Signal Processing in Bioengineering
Instituto Superior Técnico, Lisboa
Lab 1 – Rehearsal – I/O (WAV, MAT and EDF) - BSP
March 15, 2021

João Saraiva
86449

All code was developed in a Python equivalent manner to MATLAB and is available at `https://github.com/jomy-kk/PDSBLabs` in `lab01` directory.

To make use of the same procedures of MATLAB Signal Processing toolbox, we may use the package `scipy.signal` and to read/write `wav` and `mat` files we may use the package `scipy.io`. To read/write EDF/EDF+ files we may use the `pyedflib` package. Plots and graphic work can be done using the `matplotlib` package. The remaining differences are just a matter of translation.

1 Reading WAV Files

We can open the `wav` files using the procedure `scipy.io.wavefile.read`, which returns an array of n data arrays, where n is the number of channels. Since for both $n = 1$, we assume both files have only one channel. For `signal.wav`, the sampling frequency is 256 Hz and its duration is about 15.32 s. For `tunes.wav`, the sampling frequency is 8000 Hz and its duration is 7 s. Both signals can be found in Figures 1 and 2. To play the sounds, the built-in function `playsoynd` can be used.

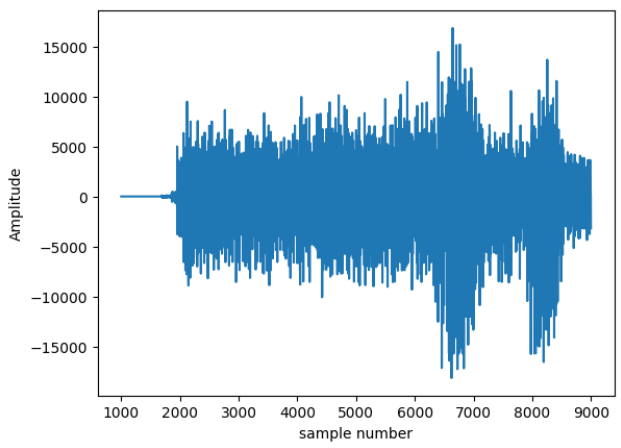

Figure 1: Envelope of `tunes.wav` starting at sample 1000 and with 1 second duration.

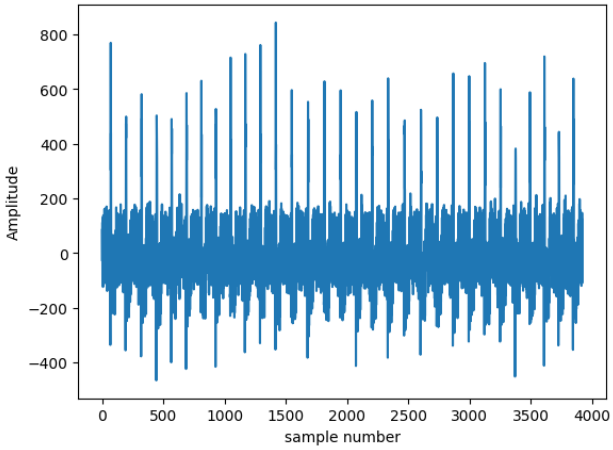

Figure 2: Full signal of `signal.wav`.

2 Writing WAV files

The procedure `sine` was created. It receives a frequency (in Hz), a sampling frequency (in Hz) and a duration (in seconds) and it returns the corresponding sinusoidal signal. Two signals sampled at 22 kHz were generated, one with sinusoidal frequency 200 Hz, and another with 3000 Hz. Short samples of their first seconds can be seen in Figures 3 and 4. A 2 second signal generated with sinusoidal frequency 888 Hz, sampled at 900 Hz was saved in `mysine.wav` and in `mysine.mat`, in their respective formats.

