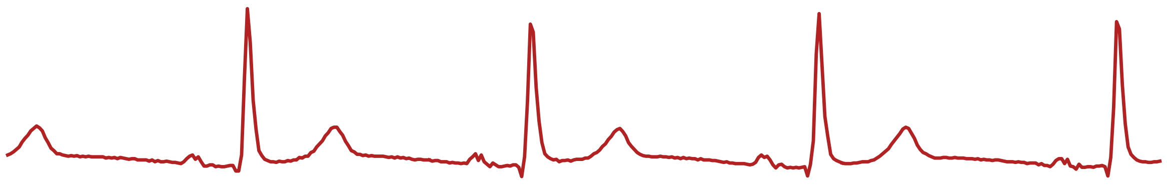
THE UNIVERSITY OF
MELBOURNE

Final report

The Art of Scientific Computing: Respitory Rates

Author:
Hespera Henzell[redacted]

Subject Handbook code:
COMP-90072



Faculty of Science
The University of Melbourne

19 May 2020

Subject co-ordinator: A/Prof. Roger Rassool

Contents

Chapter 1

Introduction

1.1 Physiology of the Heartbeat

The heart and lungs work together to supply oxygen to the body. They are both moving things, expanding and contracting next to each other in the tight chest cavity. It makes sense that the movement of one should affect the other.

Figure 1.1 shows a segment of a typical ECG. The most important feature of an ECG is the QRS complex, which occurs when the heart beats. It is characterised by a sharp dip (the Q), spike (the R-peak), and another sharp dip (the S) [6]. Typically the R peak is tall compared to the rest of the ECG. The RR-interval is the period beween two R-peaks, or two heartbeats [6].

The P-wave occurs before the heartbeat and the T-wave just after [6]. The T-waves can sometimes be quite tall, but they are not as sharp as the R peak [4] .

The RR-interval is one indication of the breath rate. As we breathe in the heartbeat quickens and the

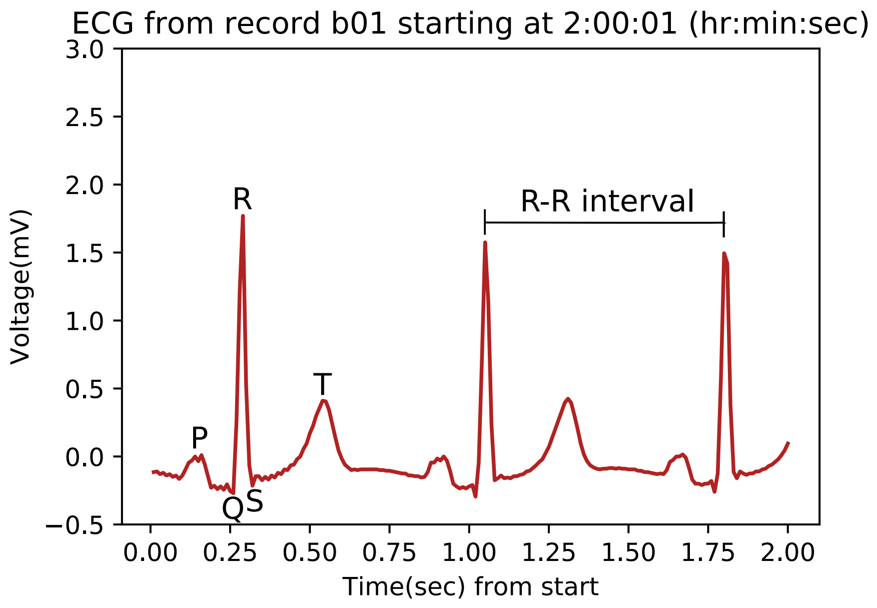


Figure 1.1: Features of the typical ECG

interval shortens, and as we breathe out our heart beat slows and the interval lengthens. This pattern is called Respirtory Sinus Arrythmia (RSA). [8]

We can treat breathing as a wave, where a full breath in and out is one period. Due to RSA, the length of RR-interval is also a wave, with the same period as the breath rate (albeit with a phase shift) [7].

1.2 Fast Fourier Transfrom

Any curve in the universe can be approximated by a series of polynomials, which are found by doing a Fourier Transform. Signals, such as the ECG recordings, are actually a sum of different waves at different frequencies. By doing a Fast Fourier Transform (FFT) on the signal, we can break it down into its consitient frequencies.

A normal resting breath rate for an adult is 12 to 20 breaths per minute [1]. If we consider taking a full breath in and out as one period, then that corresponds to a frequency of 0.2 to 0.33 Hz. Since we know the ECG contains this information, we would expect to see a frequency in the FFT in this range, corresponding to the breath rate. This project aimed to find the breath rate in the FFT of the ECG.

1.3 Overview of Code

All the code was written in Python. To start with, I found the RR-interval to estimate the breath rate. Programs were written to find the R-peak and calculate the RR-interval. Finally a program, `pbf` was written to roughly estimate the breath rate from the changes in the RR-interval, as well as graph the changing RR interval.

Programs were written to perform a FFT on segments of the ECGs, and graph the frequencies of the FFT where we expect to see the heart and breath rates. A spectrogram was also created to see changes in the signal in time.

