SELECTED TOPICS IN BIOMEDICAL SIGNAL PROCESSING: DATA-DRIVEN FILTER DESIGN FOR BIOMEDICAL SENSOR ARRAYS

1. EEG artifact removal

1.1. Unsupervised artifact removal using CCA

Visualize the EEG in eegdata. You can use the eegplot_simple.m *function for it. Alternatively, the* eegplot *function of the EEGLAB toolbox can also be used for visualization.*

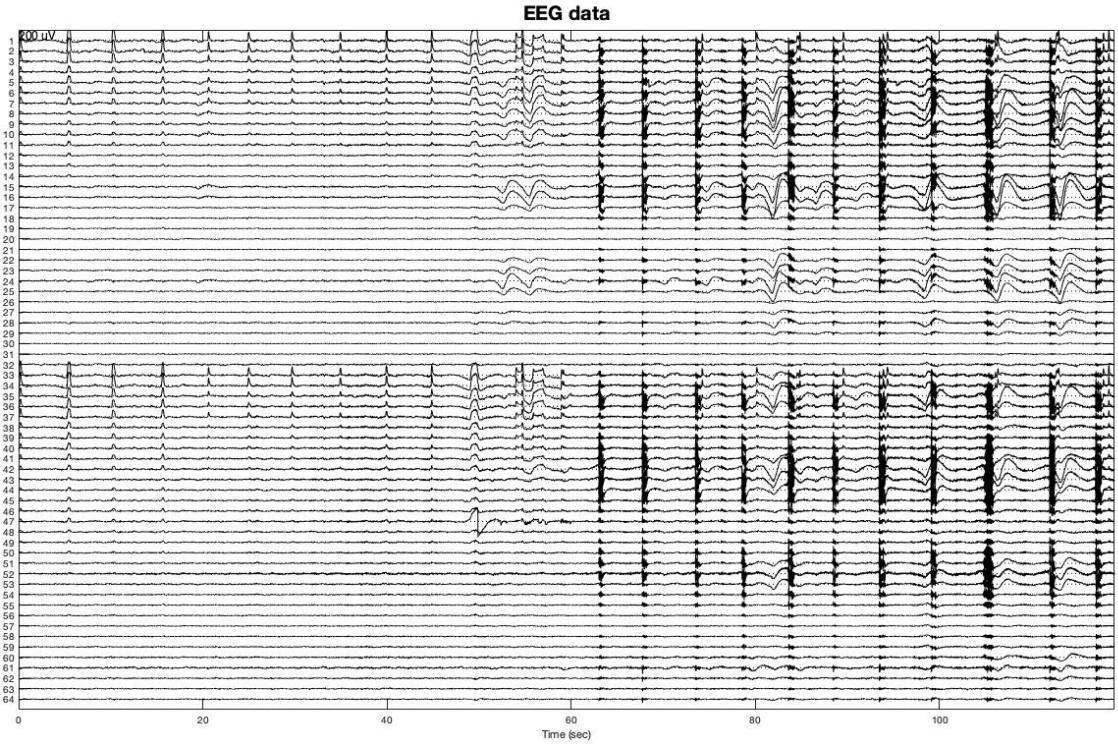

Figure 1: EEG data

Let us first consider the two different artifacts present in the EEG, namely eye-blink and muscle artifacts. Try to locate them in the data. Mention at what times the artifacts are predominantly observed. Are all channels equally affected by both artifacts? Why or why not?

In this EEG data we can observe two different artifacts:

- **Eye-blink artifact:** it is shown as little peaks during the first half of the time measurements. These peaks are visible only in some channels, 1-7 and 33-42 more or less. The first group of channels correspond to Fp1, AF7, AF3, F1, F3, F5 and F7 electrodes (marked in orange); and the second group of channels, to Fpz, Fp2, AF8, AF4, AFz, Fz, F2, F4, F6 and F8 electrodes (marked in green).
  We could expect these results because the amplitude of the EEG waves depends on the distance between the electrode and the point where the signal was produced. As we can see, the channels mentioned before are those corresponding to the electrodes located near the eyes.

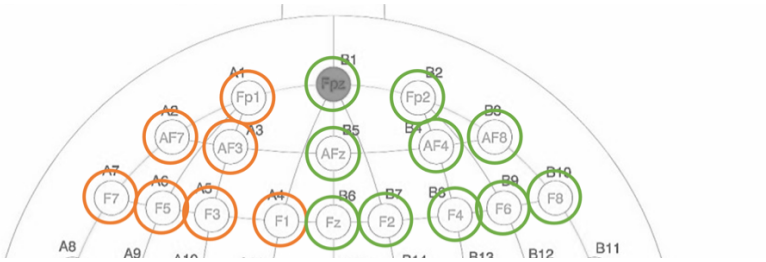

Figure 2: Electrodes with eye-blink artifact

- **Muscle artifact:** we can observe muscle artifacts in the second half of the time measurements. These artifacts are characterized for their high frequencies and are generated by the motor unit potentials.

Apply CCA as a blind source separation algorithm to the EEG data and estimate the sources. Visualize all sources found by CCA and plot them (you should have an equal number of sources as there are channels in EEG).

The canonical correlation analysis can be used as a blind source separation method by applying it to a signal and its instantly delayed, so we use d=1. Our EEG data is the result of multiplying several sources by a mixing matrix A, so to estimate those sources first we have to estimate the demixing matrix W.

$$X(t) = A \cdot S(t) \rightarrow \hat{S}(t) = W \cdot X(t)$$

We first obtained the W using the *canoncorr* function provided by MATLAB and then multiplied it by our EEG measurements $X(t)$. We plotted the resulting estimated sources $\hat{S}(t)$ and got the following figure:

