# Non-invasive extraction of the fetal electrocardiogram

## 1. Contextualizing the Problem

Ensuring the safety and well-being of the fetus during pregnancy is crucial, and prenatal care plays a significant role in achieving this goal. Early detection of diseases and congenital problems is essential, but it can be challenging to infer the fetus' vital signs while prioritizing the mother's comfort and safety. Therefore, the development of non-invasive methods like NI-fECG is necessary.

NI-fECG technology has numerous benefits, including its non-invasive nature, which eliminates the need to pierce or puncture the skin, thus preventing infections and discomfort for the mother. It also provides accurate and continuous monitoring, enabling healthcare providers to detect changes in heart rate pattern and offer precise diagnoses.

Despite its advantages, NI-fECG has certain limitations. It is not widely available in all healthcare settings, and the required equipment and personnel can be expensive, making it less affordable for some patients and healthcare facilities. Additionally, maternal signals, such as heart rate and muscle activity, can interfere with the fetal signal.

However, ongoing improvements are being made in this field. Signal processing techniques such as adaptive filtering and wavelet analysis have been introduced to enhance measurement accuracy. Additionally, machine learning approaches like deep learning have been investigated, showing promising results in detecting fQRS complexes.

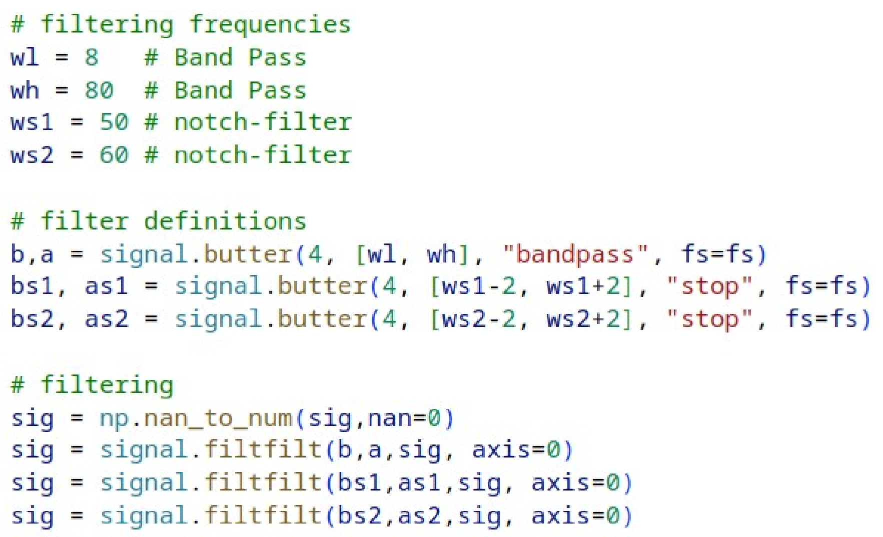
In conclusion, the development of NI-fECG technology is vital for improving prenatal care's safety and comfort. Although there are advantages and limitations, the technology is continually evolving and improving, making it an area of focus for ongoing development.

The objective of this work is to have a working algorithm for extracting the fetal ECG from electrodes placed on the mom’s abdomen. The steps taken and the results obtained will be exposed and commented on. In addition, possible improvements on the work done will be highlighted.

# 2. The Algorithm In Summary

## 2.1. Preprocessing

For preprocessing the channels, the same approach as [2] was used with a single modification made: the cutoff frequency for the highpass filter was increased to 8Hz as that was shown in [1] to be better performing.