Using the spectrogram to find segments of the ECG without noise, I could successfully find a signal close to where `pbf` predicted the breathing rate to be for some records. Thus we appear to be able to find the breath rate in the FFT.

1.4 The Data

The ECGs were taken from the Apnea-ECG Database on Physionet, containing ECGs taken overnight from people with and without sleep apnea [2] [5]. Figure 1.2 shows an ECG taken during an apnotic episode.

There were three sets of recordings, prefixed with “a”, “b” or “c”, which I will refer to as the a, b and c

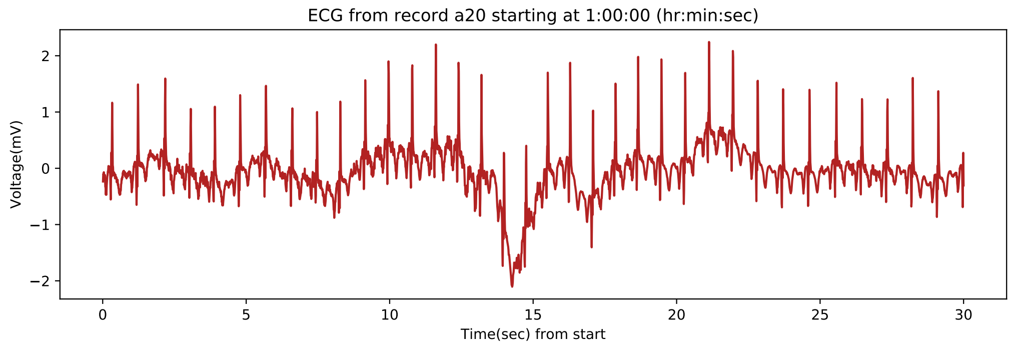


Figure 1.2: ECG signal taken during sleep apnea. Made with `ecg_g`.

groups from here on. From the accompaning information, we know that recordings from the a group came from people suffering from quite bad apnea, having from 100 to 534 minutes of apnea in their recordings. The b group were not as badly afflicted. They ranged from having 10 to 93 minutes of apnea in their recordings. The c group were not troubled with apnea, only having a few minutes of apnotic sleep, if any. Out of the three groups, the c group contained the youngest people.

Since RSA is diminished by age and cardiovascular disease (a comorbidity of sleep apnea)[8], group c should show the strongest RSA. Later in this project, when I was using the RR-interval to predict the breath rate, I used group c.

Chapter 2

Code

2.1 Directory Structure

The working directory for this project contained the code and a folder named "data" where all the ecg recordings were kept.

2.2 Parameters and Variables

All the programs written use a common set of parameters and variables

Parameters and Variables	
`record`	name of the record (as a string)
`offset`	time of recording (seconds) to start reading the data
`seconds`	number of seconds of the ECG we wish to record
`ecg`	a segment of an ECG read into a useable form by `ecg_r`
`freq`	frequency of record (Hz)
`time_step`	`freq`$^{-1}$, or number of seconds per reading
`byte`	number of bytes reading in ECG is stored in
`AD`	A\D units per mV in record

2.3 Reading and Graphing the ECGs

The program `ecg_r` was written to read the data, and `ecg_g` was written to graph it.

The ECGS are stored in 16-bit signed two's complement binary files. When `ecg_r` first reads this into the computer, it stores each number in two bytes. There are 100 samples per second. This means to get to the right start point, `ecg_r` must skip 200 times the number of bytes ahead as the number of seconds we wish to start at. There are 200 A/D units per millivolt in the records, so to make sure we have our data in millivolts, `ecg_r` divides the each integer it reads by 200.

`ecg_g` uses `ecg_r` to obtain a small sample of a recording, and plots it with `pyplot` from `matplotlib`. Figure 1.2 was made with `ecg_g`.

2.4 Finding Peaks

The program written to find individual peaks is `peak_find`. Once triggered, it finds a local maximum, and classifies it as a R-peak if it is sharp enough. There is a program flowchart of `peak_find` in figure 2.1.

First, in a while loop `peak_find` updates the maximum height of the R-peak and the time this occured at, as it comes across higher values. Every number lower than the current maximum increases a counter by 1, and when that counter reaches five, the while loop stops and `peak_find` stops updating the maximum.

Initally this was all `peak_find` contained, however it sometimes misclassified T-waves as R-peaks. T-waves can be as high as R-peaks, but unlike R-peaks, which are sharp and jaggard, they are smooth and comparitively wide. So `peak_find` was modified to find the width of the peak. If the width of the peak was less than 8 readings, `peak_find` decided the peak was caused by an R-peak, and returned the time of the peak. Otherwise it returned the string `"Not Peak"` . The width of 8 was found by trial and error.