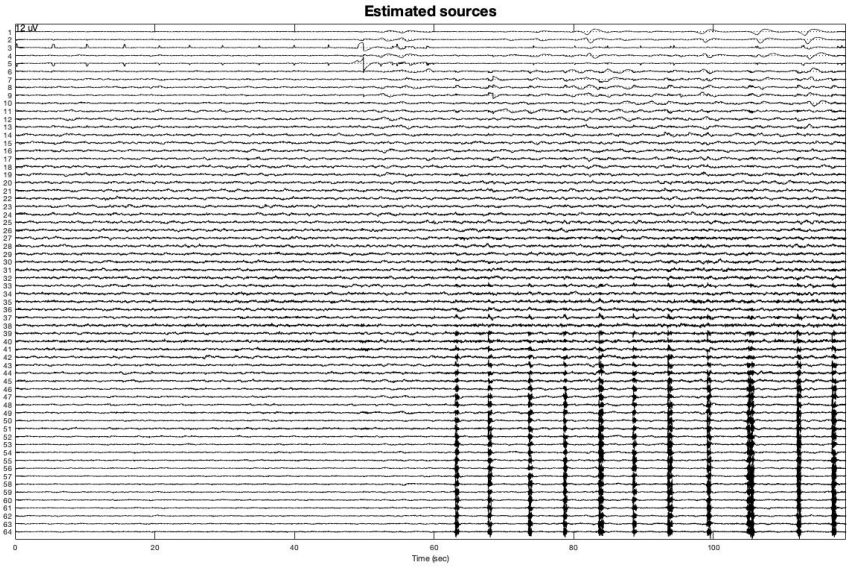

Figure 3: Estimated sources

Explain how you can reconstruct the EEG using the estimated sources such that only muscle artifacts are reduced. Illustrate the artifact reduction by plotting one or more EEG channels before and after removal. Zoom in on a few artifacts to illustrate the quality of artifact removal.

The CCA algorithm searches for two filters that return two signals that are maximally correlated. We can use this property to remove the muscle artifact because the EEG signal and the muscle artifact have a very low autocorrelation.

We try to find the maximum correlation, so we compute the eigenvalues and eigenvectors, where the correlation is related to the eigenvalues. As we are trying to filter the muscle artifact, with the lowest correlation, we should cut off the lowest eigenvalues of the estimated sources, placed them in a new vector $\widehat{S_{cut}}(t)$, and calculate the EEG signal again with the resulting matrix $X_{clean}(t)$.

In the following image, we can see that the muscle artifacts belong to the last channels; they appear, more or less, from channel 37 to channel 64. This first channel value corresponds to the eigenvalue located in the elbow of the autocorrelation graph.

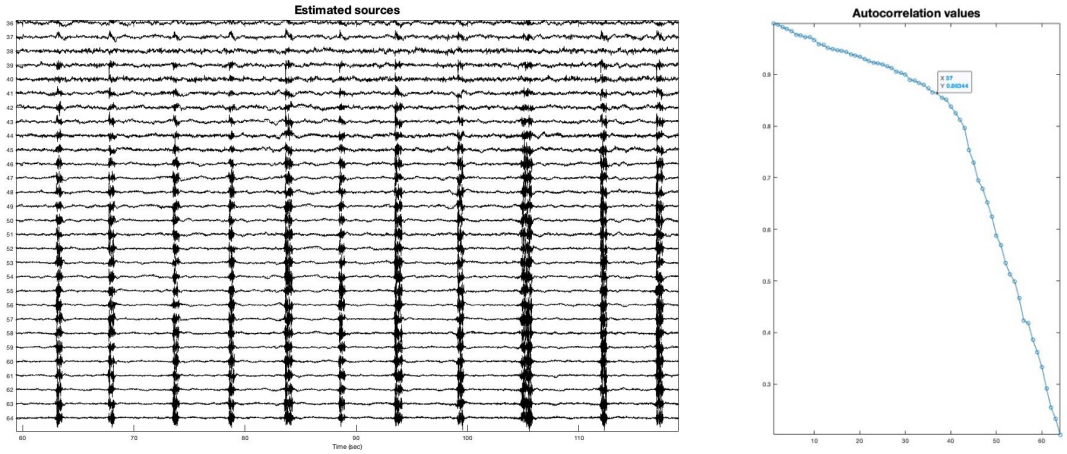

Figure 4: Estimated sources. Autocorrelation values

After removing the sources that caused the muscle artifact, we obtain a clean EEG signal. We can observe in the following image the original EEG data, plotted in green, and the clean EEG data, in blue; most of the artifact has been removed successfully.

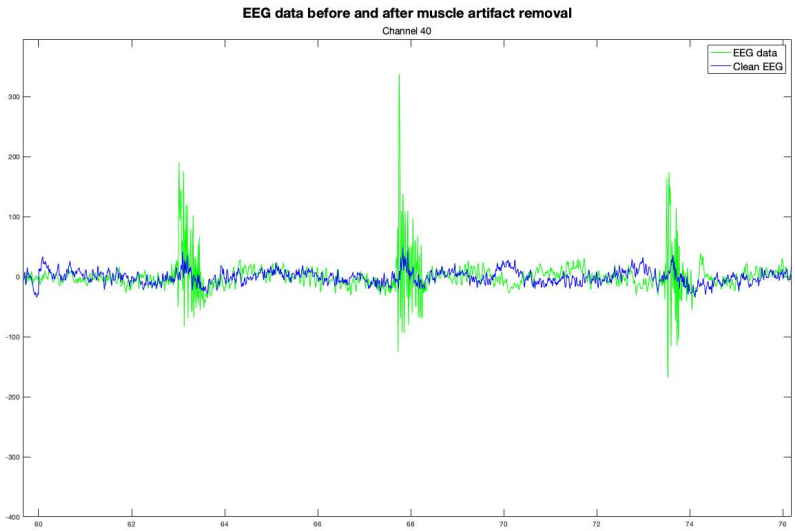

Figure 5: EEG data after artifact removal

Could we also remove eye-blink artifacts from EEG using CCA? If so, describe the strategy to achieve eye-blink artifact removal.

We can expect that the eye-blink artifact cannot be removed as the muscle artifact due to the correlation with the EEG signals. As the correlation is very high, we should cut the estimated sources corresponding to those channels; however, we would be deleting information related to the clean EEG signal, so we would be losing many precious data.

We could solve this by introducing time lags and having some EEG data without any eye-blink artifact so the correlation between those measurements and the ones among the eye-blin peaks would be even higher. Hence, we would be able to cut some eigenvalues, but being aware of the threshold we take to make that cut.