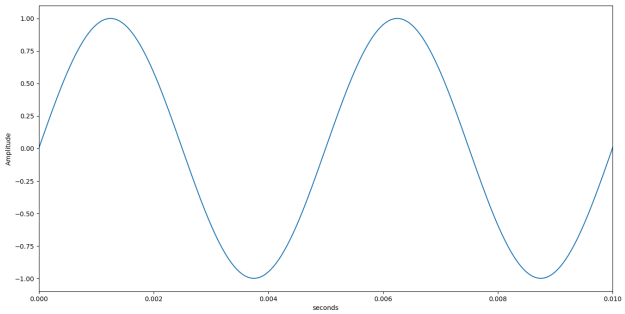

Figure 3: Envelope of the first 0.01 seconds of the generated sinusoidal of frequency 200 Hz. Sampling frequency: 22 kHz.

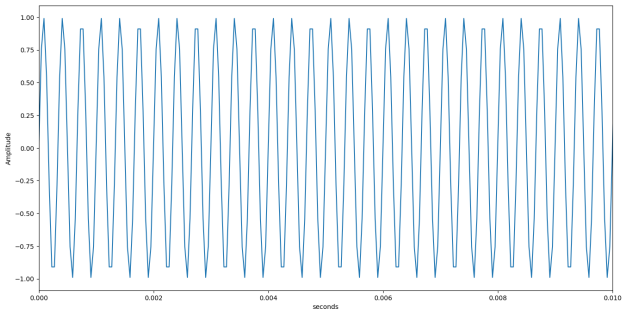

Figure 4: Envelope of the first 0.01 seconds of the generated sinusoidal of frequency 3000 Hz. Sampling frequency: 22 kHz.

3 Zero-crossing method

The procedure `zero_crossing` computes where the given signal amplitude crosses zero and returns a list of the sample indices of the signal after which a zero crossing occurs. For instance, if the sign of the amplitude changes between sample 3 and sample 4, it will return a list where 4 is included. With `numpy` this can actually be written in just one or two lines of code. For the proposed signal, the zero-crossing points are depicted in Figure 5.

An identical procedure to the one provided in `x_fzcross.m` is implemented in the function `avg_period`. It computes the average period of a given signal, using the zero-crossing method. This technique assumes a straight line approximation between adjacent points and interpolates between them to get x coordinates where approximately zeros occur, with three decimal places precision. For the proposed signal, the average period was found to be 0.5 seconds.

This is a very simple and inexpensive method, but it is not very accurate and does not deal with harmonics. It is mostly used to get a rough fundamental frequency of the signal. A better approach would be to apply the FFT to the signal to get a frequency spectrum and matching its highest peak to the signal's fundamental frequency.

4 Moving-average filter

The procedure `moving_average` receives a signal and a kernel size (N) and returns the filtered signal by a moving average window of size N. The application of this filter to the previously created `mysine.mat` signal with $N = 10$ and $N = 150$ can be found in Figure 6. We can notice that smoothing and attenuation are properties of the moving average filter, but for too large kernels the signal looses its profile getting too attenuated.

5 Reading EDF files and processing

The procedure `pyedflib.highlevel.read_edf` allows us to read an EDF file and return us three arrays:

- the complete signal of each channel;
- the header of each channel;
- the header of the file.

Reading the `russek_rc_reduced.edf` file we can found 15 channels, described below:

	Label	Rate (Hz)	Duration (s)
1	SAO2	60	320
2	BPM	60	320
3	SAO2Pleth	60	320
4	EEG C3	1024	320
5	EEG C4	1024	320
6	EMGQE	1024	320
7	EOGE	1024	320
8	ECG1	1024	320
9	ECG2	1024	320
10	MIC	1024	320
11	CANULA	1024	320
12	CINTA_TORACICA T	1024	320
13	CINTA_ABDOMINAL	1024	320
14	BODY_POSITION PO	1024	320
15	EEG PZ_RONCO	1024	320

The first 20 seconds of the channel ECG1 can be found in Figure 7. Next, the procedure `get_edf` is a pure translation of the provided `getedf.m` in MATLAB. It was used to read the signal in channel ECG2 and its first 20 seconds can be found in Figure 8.