If a segment of an ECG ended just before the peak of an R-peak, `peak_find` returned Index Errors. The

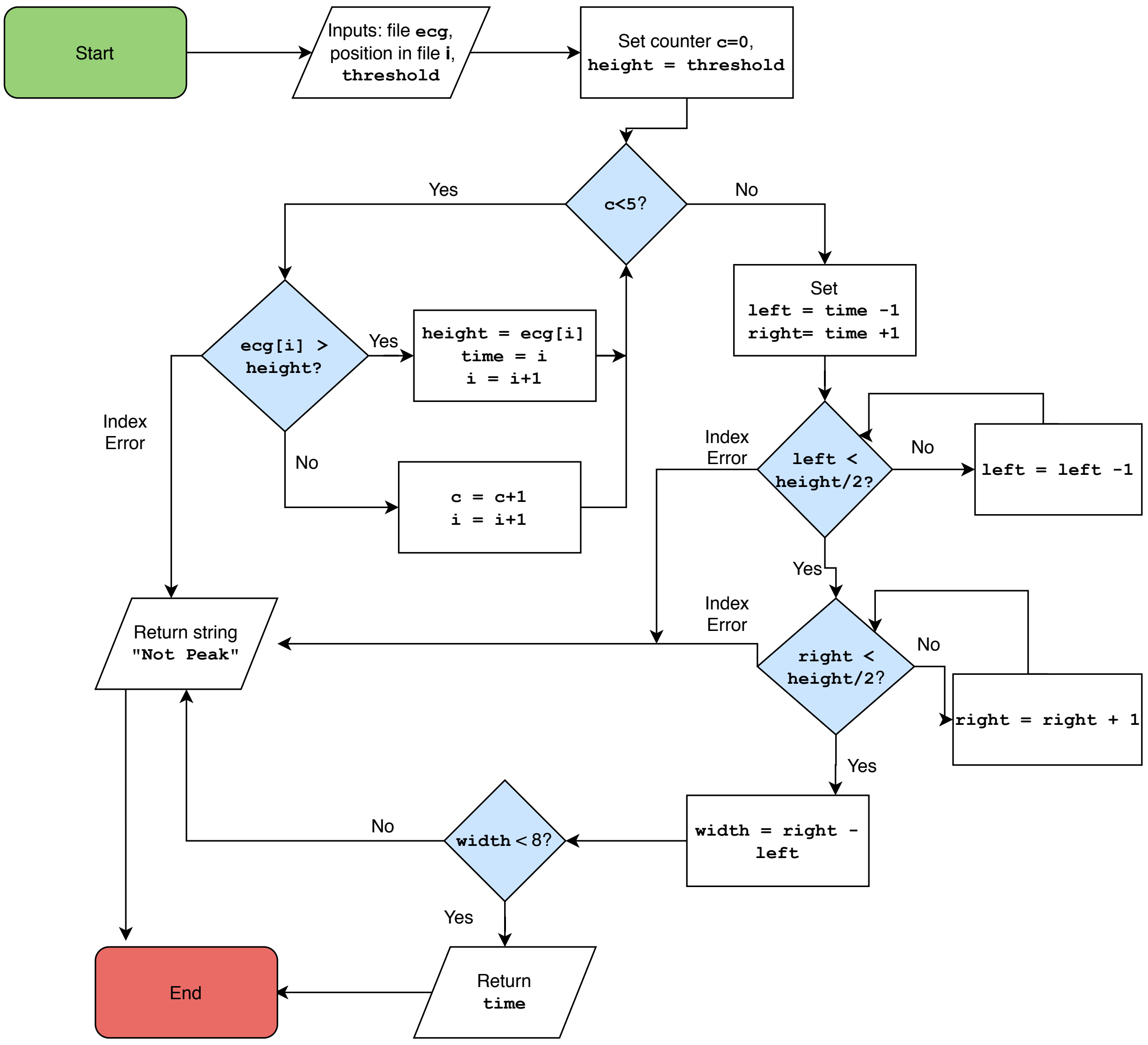


Figure 2.1: Program flowchart of `peak_find`

troublesome piece of code was:

programs/problem.code.py

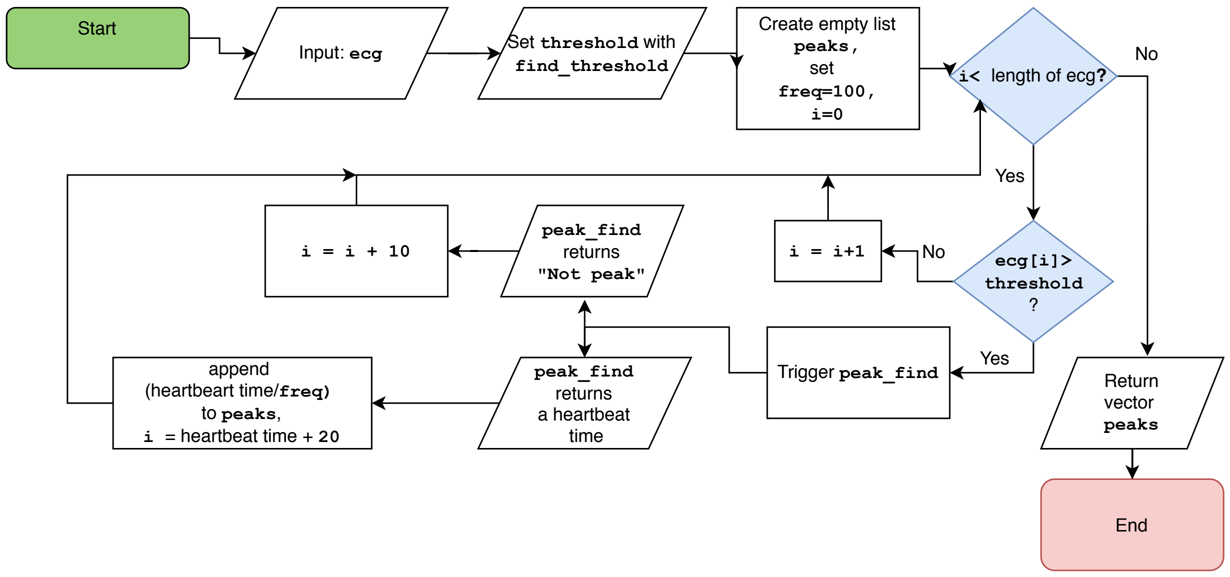
```
while c<5:
    if ecg[i]>height:
        height =ecg[i]
        time   =i
    else:
        c+=1
        i+=1
```