1.2. Supervised artifact removal using MWF

Use the mwf_getmask *and* mwf_process *functions of the mwf toolbox to remove eye blink artifacts from the EEG. Mark all the eye blink artifacts in the first 30 seconds of data while creating the mask required for MWF filter computation. Do NOT mark the muscle artifacts (yet). Ignore the delay parameter of* mwf_process *for now. You can use just the EEG data and the mask created using* mwf_getmask*.*

To remove the blink-eye artifact present in the EEG signal with the multi-channel Wiener filter we first have to design the mask, using the function *mwf_getmask*. This function will show a GUI where we have selected some artifact intervals during the first 30 seconds of data:

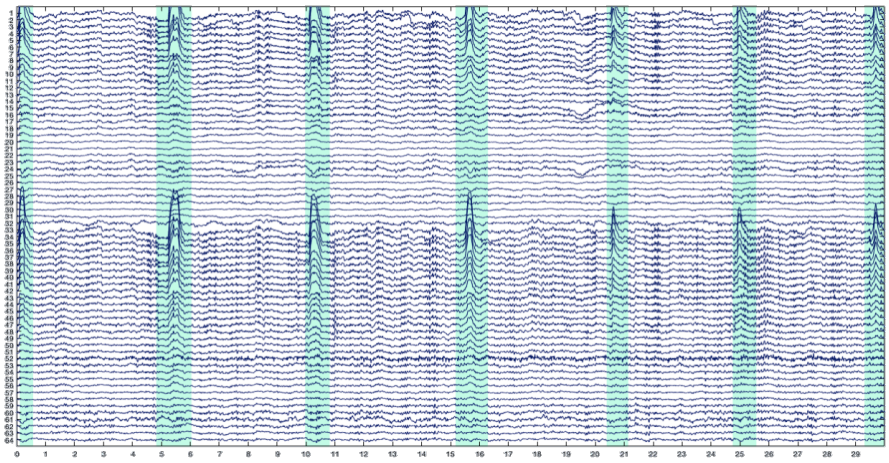

Figure 6: GUI for mask creation

Plot the raw-EEG, estimated artifacts and MWF filtered EEG signal for one representative channel (for e.g. 30 sec duration) on the same figure with different colors. Zoom in on a few artifacts to illustrate the quality of artifact removal.

After getting the mask, the function *mwf_process* gave us the clean EEG signal and the artifact that was removed, which is shown in the following picture. In the first graph, we see the original EEG signal in black and the artifact that will be removed in red. We can appreciate that all the peaks are marked as part of the artifact signal. On the other graph, we can see how the clean EEG in blue is quite similar to the original one, but without the peaks corresponding to the eye-blink artifact.

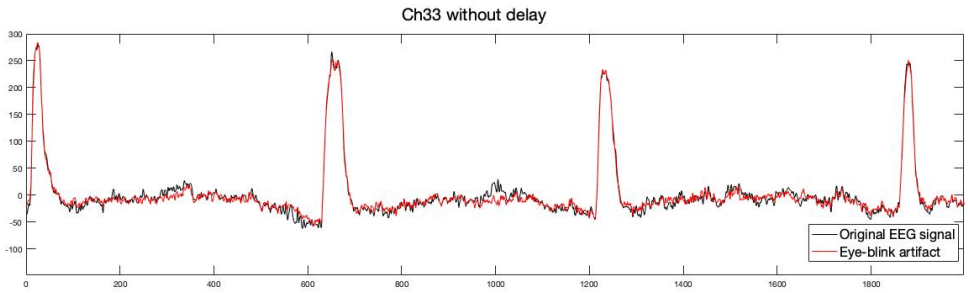

Figure 7: raw-EEG and estimated artifacts

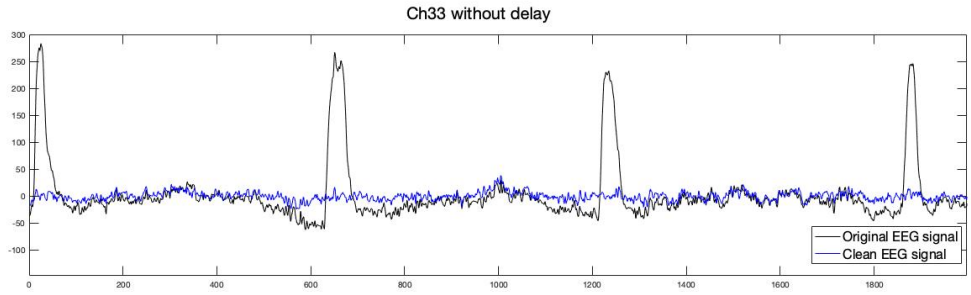

Figure 8: raw-EEG and MWF filtered EEG signal

The MWF filter generated using mwf_process is a purely spatial filter. Why? Consider the optional 'delay' parameter of the mwf_process function. What does it do? Redo item 1 but now with 3 as the delay parameter. Note that the mask remains the same. Carry out item 2 with the new results.

The filter used for these estimations is a purely spatial filter because it combines several channels in an instantaneous way, without taking into account the time variable; it is time independent. We can make it a spatio-temporal filter adding some delay in the function. The delay is the number of time lags included in the MWF; including these time lags increases the estimation of the clean signal.