A translation of the provided `findextremas.m` can be found in the procedure `find_extremas`. For both the signals in ECG1 and ECG2, the implemented procedure yields the same peak sites as the already established `scipy.signal.find_peaks` does. These peaks can be assumed to be the R peaks of the QRS complexes. They are depicted for ECG1 in Figure 9 for the first minute. A prominence of 1000 was used.

Also, the R-R interval (RRI) in milliseconds is plotted in Figure 10. The average RRI of the whole signal is about 642 ms.

Next, the CINTA_TORACICAT channel was loaded and by counting the number of breaths for each minute (counting how many times the chest rises), the mean respiration rate per minute (RPM) was found to be 15.5 breaths per minute. This is okay, since normal respiration rates for an adult person at rest range from 12 to 16 breaths per minute. The "instantaneous" RPM is depicted in Figure 11. Of course, we should ignore the first seconds, and we can see it converges to the mean RPM.

6 Reading EDF files and filtering

A snapshot of the first 20 seconds of all channels of `russek_rc_reduced.edf` can be found in `/lab01/ex6/sc1.png`. An IIR notch filter was designed using `scipy.signal.irrnotch` with the required specifications, and applied forward and backwards to the specified channels using `scipy.signal.filtfilt`. The system frequency response can be found in Figure 12. The filtered signals can be found in a snapshot of their first 20 seconds in `/lab01/ex6/sc2.png`.

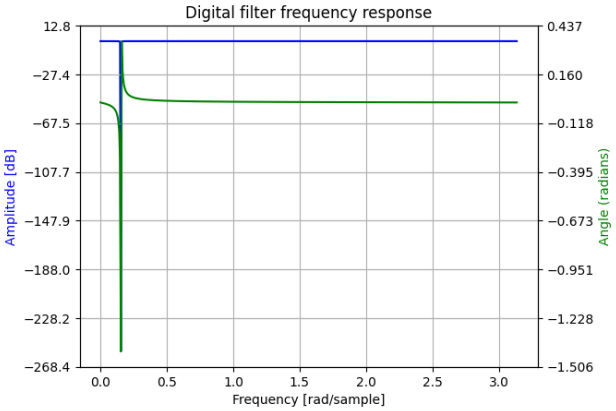

Figure 12: Bode diagram of the notch filter designed to reject power line at 50 Hz, with a quality factor of 35.