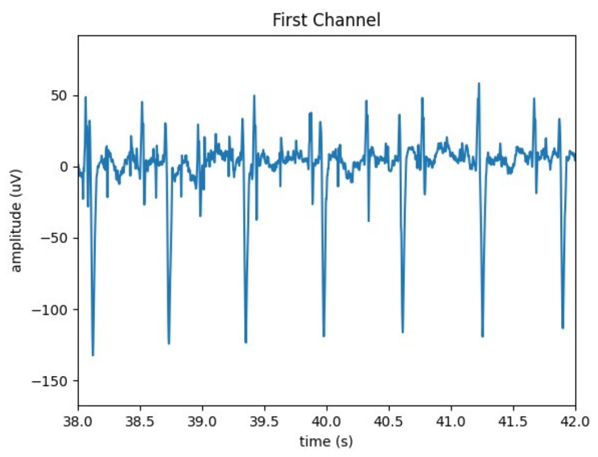


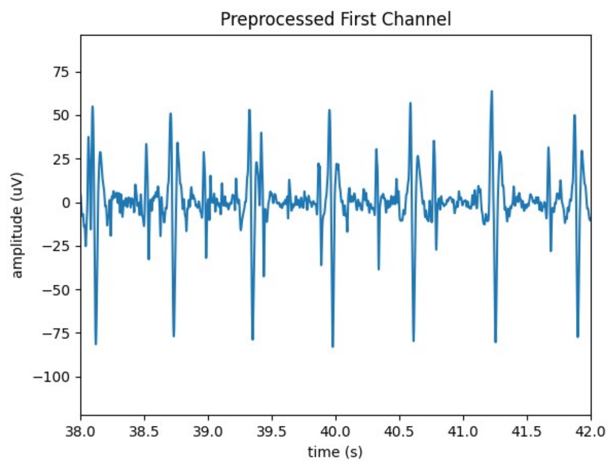
```
# filtering frequencies
wl = 8    # Band Pass
wh = 80   # Band Pass
ws1 = 50 # notch-filter
ws2 = 60 # notch-filter

# filter definitions
b,a = signal.butter(4, [wl, wh], "bandpass", fs=fs)
bs1, as1 = signal.butter(4, [ws1-2, ws1+2], "stop", fs=fs)
bs2, as2 = signal.butter(4, [ws2-2, ws2+2], "stop", fs=fs)

# filtering
sig = np.nan_to_num(sig,nan=0)
sig = signal.filtfilt(b,a,sig, axis=0)
sig = signal.filtfilt(bs1,as1,sig, axis=0)
sig = signal.filtfilt(bs2,as2,sig, axis=0)
```

*Figure 1: code for the preprocessing step*

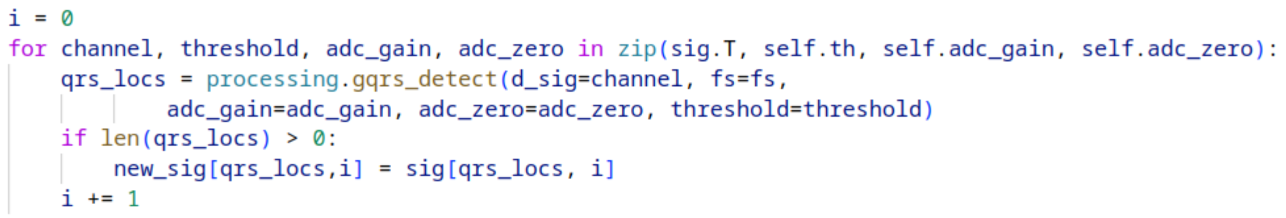




Not much thought was put into chosing the preprocessing method, it'd be best if there was made a benchmark. A good starting point would be to use Mr. Collette's preprocessing approach and check to see if the performance is improved. Another important imrpovement consists of checking for muddled channels, e.g. channel 1 from test-set-a/a55. This isn't currently done and would definitelly improve the scores.

## 2.2. mQRS Detection

For adult QRS detection there's a wide range of possible algorithms to use. [3] has made some benchmarking and from the ones in the study the "gqrs" from the wfdb package was picked. This particular algorithm has the advantage of being implemented in matlab and python and it's both wildly used and well performing, seemingly being a good fit for this work.



```python
i = 0
for channel, threshold, adc_gain, adc_zero in zip(sig.T, self.th, self.adc_gain, self.adc_zero):
    qrs_locs = processing.gqrs_detect(d_sig=channel, fs=fs,
            adc_gain=adc_gain, adc_zero=adc_zero, threshold=threshold)
    if len(qrs_locs) > 0:
        new_sig[qrs_locs,i] = sig[qrs_locs, i]
    i += 1
```

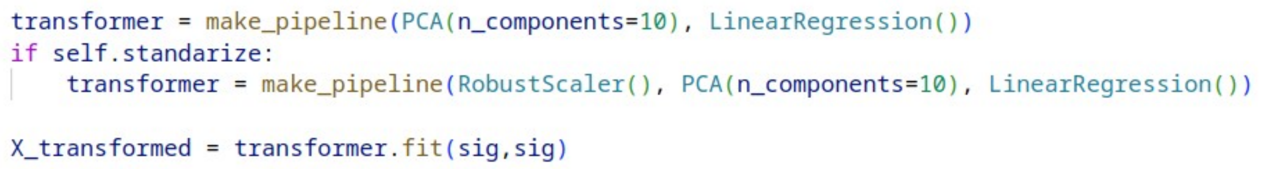
*Figure 2: code for the mQRS detection*

## 2.3. Template Creation

As *template subtraction* is the approach used, a key step in the algorithm has to be creating the template. The approach used follows what was done in [4] and it makes use of *Primary Component Regression* to generate the template.