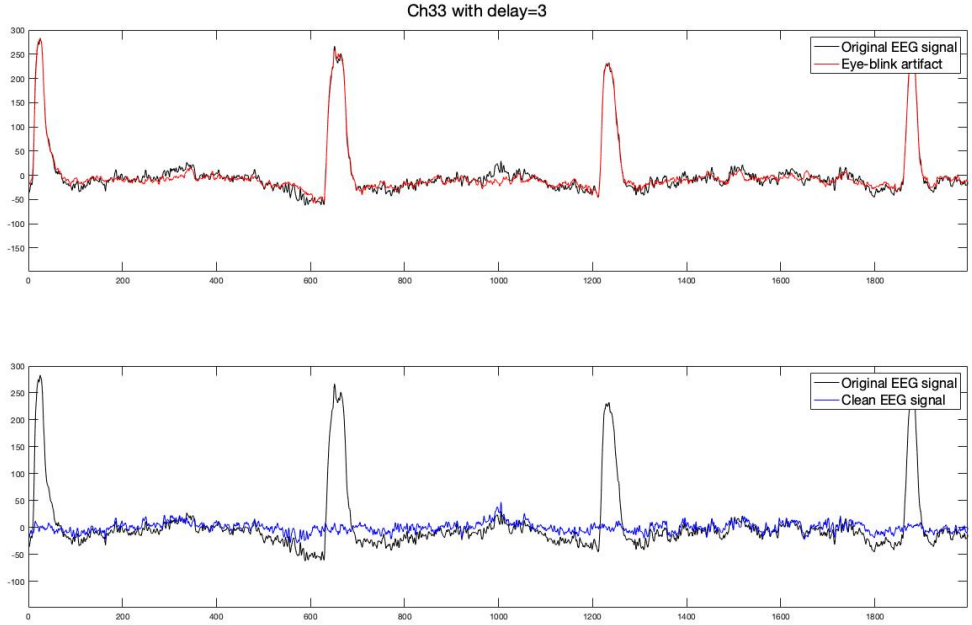

Figure 9: raw-EEG, estimated artifacts and MWF filtered EEG signal with delay=3

Do you observe a better artifact removal after item 3 compared to item 1? In the output variables of the mwf process, there are two performance metrics named SER and ARR. What do they represent? How can they be used to evaluate artifact removal? Display SER and ARR in the title of both plots generated in item 2 and item 3.

In the *mwf_process* function there are two output parameters:

- SER: Signal to Error Ratio. It measures the clean EEG distortion. It is calculated separately in the *mwf_performace* function. It uses the clean EEG points marked in the mask interface and the clean EEG points calculated with the designed filter and gives us the quality of the clean EEG data. The larger the SER parameter is, the cleaner EEG signal we have reached.

- ARR: Artifact to Residue Ratio. It measures artifact estimation. It is very similar to the SER parameter but taking into account the artifact estimation and the artifact points marked in the mask interface. As the SER parameter, the ARR represents better results when it becomes bigger.

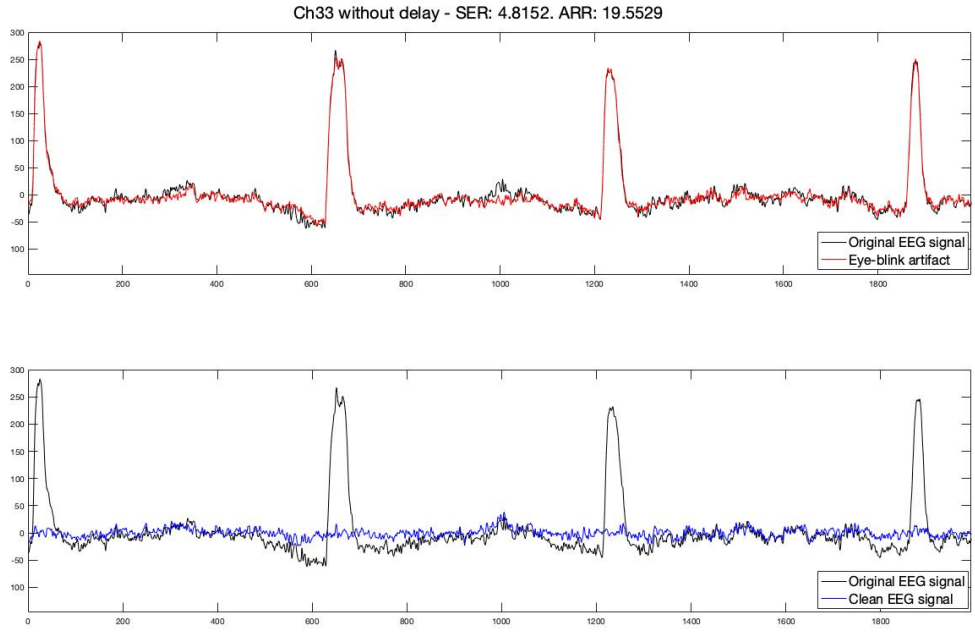

Figure 10: raw-EEG, estimated artifacts and MWF filtered EEG signal

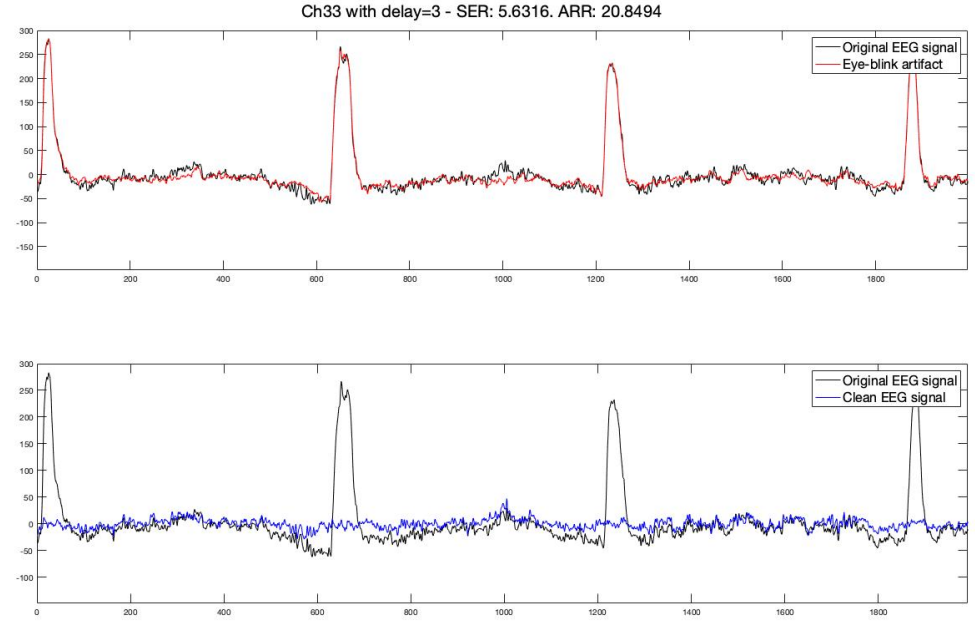

Figure 11: raw-EEG, estimated artifacts and MWF filtered EEG signal with delay=3

As we have observed in Figure 10 and in Figure 11, the SER and ARR parameters for both estimations are larger in the delay case, meaning that the EEG signal we obtained is better than the first one without delay.