The segment of the recording would run out before the counter `c` had the chance to reach 5 and take the program out of the loop, so `peak_find` starting trying to look for entries to `ecg` that were not there. Another part of the program could also cause the same problem. The quickest solution was placing troublesome sections in try loops, and returning `"Not Peak"` if an Index Error occured.

The program `heartbeat_vector` uses `peak_find` to identify all the peaks in a segment of an ECG. Figure

2.2 contains a program flowchart of `heartbeat_vector`. It should identify the R-peaks in a given segment and return a vector of times they occur. It begins to look for R-peaks after the signal reaches a certain threshold. This is initially set by the program `find_threshold`, at halfway between the maxium and mean values in the first 10 seconds of the segment of interest. The threshold starts being updated at 5 seconds, to 65% of the height of the previous peak. Reaching the threshold triggers `peak_find`.

If the threshold has been triggered, the next point is also likely to trigger the threshold. To avoid looking for the same peak again, `heartbeat_vector` jumps ahead 0.2 seconds if `peak_find` returns a R peak. The human heart has a refractory period of 0.2 seconds after a QRS complex when it cannot beat again [4]. If `peak_find` returns `"Not Peak"`, `heartbeat_vector` jumps ahead 0.1 seconds. It needs to move far enough that the ECG has moved back below the threshold, but not so far as to miss a heartbeart. With testing, 0.1 seconds was found to work well.



Start
Input: ecg
Set threshold with find_threshold
Create empty list peaks, set freq=100, i=0
i< length of ecg?
No
Yes
ecg[i]> threshold ?
No
i = i+1
Yes
Trigger peak_find
peak_find returns "Not peak"
i = i + 10
peak_find returns a heartbeat time
append (heartbeart time/freq) to peaks, i = heartbeat time + 20
Return vector peaks
End

Figure 2.2: Program flowchart of `heartbeat_vector`

`test_ecg` uses `heartbeat_vector` to find peaks, and graphs them on top of the ECG.

Some examples of `heartbeat_vector`'s performance, graphed with `test_ecg`, are shown in figure 2.3. After looking at many segments of ECGs with `test_ecg`, I could see that `heartbeat_vector` generally manages to find heartbeats, and ignore high T-waves. It performed better with a shifting baseline than expected, even with the very simple way the threshold was updated. It did struggle with noise (the signals supplied to it were completely unfiltered). It also sometimes skipped low R-peaks. More sophisticated methods like the Pan Tompkins algorithm search for missed beats when the interval between beats is long [4] - this program is much more crude than that.

However, `heartbeat_vector` was good enough at identifying R-peaks for the purposes of this project.

2.5 Predicting Breath Rate from Heart Rate

Once we could predict the heart rate, it was easy to find and graph the RR-interval. The program `RR_int` reads a segment of an ECG into Python, and uses `heartbeat_vector` to identify the heartbearts. It then finds the interval between heartbeats, and the time halfway through the inteval. It returns a vector of heartbeat intervals, and a vector of the time they occur (taken mid interval). The beats can then be plotted using the graphing program `RR_int_graph` or `pbf`.

Initially, `RR_int` just returned the intervals between heartbeats. However, as discussed below, the graphs of changes in the heartbeats were useful for seeing changes in the ECG, so the vector of times halfway through the interval was added. Then the times the ECG changes could be located from a graph of the RR-intervals.

`pbf` predicts the heart rate by counting the peaks in the RR-interval supplied by `RR_int`, as well as graph-

