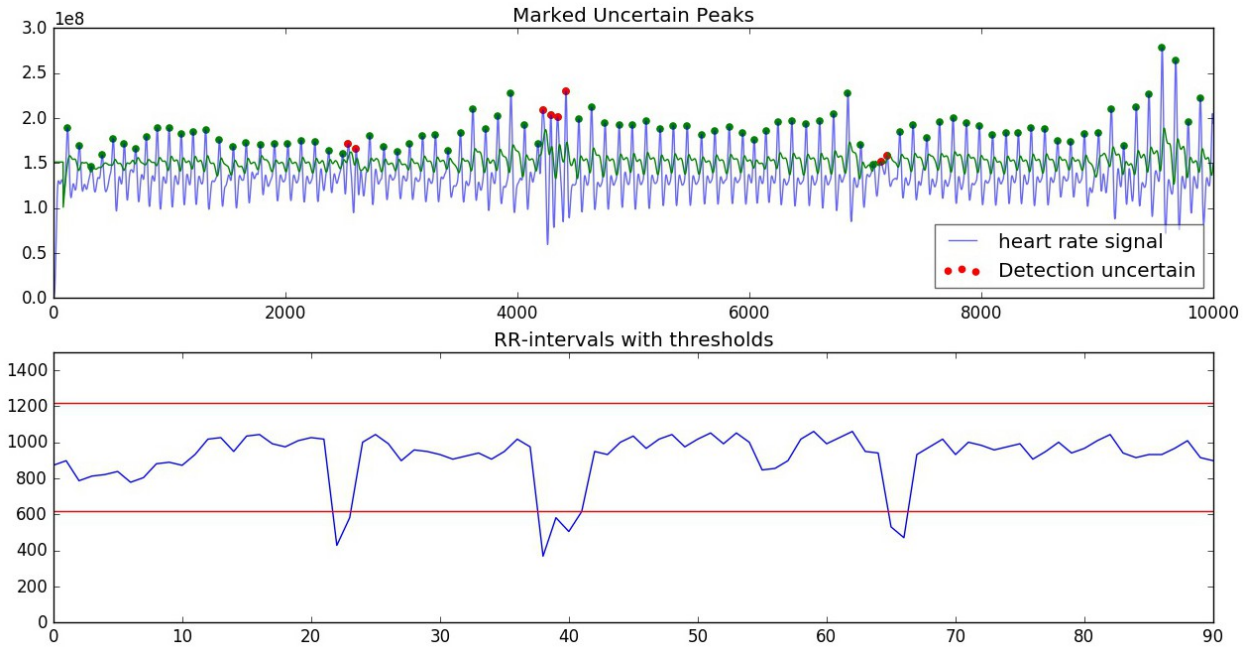


Figure showing how the SDSD responds strongly even to a single missed beat (bottom plot), and is lowest when all peaks are properly detected (middle plot).

Whenever clipping occurs, the algorithm detects this and will attempt to reconstruct the waveform by spline interpolation. This is discussed under *Clipping detection and interpolation*

2.3.3 Peak rejection

After the fitting phase, several incorrectly detected peaks may still remain due to various factors. These are tested and rejected based on a thresholded value for the RR-intervals in the section:



Thresholds are computed based on the mean of the RR-intervals in the segments. Thresholds are determined as **RR_mean +/- (30% of RR_mean, with minimum value of 300)** (+ or - for upper and lower threshold, respectively). If the RR-interval exceeds one of the thresholds, it is ignored.

2.3.4 Calculation of measures

All measures are computed on the detected and accepted peaks in the segment. When RR-intervals are used in computation, only the intervals created by two adjacent, accepted, peaks are used. Whenever differences in RR-intervals are required (for example in the RMSSD), only intervals between two adjacens RR-intervals, which in turn are created by three adjacent, accepted, peaks are used. This ensures that any rejected peaks do not inject measurement error in the subsequent measure calculations.

Time-series

Time series measurements are computed from detected peaks. The output measures are:

- beats per minute (BPM)
- interbeat interval (IBI)
- standard deviation of RR intervals (SDNN)
- standard deviation of successive differences (SDSD)
- root mean square of successive differences (RMSSD)
- proportion of successive differences above 20ms (pNN20)
- proportion of successive differences above 50ms (pNN50)
- median absolute deviation of RR intervals (MAD)