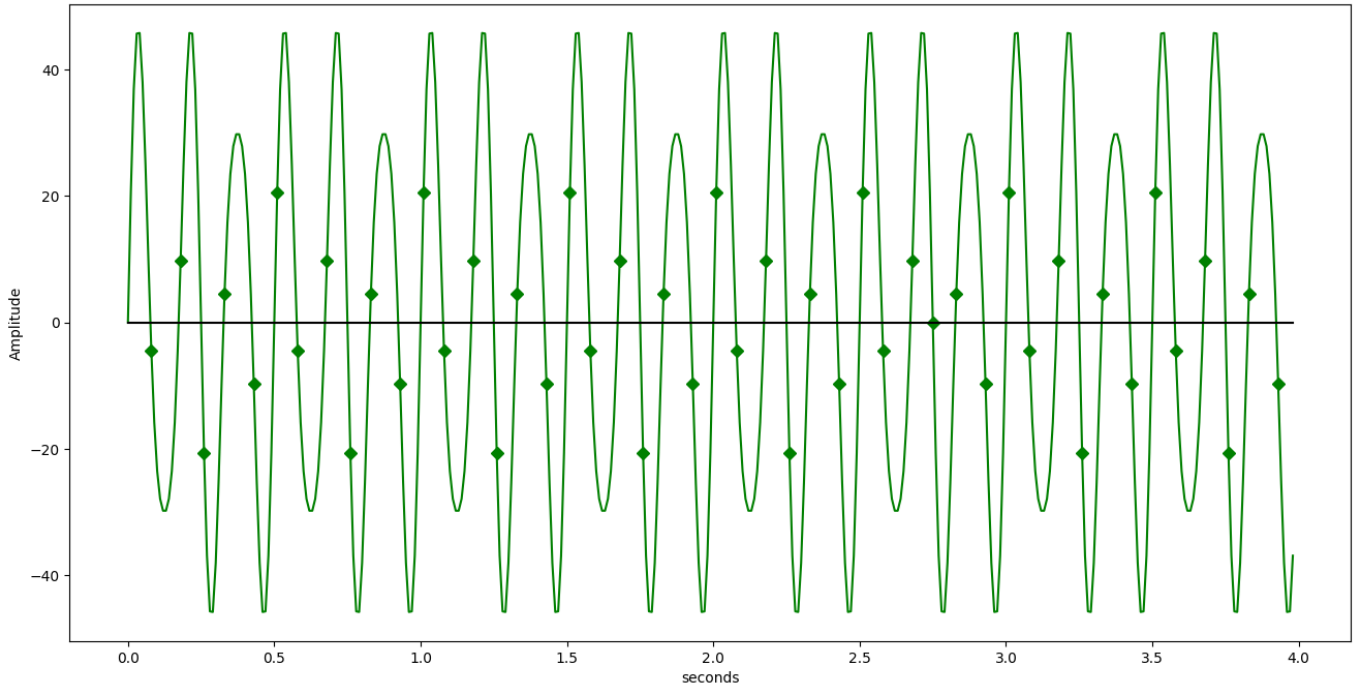

Figure 5: Generated signal (composed by 2 sinusoidal waves) and its highlighted points after zero crossing.