This is probably the biggest step in the entire process. It consists in the following sub-steps:

1. Defining the **PCR** model. For defining the model the implementations inside the sklearn python package were used.

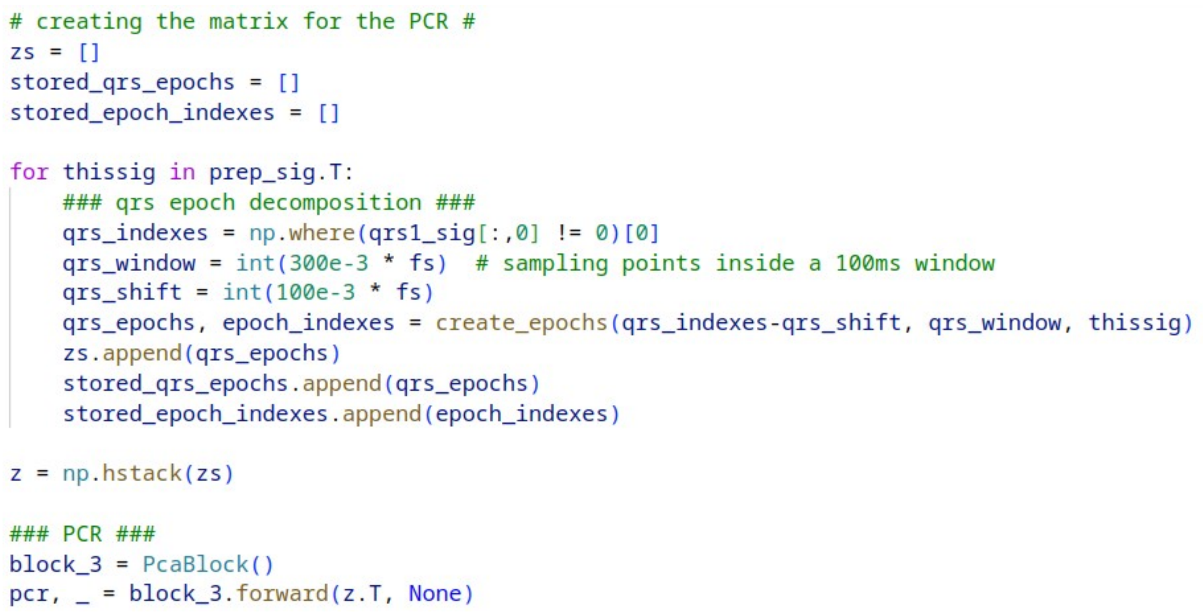


```python
transformer = make_pipeline(PCA(n_components=10), LinearRegression())
if self.standarize:
    transformer = make_pipeline(RobustScaler(), PCA(n_components=10), LinearRegression())

X_transformed = transformer.fit(sig,sig)
```

*Figure 3: code for definiting and fitting the data into the PCR model*

2. Creating a matrix z made up of extracted mQRS complexes. This matrix will later be fed into the **PCR** model which assumes that each column of the matrix is a sample and that each row is a feature. For this reason the extracted mQRS epochs will need to be of the same length and they'll be inserted in column-wise in the matrix.



```python
# creating the matrix for the PCR #
zs = []
stored_qrs_epochs = []
stored_epoch_indexes = []

for thissig in prep_sig.T:
    ### qrs epoch decomposition ###
    qrs_indexes = np.where(qrs1_sig[:,0] != 0)[0]
    qrs_window = int(300e-3 * fs)  # sampling points inside a 100ms window
    qrs_shift = int(100e-3 * fs)
    qrs_epochs, epoch_indexes = create_epochs(qrs_indexes-qrs_shift, qrs_window, thissig)
    zs.append(qrs_epochs)
    stored_qrs_epochs.append(qrs_epochs)
    stored_epoch_indexes.append(epoch_indexes)

z = np.hstack(zs)

### PCR ###
block_3 = PcaBlock()
pcr, _ = block_3.forward(z.T, None)
```