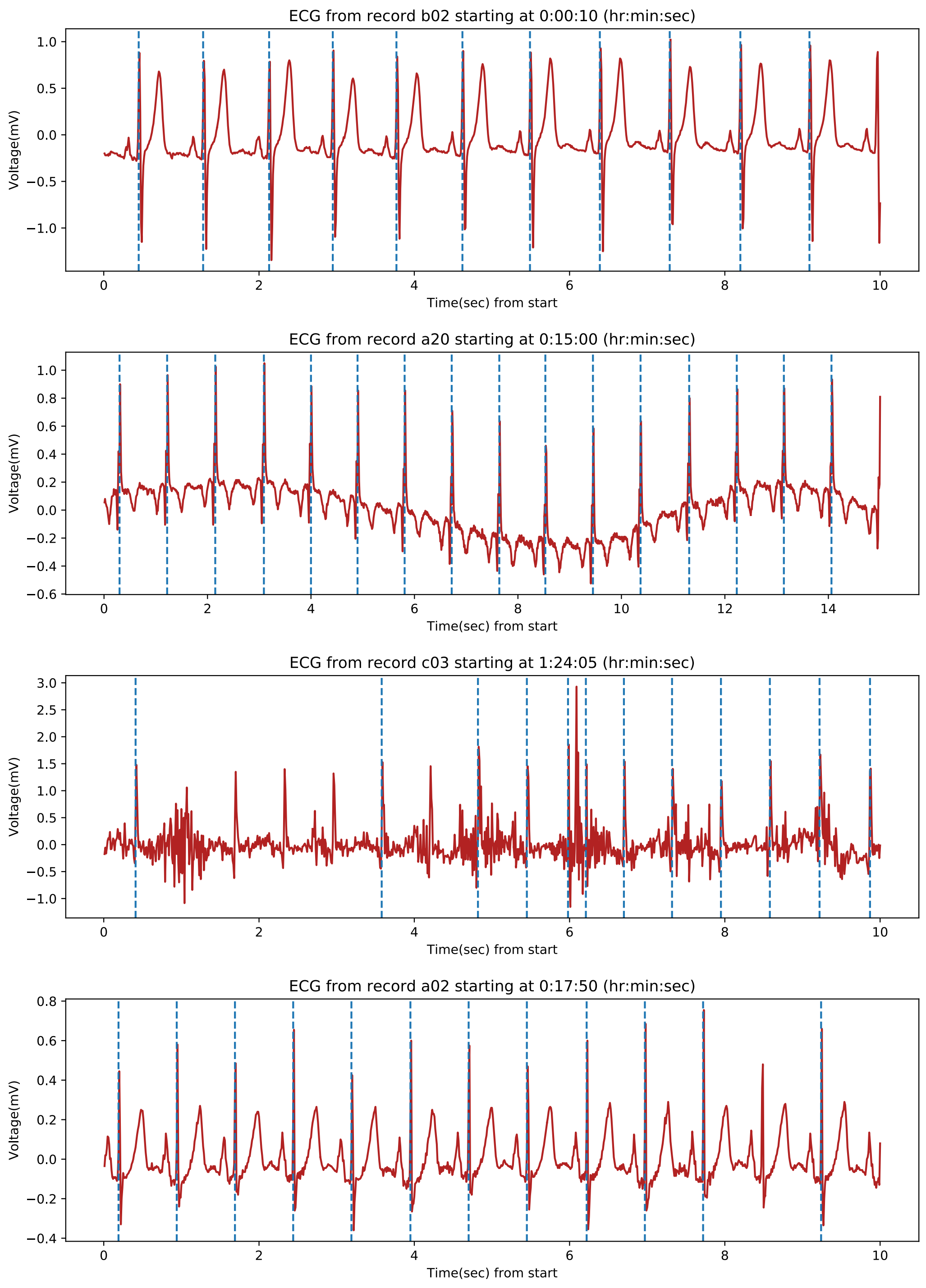


Figure 2.3: Segement sof ECGs with blue lines where `heartbeat_vector` has identified heartbeats. In the top ECG `heartbeat_vector` has correctly identified the R peaks and ignored the high T-waves. Second from the top, all the R-peaks have been correctly identified despite basline shift. In the third ECG from the top, `heartbeat_vector` has not worked well on a noisy segment. In the bottom ECG, `heartbeat_vector` has skipped a low R-peak. Plotted by `test_ecg`.

ing the interval. It estimates the breath frequency in Hertz with the equation:

$$\text{breath frequency (Hz)} = \frac{\text{number of peaks}}{\text{seconds}}$$

`RR_int` was not designed to identify the intervals perfectly. It relies on `heartbeat_vector` to find the heart beats, which as we have seen, is not always robust. If we look at Figure 2.4 we can see 8 very large intervals in a graph created by `pbf`; an indication we might have missed beats. Areas where `pbf` show a change in hearbeat length need to be examined to determine what is going on with tools like `test_ecg`. Sometimes the heartbeat just changes, but other times this corresponds to some feature of the ECG, such as noise, that we know `heartbeat_vector` struggles with. This makes it most likely `heartbeat_vector` is not working well in those segments. However, in segments where `pbf` shows a consistent pattern, it seems highly likely that `heartbeat_vector` is working well.

Ultimately, I did not want to use `pbf` to find the breath frequency, but instead to validate the breath frequency