	Delay = 0	Delay = 3
SER	4.8152	5.6316
ARR	19.5529	20.8494

Table 1: SER and ARR parameters

Note the message that is displayed while the MWF is applied. What does it signify? Carry out item 1, retaining only the first few generalized eigenvalues of the estimated artifact covariance matrix. (Hint: Note the parameter p in the code of mwf compute.m). What would be an ideal number of eigenvalues to be retained for the estimated artifact covariance matrix for this problem? Justify your answer.

When we run the code, we receive the following message: "*Rejecting 39 negative eigenvalues (total: 64) ... Rank = 25*".

It means that, after executing the GEVD on the covariance matrices of the artifact and the clean EEG, calculating the joint diagonalization and the difference between these two diagonalized results, there are 39 negative values in the resulting matrix. Thus, we are taking a rank = 25.

The ideal number of eigenvalues to retain would be between 2 and 6 (the smaller the better) because, as we can see in the following figure, when it reaches de second eigenvalue it becomes more and more constant. The second value represents the elbow of the graph, where we use to get the best results.

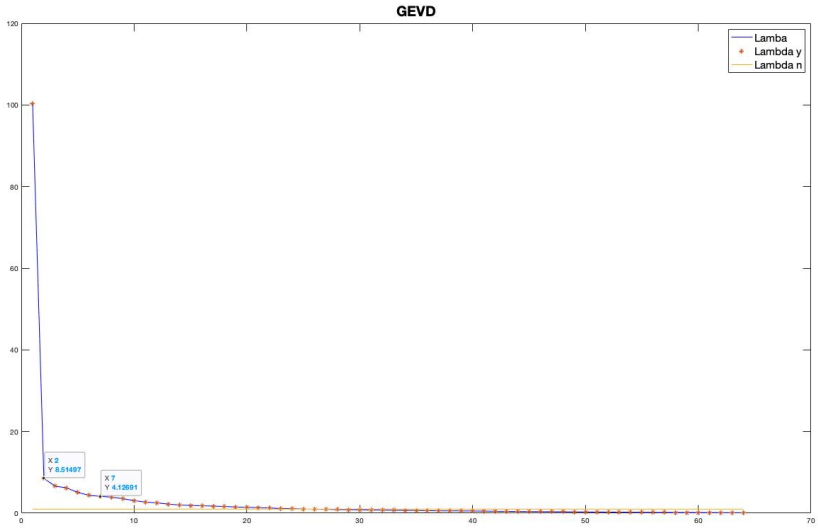

Figure 12: Generalized Eigenvalues

After applying the MWF obtained with the two first eigenvalues, we get the following results on the EEG signal.

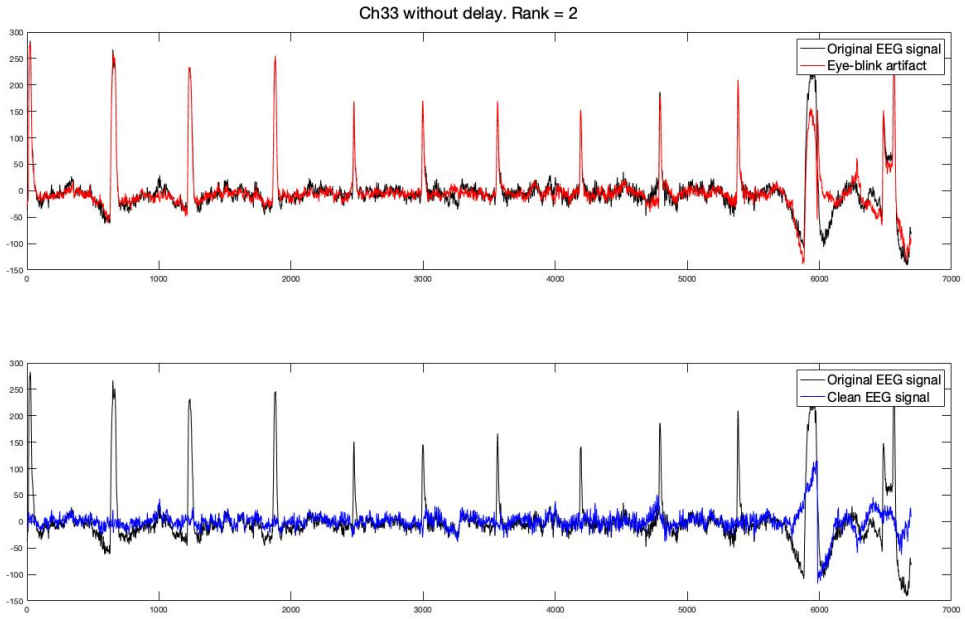

Figure 13: raw-EEG, estimated artifacts and MWF filtered EEG signal.

Now, we will consider muscle artifacts. Redo item 3, but now using a mask for all muscle artifacts. Ignore eye blink artifacts and also the signal duration restriction while creating this mask. Store the new mask in a different variable to the one used in task 2. Illustrate muscle artifact removal by plotting a segment of a channel with artifacts and after artifact removal. The plots should display SER and ARR values like the earlier plots. Comment on the results.