*Figure 4: code for creating the matrix z and feeding it into the PCR model*

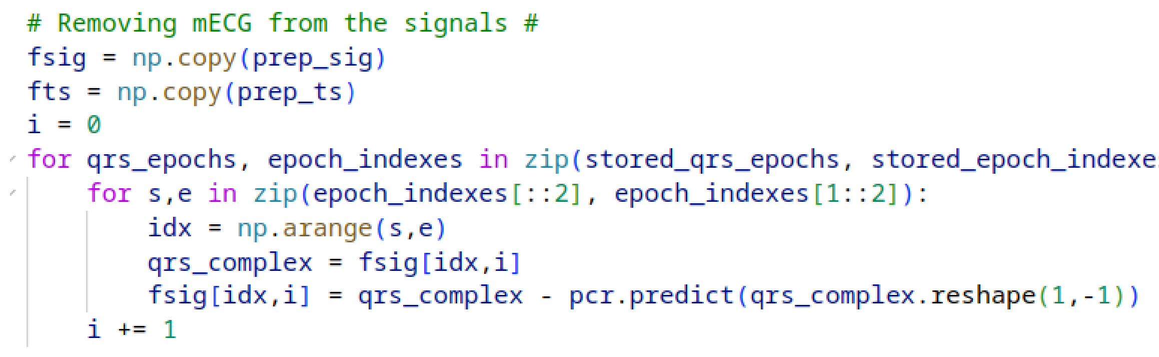
Afeter feeding the data into the **PCR** block, a model for predicting the regression of the mQRS complexes is obtained and will be used to perform the *template subtraction* in the next step

It's not well understood if the implemantation done here operates exactly as expected. It'd be interesting to look better into what was done in [4] and see if the current script follows the instructions correctly.

It's also important to note that [4] claims to also perform these steps for removing the P and T-waves, although it wasn't made very clear the steps they took to do so and thus it wasn't reproduced. neurokits2 has an existing implementation for detecting P and T-waves and might be a good option for solving this.

## 2.4. Template Subtraction

This step consists in removing the predicted mQRS complexes from the original signal, hopefully leaving only the fQRS signal and some noise.



```python
# Removing mECG from the signals #
fsig = np.copy(prep_sig)
fts = np.copy(prep_ts)
i = 0
for qrs_epochs, epoch_indexes in zip(stored_qrs_epochs, stored_epoch_indexes):
    for s,e in zip(epoch_indexes[::2], epoch_indexes[1::2]):
        idx = np.arange(s,e)
        qrs_complex = fsig[idx,i]
        fsig[idx,i] = qrs_complex - pcr.predict(qrs_complex.reshape(1,-1))
    i += 1
```