Frequency Domain

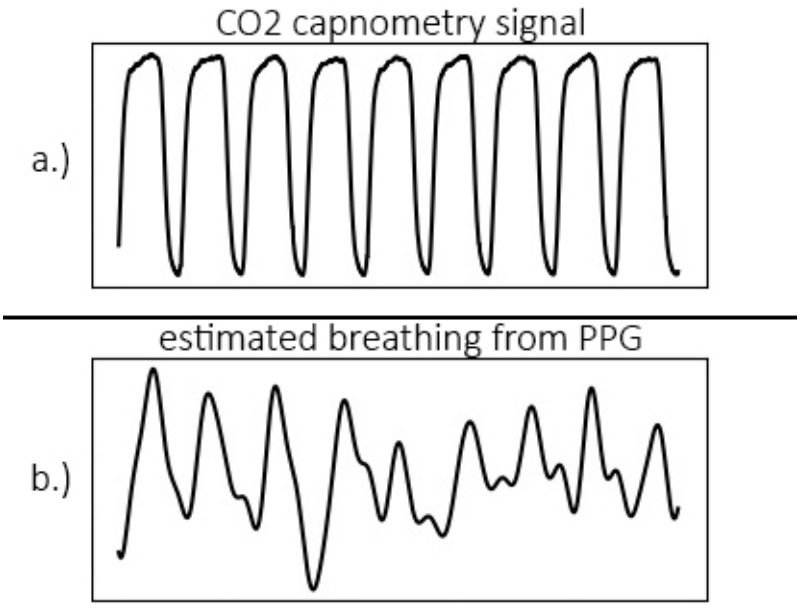
Frequency domain measures computed are:

- low-frequency, frequency spectrum between 0.05-0.15Hz (LF)
- high-frequency, frequency spectrum between 0.15-0.5Hz (HF)
- the ration high frequency / low frequency (HF/LF)

The measures are computed from the PSD (Power Spectral Density), which itself is estimated using either FFT-based, Periodogram-based, or Welch-based methods. The default is Welch's method.

Estimating breathing rate

One interesting property of the heart is that the frequency with which it beats is strongly influenced by breathing, through the autonomous nervous system. It is one of the reasons why deep breaths can calm nerves. We can also exploit this relationship to extract breathing rate from a segment of heart rate data. For example, using a dataset from[1] which contains both CO2 capnometry signals as well as PPG signals, we can see the relationship between breathing and the RR-intervals clearly. Below are plotted the CO2 capnometry signal (breathing signal measured at the nose), as well as the (upsampled) signal created by the RR-intervals:



The problem is now reduced to one of peak finding. Breathing rate can be extracted using the toolkit. After calling the 'process' function, breathing rate (in Hz) is available in the dict{} object that is returned.

```
import heartpy as hp

data = hp.get_data('data.csv')
```

(continues on next page)

[1] 23. Karlen, S. Raman, J. M. Ansermino, and G. A. Dumont, "Multiparameter respiratory rate estimation from the photoplethysmogram," IEEE transactions on bio-medical engineering, vol. 60, no. 7, pp. 1946–53, 2013. DOI: 10.1109/TBME.2013.2246160 PMED: http://www.ncbi.nlm.nih.gov/pubmed/23399950

(continued from previous page)

```
fs = 100.0
working_data, measures = hp.process(data, fs, report_time=True)
print('breathing rate is: %s Hz' %measures['breathingrate'])
```

This will result in:

```
breathing rate is: 0.16109544905356424 Hz
```

2.3.5 References

2.4 Development

2.4.1 Release Notes

V0.8.1

- Added changelog to repository
- Implemented clipping detection and interpolation functionality
- Changed FFT calculation flag to default False, as in some cases the FFT takes very long to compute. Possible causes and fixes to be investigated
- Pushed readthedocs.io documentation source structure to repository
- Added encoding argument to get_data function, per the NumPy deprecation of not using encoding. For more info: https://docs.scipy.org/doc/numpy-1.14.0/release.html#encoding-argument-for-text-io-functions

V0.8.2

- RR_difference interval no longer taken into account when RR-intervals are not technically adjacent due to rejected peak presence in between
- Moved matplotlib import statement so that it is no longer necessary unless calling the plot functionality, reduces need to install irrelevant dependencies when plotting functionality not needed
- Added Hampel Filter with settable filtersize
- Added method to suppress noisy segments called 'Hampel Corrector', called such as it's simply a Hampel Filter with large window size. Computationally on the expensive side so disabled by default, but very good at suppressing noisy segments without influencing peak positions in the rest of the signal.
- Added breathing rate extraction method. Stores estimated breathing rate in measures['breathingrate']
- Made BPM threshold values settable
- Added Periodogram- and Welch-based PSD estimation
- Added support for edge case where clipping segment starts early in signal, meaning there is insufficient data to interpolate accurately.

V1.0

- Released Version 1.0
- Added flag to disable scaling when interpolating clipping segments. Useful for data with large amplitude variations.
- Added marking of rejected segments in plotter
- Added automatic peak rejection when first peak occurs within 150ms, since the signal might start just after a peak, which creates slight inaccuracy.
- Added segment rejection based on percentage of incorrect peaks.

V1.0.1

- Changed segmentwise rejection API to simplify plotting

V1.1

- We are now officially called HeartPy
- Changed overall structure to get rid of global dicts, allows for modular or multithreaded use easier.
- Changed docs to reflect changes

2.4.2 Questions

contact me at P.vanGent@tudelft.nl