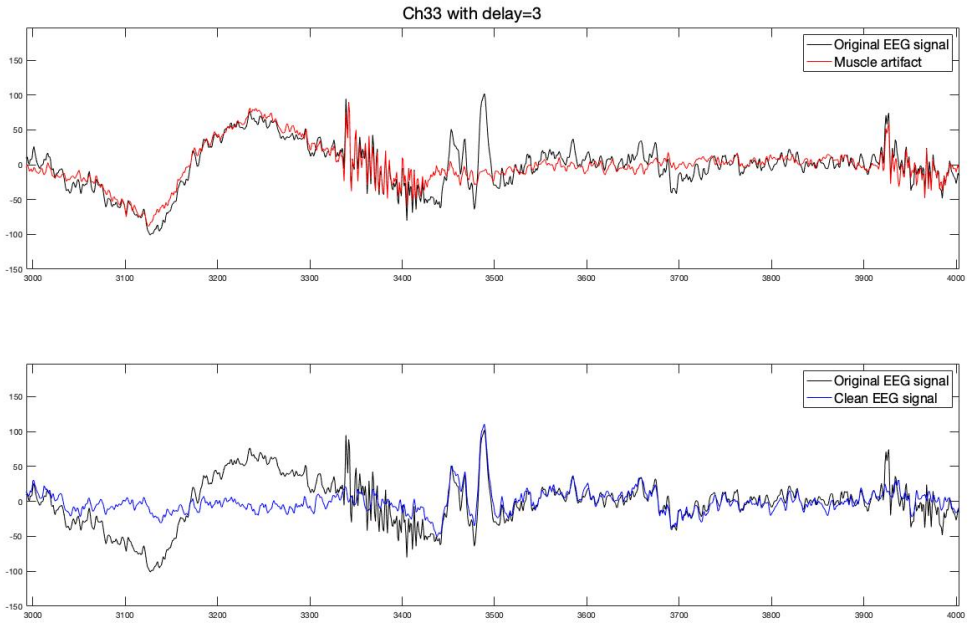

Figure 14: raw-EEG, estimated muscle artifacts and MWF filtered EEG signal

We can observe how this muscle artifact is worst removed than the blink-eye one. It is because this kind of artifact is not a clearly separated from the clean EEG data as the blink-eye one, so selecting the artifact intervals when constructing the supervised mask is more complicated. Thus, the program is using a worst mask to compute the artifact removal.

Compute SER and ARR of the muscle artifact removal using CCA carried out in item 4. Which algorithm performs better here (CCA or MWF)? Can you explain why?

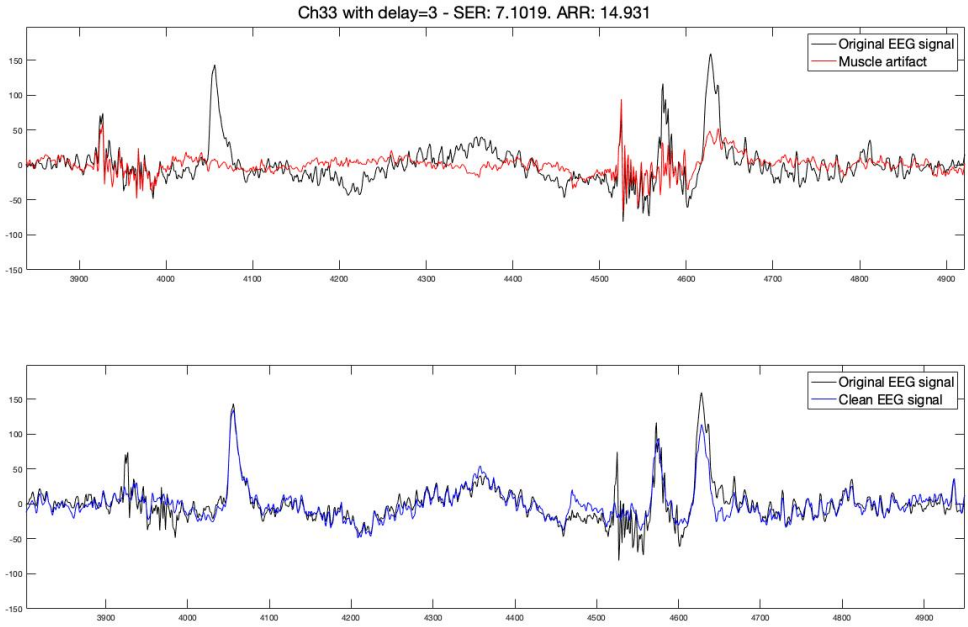

Figure 15: raw-EEG, estimated muscle artifacts and MWF filtered EEG signal

2. Threshold-based spike sorting in neural probes

2.1. Filter design in threshold-based spike sorting

Prior to the filter design, a template of the target neuron waveform is estimated based on a prior clustering step. This part is done for you, and you can find this spatio-temporal template in the variable template. Visualize this spatio-temporal template of the target neuron spike using the plot MultiChannel function. Read the help documentation of this function in plotMultiChannel.m.

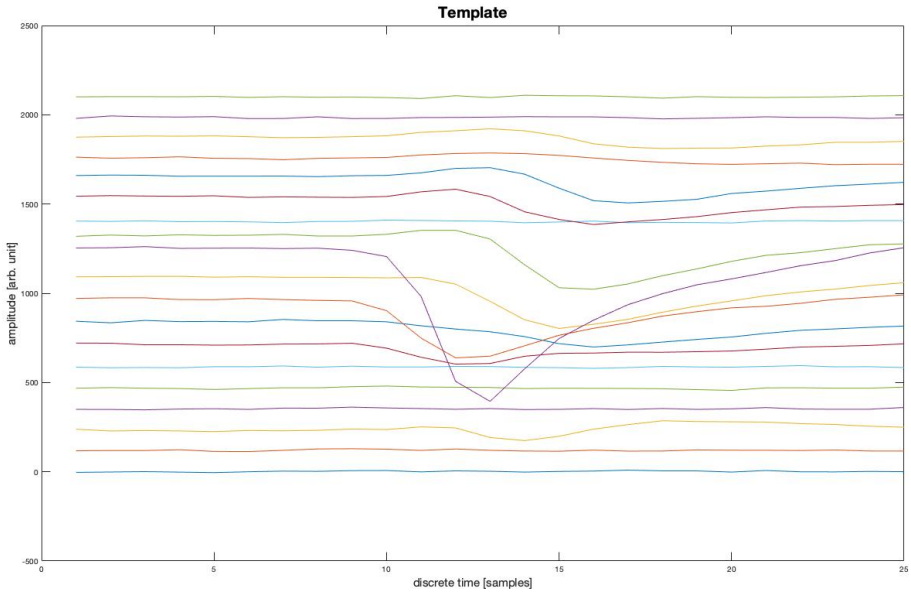

Figure 16: Template of the target neuron waveform

In Figure 16 we can see the target neuron waveform. In the x axis we have the time variable and in the y axis we can observe the 19 different channels where the neuron signal is picked up.

First, we use the template as a filter. Apply the template estimated in the previous step as a 'naive' matched filter on the training data. Complete applyMultiChannelFilter function to perform this. This function will be re-used when applying filters on data in later tasks.

The filtered training data is used further to select a threshold for threshold-based spike sorting. In the script, you can find code which performs a sweep over a number of values as potential thresholds. For each of these values, sensitivity and precision of the spike sorting on the training data is calculated for you. Plot a precision-recall (P-R) curve using these values. Comment on the P-R curve. Observing the P-R curve, select a good threshold for spike sorting, indicate it on the plot and comment.