*Figure 5: code for removing the mQRS*

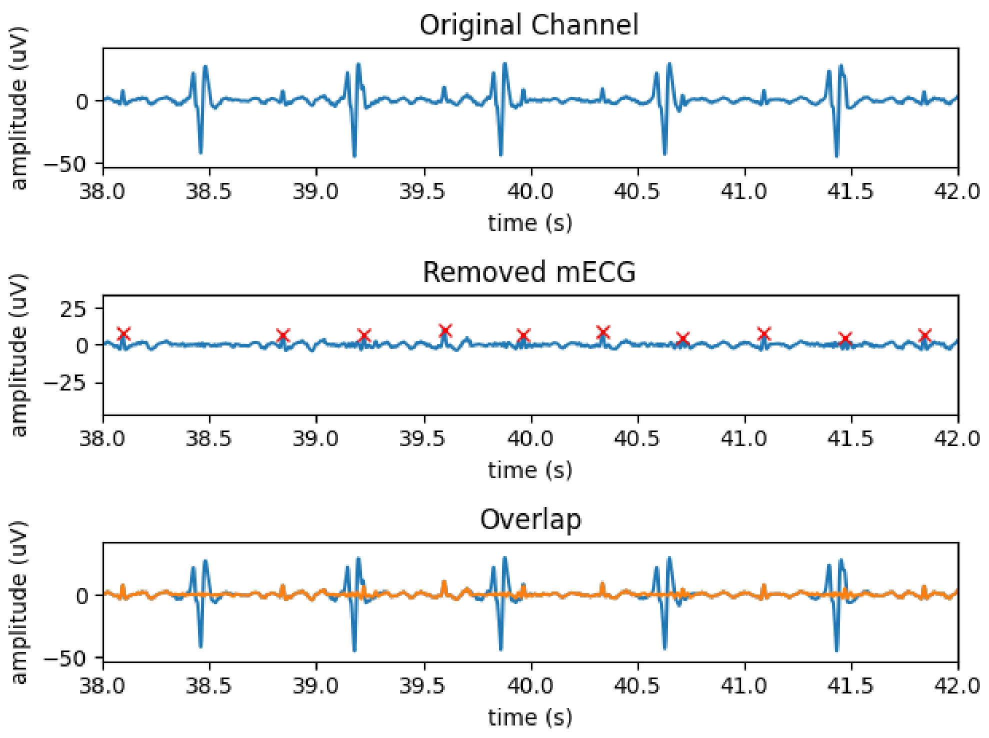
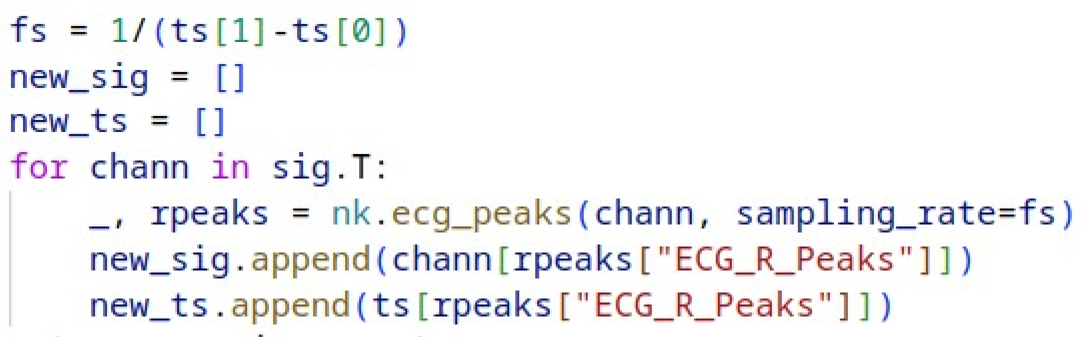


*Figure 6: channel before and after the template subtraction. The last plot shows an overlapping comparison*

## 2.5. fQRS Detection

Next, it's necessary to detect the QRS complexes in the resulting channels. As [1] pointed out, there is no real benchmarking done on fQRS detection algorithms and the current implementations mostly consist on adult QRS detection algorithms with some changes on its parameters. We've used the default implementation present in the python package (analysed in [6]) that seemed to be the best performing one.



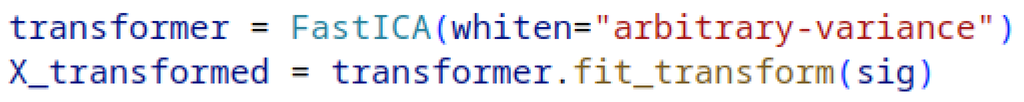
```
fs = 1/(ts[1]-ts[0])
new_sig = []
new_ts = []
for chann in sig.T:
    _, rpeaks = nk.ecg_peaks(chann, sampling_rate=fs)
    new_sig.append(chann[rpeaks["ECG_R_Peaks"]])
    new_ts.append(ts[rpeaks["ECG_R_Peaks"]])
```

*Figure 7: code for detecting fQRS complexes*

This algorithm isn't suited for fetal QRS detections and better ones could be found, a deeper investigation is needed.

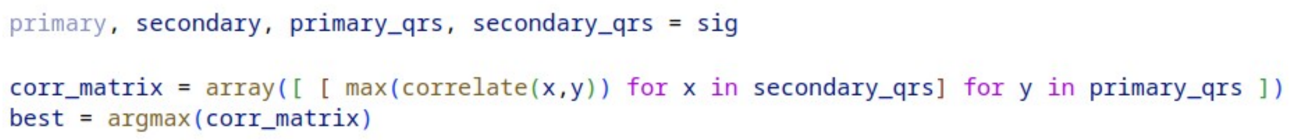
## 2.6. Channel Selection

Lastly, the best channel should be selected. Instead of picking from within the available ones, [1] sugested to pass them through an **ICA** step, allowing for better results. The best **ICA** channel will be the one whose QRS complexes correlate the best with the original fECG channels.



```python
transformer = FastICA(whiten="arbitrary-variance")
X_transformed = transformer.fit_transform(sig)
```