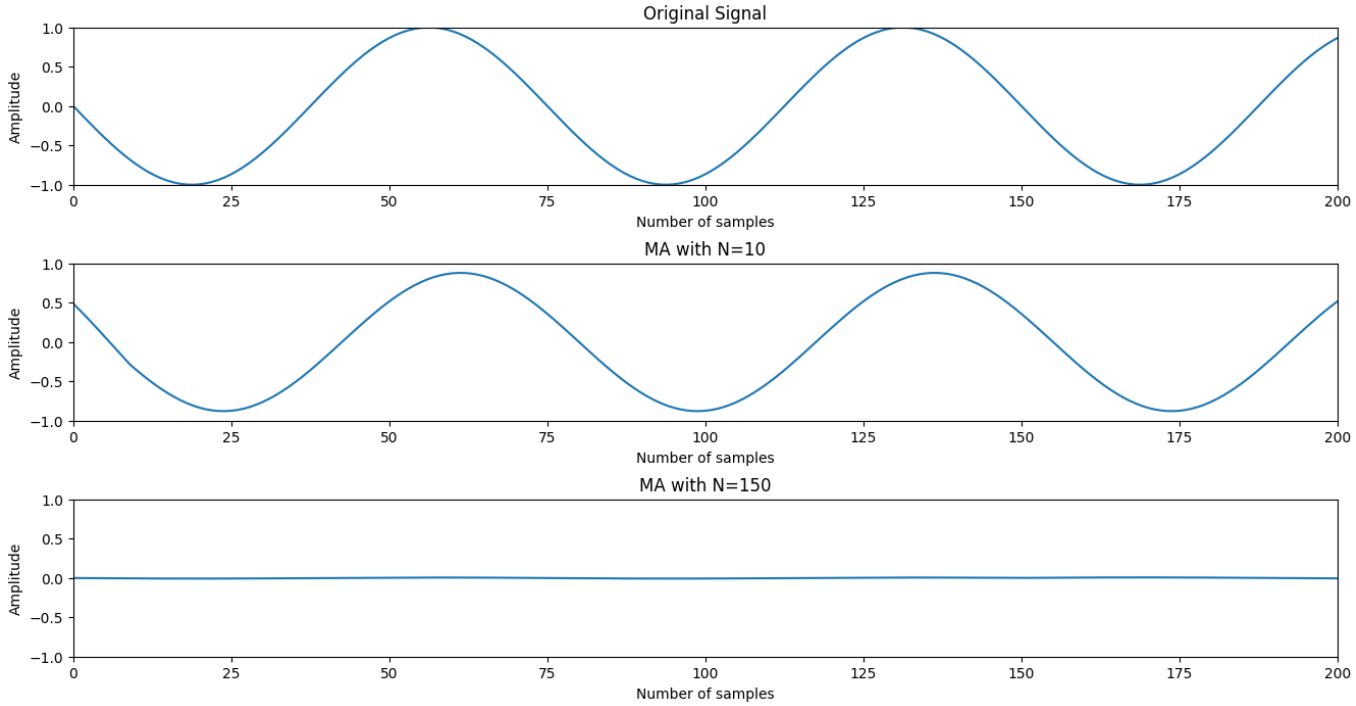

Figure 6: Envelopes of the first 200 samples of the original signal `mysine.mat` and its moving-average filtered correspondents with $N = 10$ and $N = 150$.

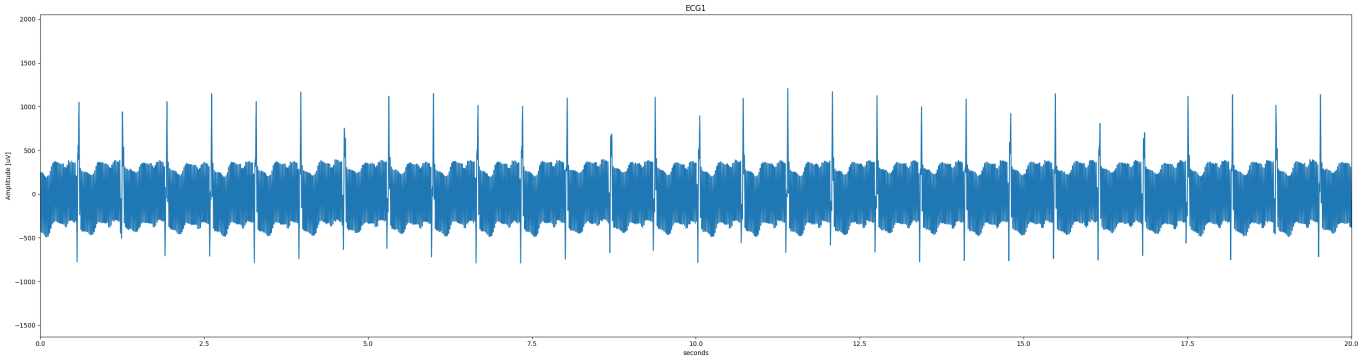

Figure 7: Envelope of the first 20 seconds of channel ECG1 of `russek_rc_reduced.edf`.

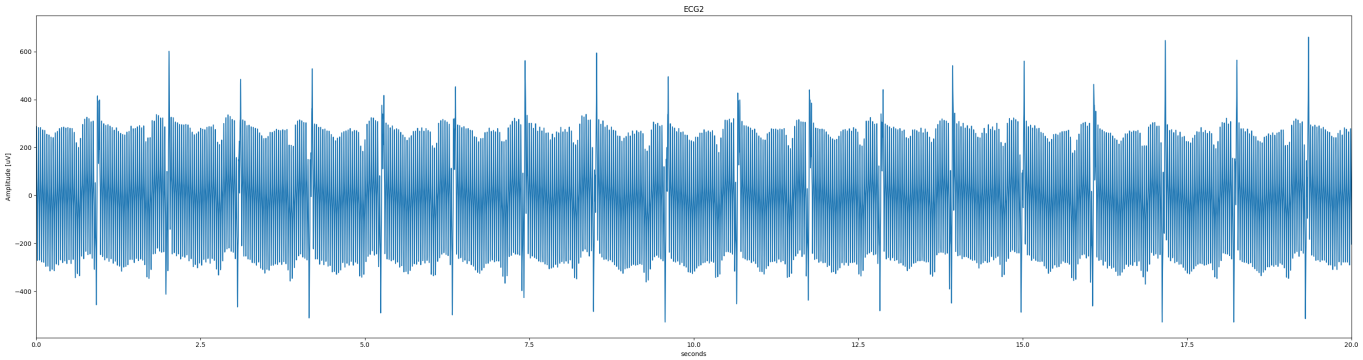

Figure 8: Envelope of the first 20 seconds of channel ECG2 of `russek_rc_reduced.edf`.