In order to analyze the P-R curve we will consider the true positive values (TP) those where the neuronal spike was detected, and, according to this, the spikes that were not detected, false negatives (FN) and the plain signal marked as spikes, false positives (FP). The P-R curve presents the precision and the recall or sensitivity of the filter. These two parameters are:

- **Precision:** this parameter measures how many values that were marked as positive were actually true positives; i.e., how many peaks marked as spikes are actually neuronal spikes.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall or sensitivity:** it measures how many of the positive values were detected; it is, how spikes were marked.

$$Recall = \frac{TP}{TP + FN}$$

We have a trade-off between the recall and the precision, as one of them reaches better results, the other becomes worst. In order to look for the best set of parameters we have use the F-score measurement. It calculates one value per set of recall and precision so we can look for the maximum. This value corresponds to the optimal threshold value. We can see in the following picture this representation:

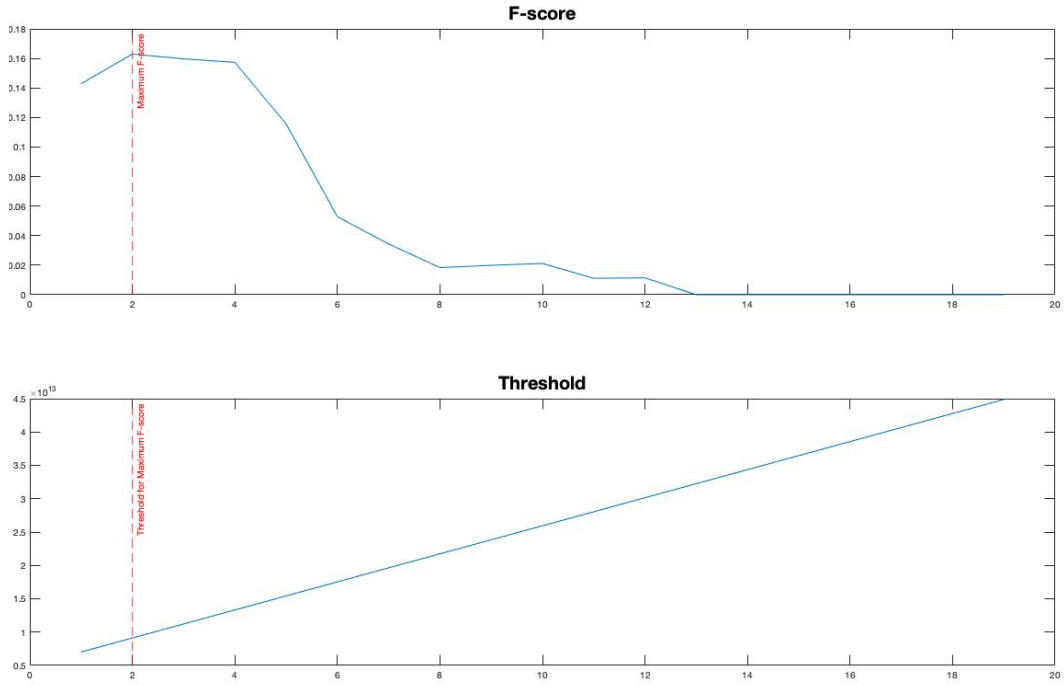

Figure 17: F-score and threshold for template filter

Our optimal threshold will be $9.12 * 10^{12}$.

We now have your template filter and threshold to validate threshold-based spike sorting on the test data. Again, the code for the same is implemented in the script in the section titled "validate template filter on testing data." Comment on the results that are printed in the command window at the end of this section. Comment also on the filter output plot, which the script plots along with this.

If we take a lower threshold, the number of TP will increase along with the number of FP, so the precision will decrease while the recall goes up. On the other hand, if we take a higher threshold, many TPs will be lost, but we will also decrease the number of FP, so the precision will increase while the recall decreases. It is a continuous trade-off between not detecting many true spikes and detecting peaks that are not really neuron spikes.

With our chosen threshold we get, in the command window, this message: *"template-filter: for your threshold: recall: 0.721, precision: 0.051"*. This means that we have detected most of the spikes (44 of 61) in exchange of having a lot of false positive values; that's why we get a very low precision.

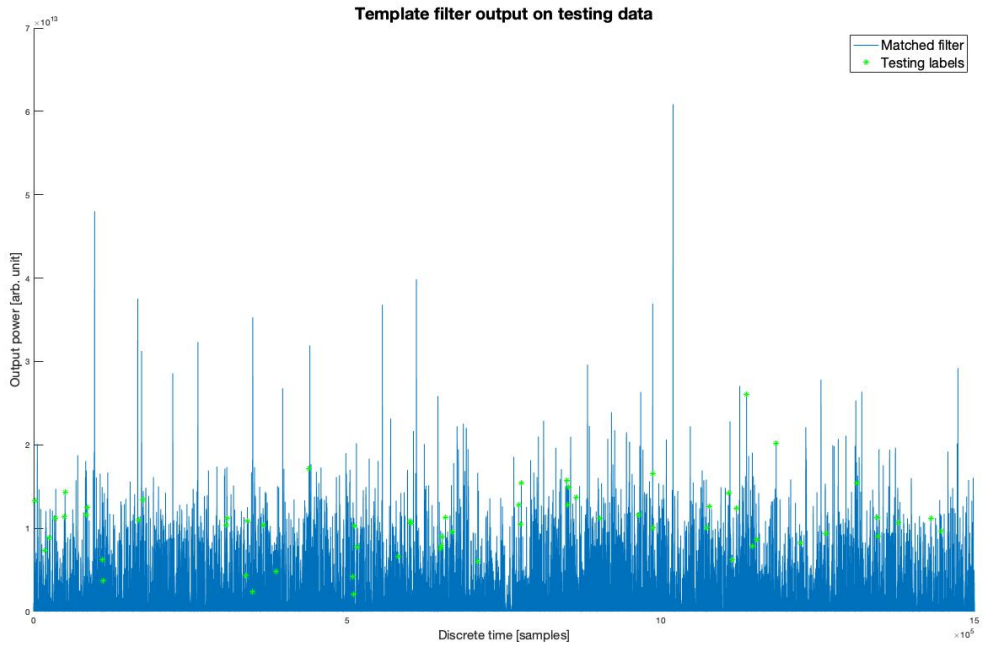

Figure 18: Template filter output on testing data

In this plot we see, in blue, the template filter output obtained with the testing data after choosing one threshold with the training data. The green dots mark the testing labels, the dots in the output