*Figure 8: code for defining the ICA model*



```python
primary, secondary, primary_qrs, secondary_qrs = sig

corr_matrix = array([ [ max(correlate(x,y)) for x in secondary_qrs] for y in primary_qrs ])
best = argmax(corr_matrix)
```

*Figure 9: code for selecting the best channel. The primary channels are the original ones and the secondary channels are the ones resulting from the ICA*

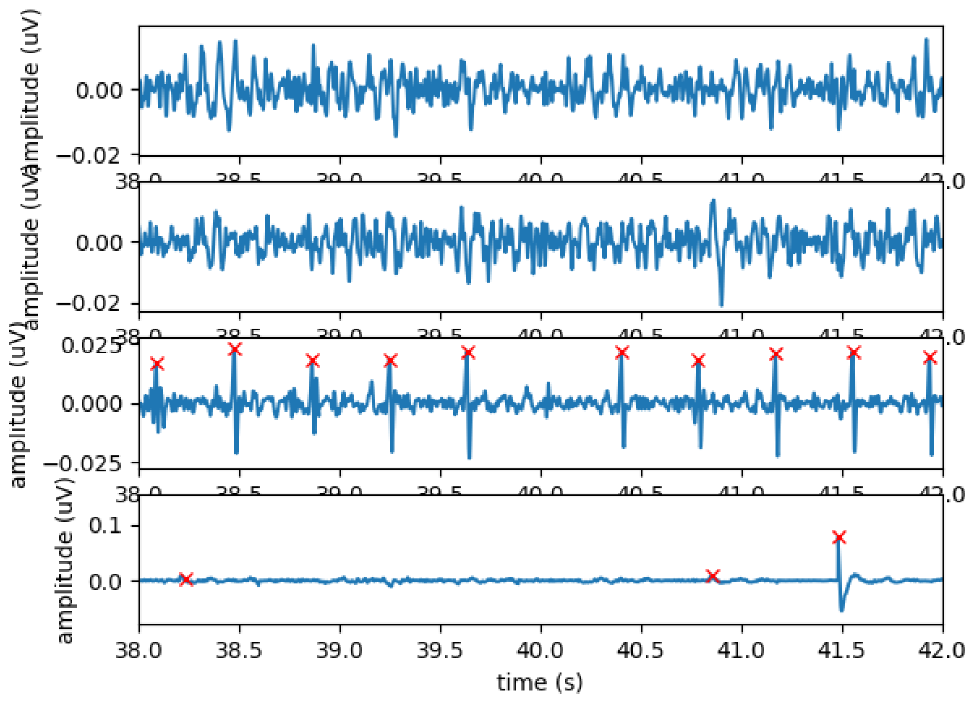


*Figure 10: all resulting ICA channels, the one selected after the channel selection step will be the third one from top to bottom*

## 3. Results

In the end, we can see that for well behaved signals this source separation algorithm works incredibly well, as for figure 12. However, by looking figure 11 allows it’s possible to conclude that there are some troublesome situations. One such case is seen in figure 13.

There are many reasons for this performance issue and implementation issues are not discarted, but it’s important to restate that many external factors can polute to the source signals such as the weight of the mother, the mother’s position while recording the signal and even the position of the fetus during the recording.