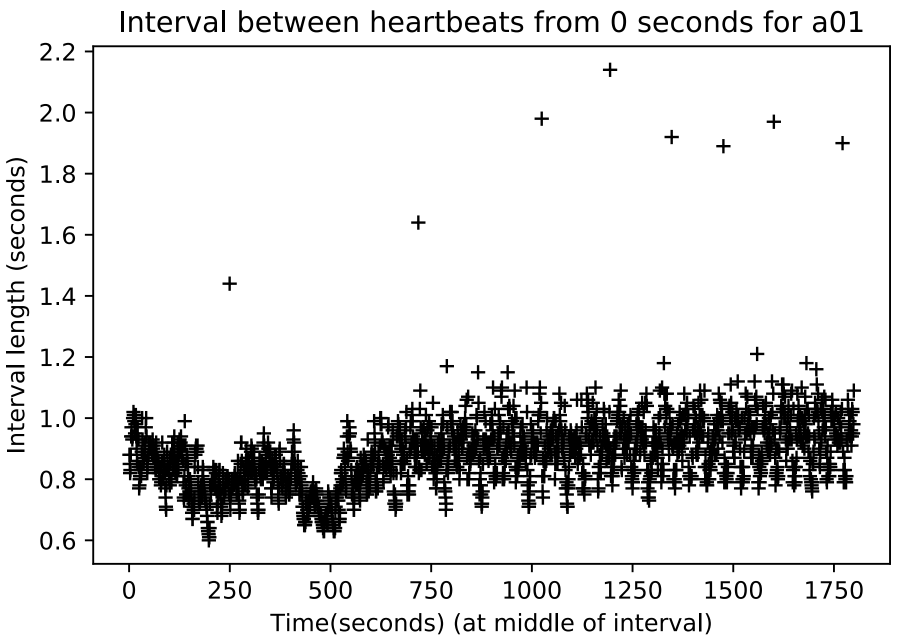


Figure 2.4: Graph of RR-interval created by `pbf`. Predicted heartrate was 0.25 Hz.

found with a FFT, so reasonably good performance was fine.

Figure 2.4 shows a typical graph of the breath rate from `pbf`. There are 8 long intervals. From examining this ECG 100 seconds at a time with `test_ecg`, I surmised that they were all missed beats. This was not too worrying, as 8 missed heartbeats over 30 minutes of recording will not change the estimated breathing frequency by much.

2.6 Performing the FFT and Finding the Breath Rate

Two programs were written to find and graph the FFT: `fft_g` and `fft_spect_g`. The only difference between them is the latter also plots a spectrogram. The two programs use built-in functions from `signal.fft` to perform a FFT. Then they take the absolute value of the FFT to remove complex values. Two plots are made of the FFT: one where the breath rate should be, and one where the heart rate should be. Normal breath frequency is between 0.2 to 0.33 Hz [1], and a normal heart rate is 60 to 100 beats per minute, or 1 to 1.6 Hz [3]. `fft_spect_g` also creates a spectrogram with built in functions that show how the signal changes with time.

Figure 2.5 shows an ideal FFT. We clearly have the heartbeat, and a clear spike for a breath rate. `pbf` predicted a breath rate of 0.22 Hz for this segement of the ECG, which is around the frequency where we see the breath rate, validating our results from the FFT.

The spectrogram shows changes in the signal over time, and so shows periods of noise, or other changes in the ECG that may make the breath rate in the FFT hard to find. Figure 2.6 contains a spectrogram with noise. For every record in the c group, I found intervals where the signal was relatively stable with the help of `pbf` and the spectrogram. I used `pbf` to predict the breathing frequency, and tried to find segments of the ECGs where I could spot this breathing frquency in the FFT.

When the FFT was plotted, the heart reate was almost always clear in the FFT. I could usually find segments of an ECG with a breath rate close to where `pbf` predicted it would be. An exception was record c02.

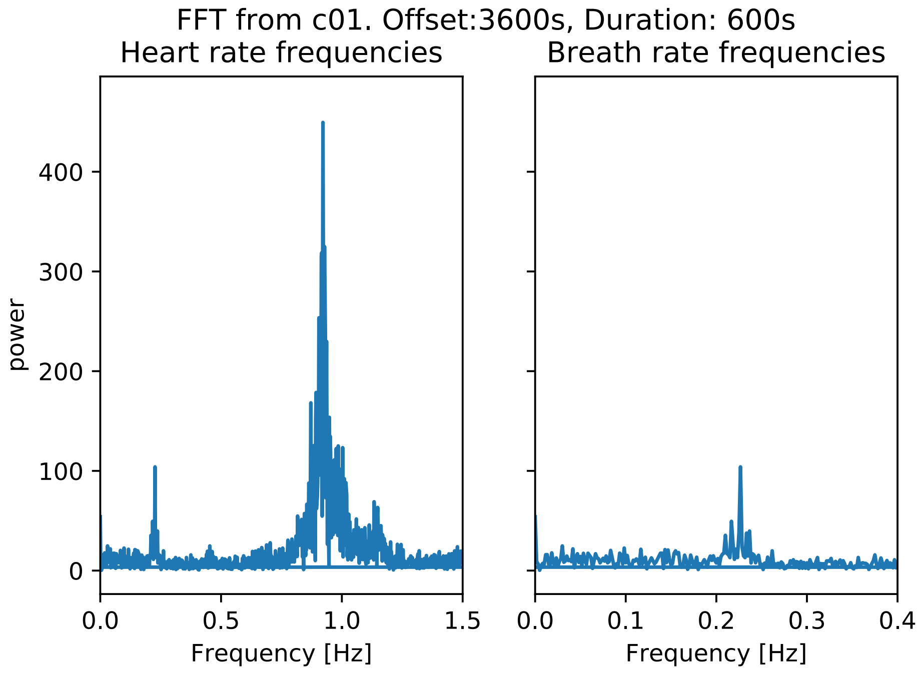


Figure 2.5: An ideal FFT taken from record c01. The breath rate and the heart rate is clear, and the breath rate is around 0.22 Hz, where `pbf` predicted it would be.