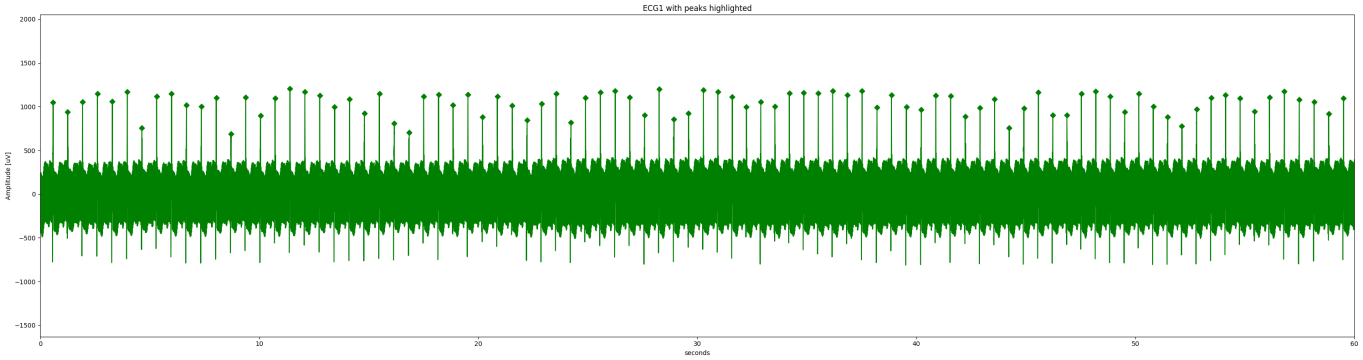

Figure 9: Estimated R peaks of channel ECG1 of `russek_rc_reduced.edf` for the first 60 seconds.