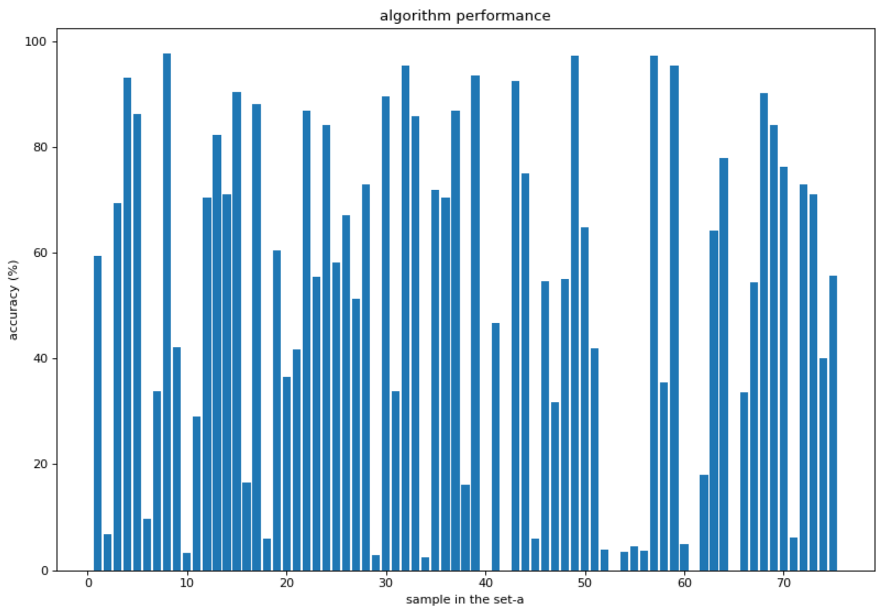
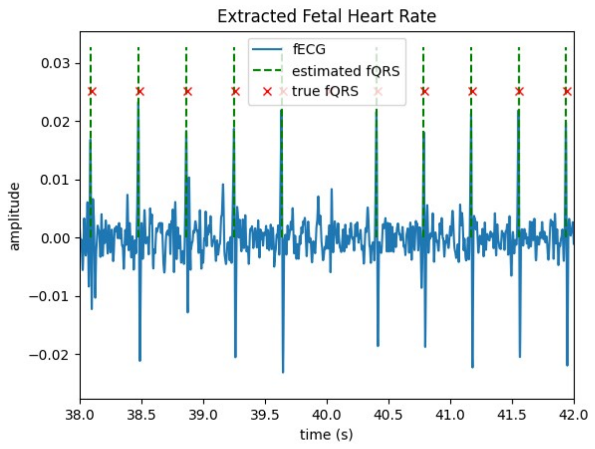


*Figure 11: algorithm accuracy using the training samples from the populartion a from the physio challange 2013*



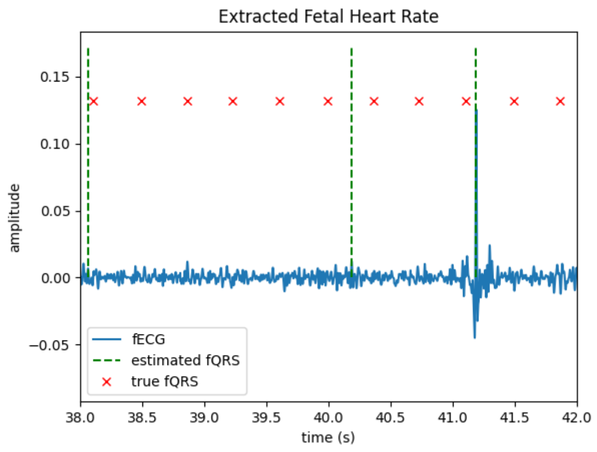


*Figure 13: results from the sample a32*

*Figure 12: results from the sample a02*

## 4. References

1. Andreotti F, Behar J, Zaunseder S, Oster J, Clifford GD. An open-source framework for stress-testing non-invasive foetal ECG extraction algorithms. Physiol Meas. 2016 May;37(5):627-48. [PMID: 27067286]

2. F. Andreotti et al. Maternal signal estimation by Kalman filtering and Template Adaptation for fetal heart rate extraction. In: Computing in Cardiology 2013. Zaragoza, Spain, 2013. p. 193-196.

3. A. E. Johnson, J. Behar, F. Andreotti, G. D. Clifford, and J. Oster. R-peak estimation using multimodal lead switching. In: Computing in Cardiology 2014. Cambridge, MA, USA, 2014. p. 281-284.

4. Lipponen JA, Tarvainen MP. Principal component model for maternal ECG extraction in fetal QRS detection. Physiol Meas. 2014 Aug;35(8):1637-48. [PMID: 25069651]

5. scikit-learn developers. (n.d.). Cross decomposition: Comparing PLS and PCR. scikit-learn. Retrieved March 28, 2023, from https://scikit-learn.org/stable/auto_examples/cross_decomposition/plot_pcr_vs_pls.html

6. Brammer JC. biopeaks: a graphical user interface for feature extraction from heart- and breathing biosignals. Journal of Open Source Software. 2020;5(54):2621. doi: 10.21105/joss.02621.