After about 2000 seconds, the area where a breath frequency should be is flat, as seen in figure 2.7.

A more common problem was having multiple possible peaks. For example, figure 2.8 shows a FFT with several spikes that could be the breath rate close to 0.2 Hz. `pbf` predicted a breath rate of 0.23 Hz, and there is a spike around there, but there are also other spikes that could be the breath frequency, such as the spike just below 0.2 Hz. Noise could also obsecure the breathing frequency, as shown in figure 2.6.

2.7 Outcome

Ultimately, I did not succeed in creating a program that shows the breathing rate clearly in the FFT. The breath rate can often be found, but the program is not at the point where it is always there or clear which peak it is. However I did create a small suite of programs that can be used to explore segments of the ECG. The programs can visualise whats happening through various methods such as plotting short segments of the actual ECG with `test_ecg`, plotting the (detected) RR-interval with `pbf` or `RR_int_graph`, or viewing the spectrogram with `fft_spect_g`. The breathing frequency can also be estimated with `pbf` and usually a segment of the ECG where we can clearly see the breathing frequency on the FFT can be found with `fft_g`.

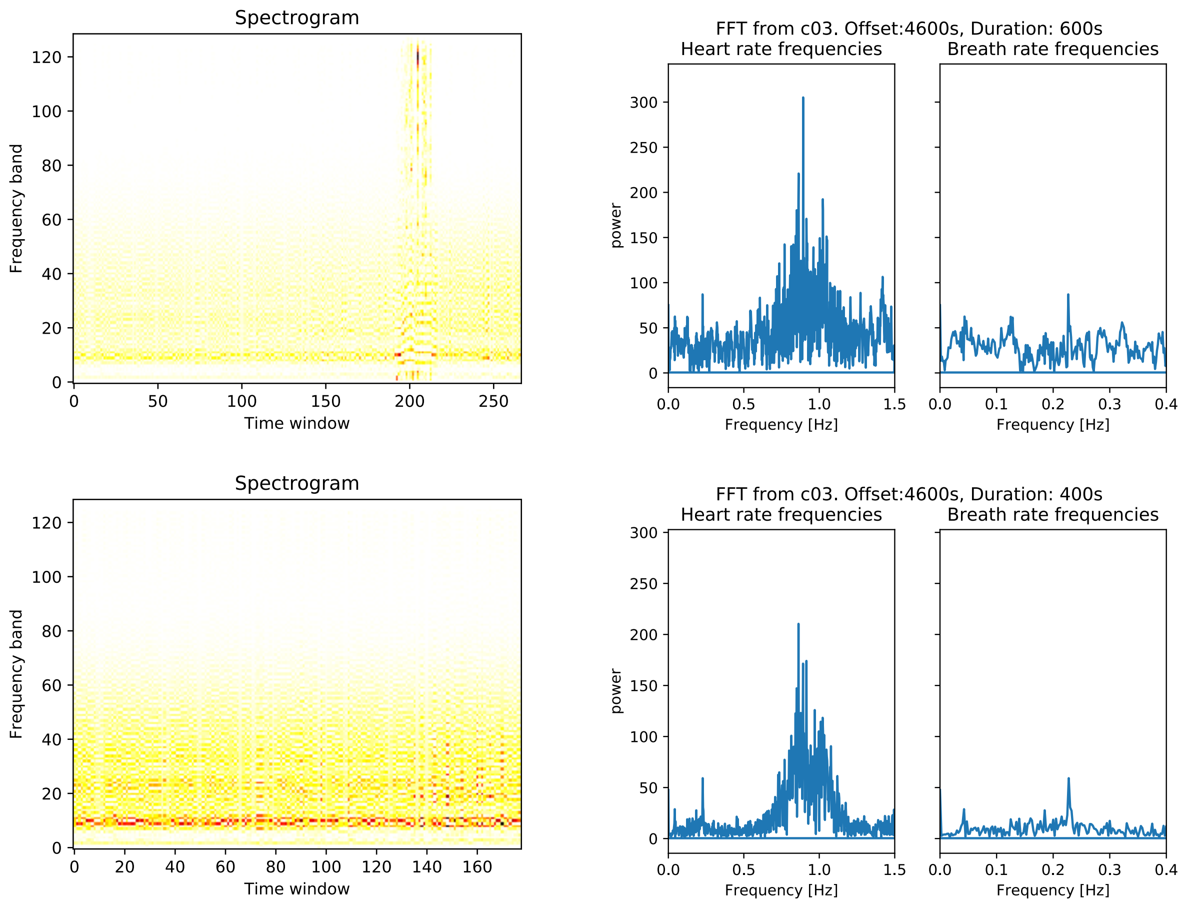


Figure 2.6: The top two images are a Spectrogram and FFT taken from a period of record c03 with noise. The bottom two are taken from the same record at the same time, but are shorter in duration and do not contain the noise. The noise is visible as a clear band in the Spectrogram in the top left, which is absent in the spectrogram below. In the FFTs, we can see the noise obscures the breath rate in the FFT in the top right corner, whereas the breath rate is clear in the FFT at the bottom left corner.