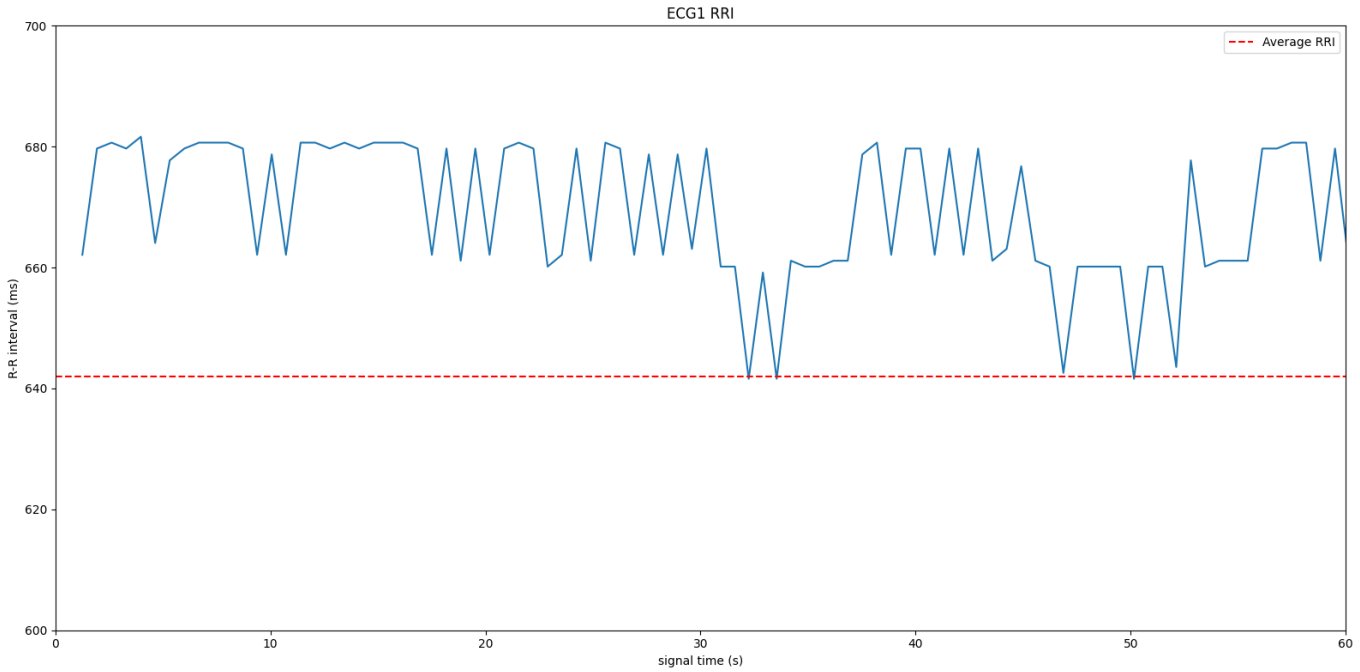

Figure 10: Estimated R-R interval of channel ECG1 of `russek_rc_reduced.edf` for the first 60 seconds.

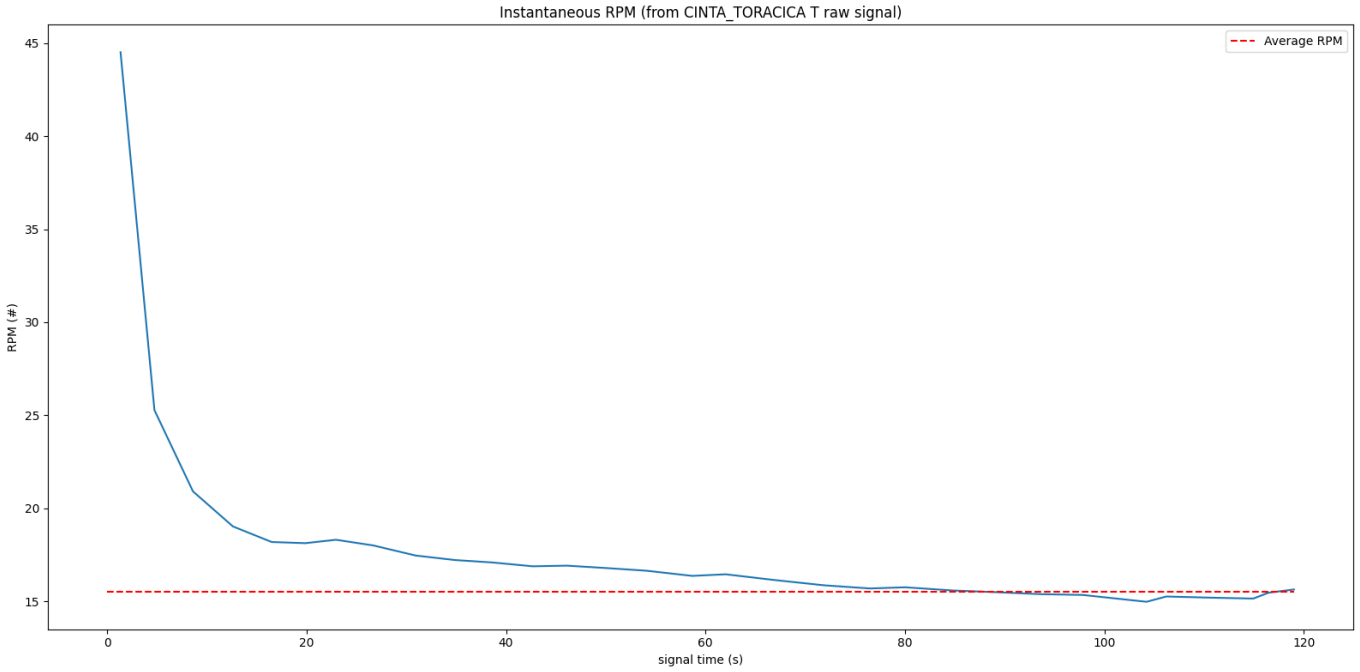

Figure 11: Estimated "instantaneous" RPM of channel CINTA_TORACICAT of `russek_rc_reduced.edf` for the first 120 seconds.