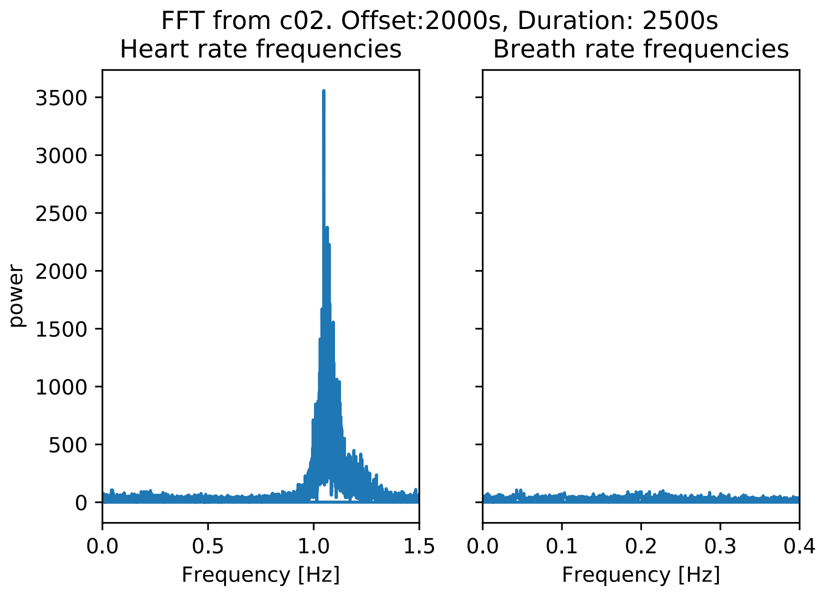


Figure 2.7: FFT taken from record c02 with no clear breath peak

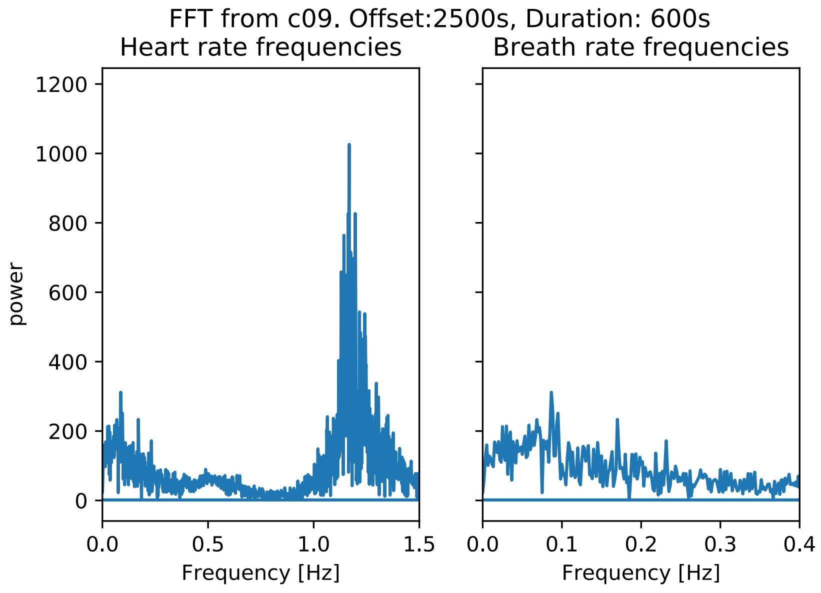


Figure 2.8: FFT taken from record c09 with multiple potential breathing frequencies

Bibliography

[1] C. Chourpiliadis and A. Bahrdwaj. *Physiology, Respiratory Rate StatPearls[Internet]*. StatPearls Publishing, Treasure Island (FL), 2020. Available from: `https://www.ncbi.nlm.nih.gov/books/NBK537306/`. [Updated 2019 Jan 28].

[2] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet. *Circulation*, 101(23):215–220, 2000.

[3] R. Oberman and A. Bhardwaj. *Physiology, Cardiac StatPearls[Internet]*. StatPearls Publishing, Treasure Island (FL), 2020. Available from: `https://www.ncbi.nlm.nih.gov/books/NBK526089/`. [Updated 2018 Oct 27].

[4] J. Pan and W. J. Tompkins. A real-time QRS detection algorithm. *IEEE Trans Biomed Eng*, 32(3):230–236, Mar 1985.

[5] T. Penzel, G. B. Moody, R. G. Mark, A. L. Goldberger, and J. H. Peter. The apnea-ecg database. *Computers in Cardiology*, 27:255–258, 2000.

[6] S. Raj, K. C. Ray, and O. Shankar. Development of robust, fast and efficient QRS complex detector: a methodological review. *Australas Phys Eng Sci Med*, 41(3):581–600, Sep 2018.

[7] F. Schrumpf, M. Sturm, G. Bausch, and M. Fuchs. Derivation of the respiratory rate from directly and indirectly measured respiratory signals using autocorrelation. *Current Directions in Biomedical Engineering*, 2(1):241–245, Sep 2016.

[8] F. Yasuma and J. Hayano. Respiratory sinus arrhythmia: why does the heartbeat synchronize with respiratory rhythm? *Chest*, 125(2):683–690, Feb 2004.