*Next, we design an SNR-optimal matched filter. For this filter design, the (lagged) noise covariance has to be estimated. Time instants corresponding to the noise segments in which the target neuron is not active in the data are provided in the noiseSegments variable. Write a function which accepts these noise segments and training data as inputs to estimate the lagged noise covariance matrix (we assume a causal filter here). Use an equal number of lags as the window length of the spike template (**why?**). Analyze the obtained matrix. Is the noise spatially white? Is the noise temporally white? Please comment.*

The number of lags must be equal to the window length of the template, in this case L=25, so we can pick a larger number of samples of the template to exploit in the correlation; thus, it will work better.

When the noise is **spatially white**, we have several channels that are seen as a sequence of uncorrelated variables, thus, in the color map representation, we would observe that one channel is only related with itself, resulting in a diagonal matrix regarding the different channels (independently of the content of each channel matrix). In this case, we do not have spatially white noise because we can see how one channel is not only related with itself, but also with other channels, even though this relation becomes lower when the channels are more separated. In order to observe this noise characteristic, we can look at the following image in a global way.

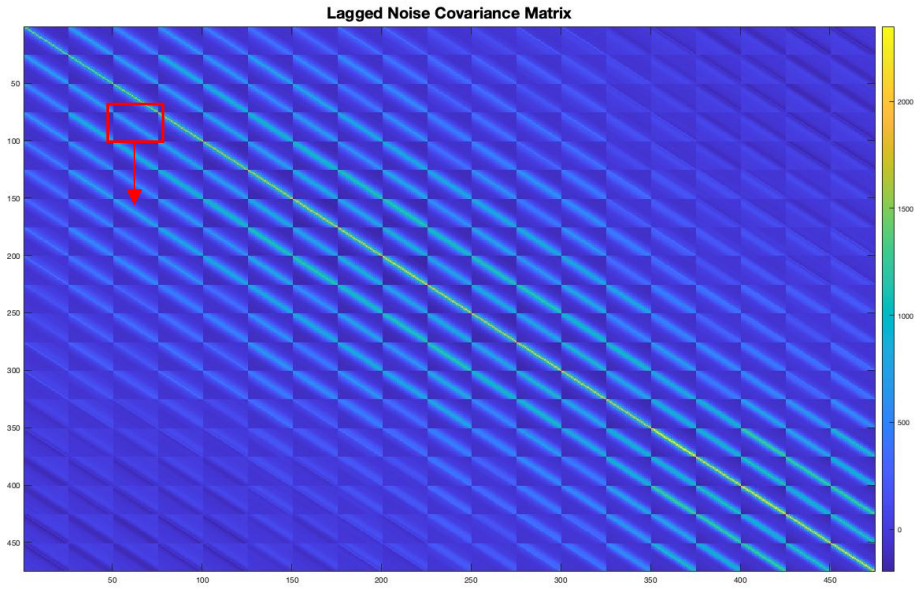

Figure 19: Lagged covariance matrix

This noise is not purely **temporally white**; if it was so the covariance matrix would only have non-zero values in the diagonal. However, in the below colormap representation we can observe that, despite it is quite similar, it does not only have positive values in the diagonal. In order to observe this noise characteristic, we should look at each of the "cells", each of them corresponding to the correlation between two channels.

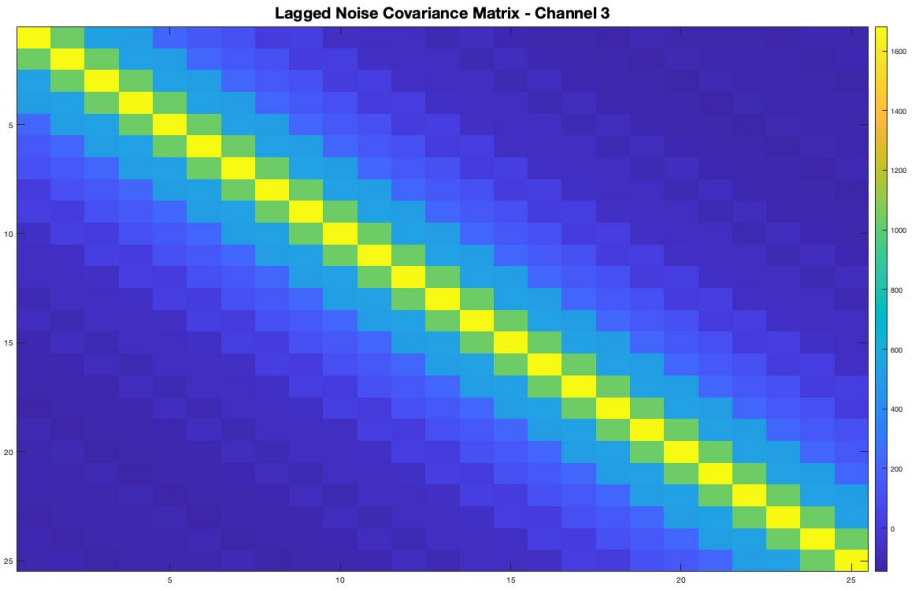

Figure 20: channel of the noise covariance matrix

The noise covariance matrix estimated in the previous step is regularized for the design of the matched filter for the target neuron. A function called regularizeCov generates this regularized noise covariance; stored in noiseCov matrix. Use noiseCov to design a matched filter for the target neuron.

As we demonstrated during the lectures, the matched filter obtained with the Lagrangian Multiplier method follows the following expression:

$$\hat{w} = {R_{nn}}^{-1} * \tau$$

We apply the matched filter you designed in the previous task on the training data using applyMultiChannelFilter completed in item 2.

The filtered training data is used further to select a threshold similar to item 3 for threshold-based spike sorting. Repeat the tasks of item 3, but now using matched-filter on training data.

Using the matched filter, *maxSNR*, we obtain the next P-R curve and F-score with its corresponding threshold:

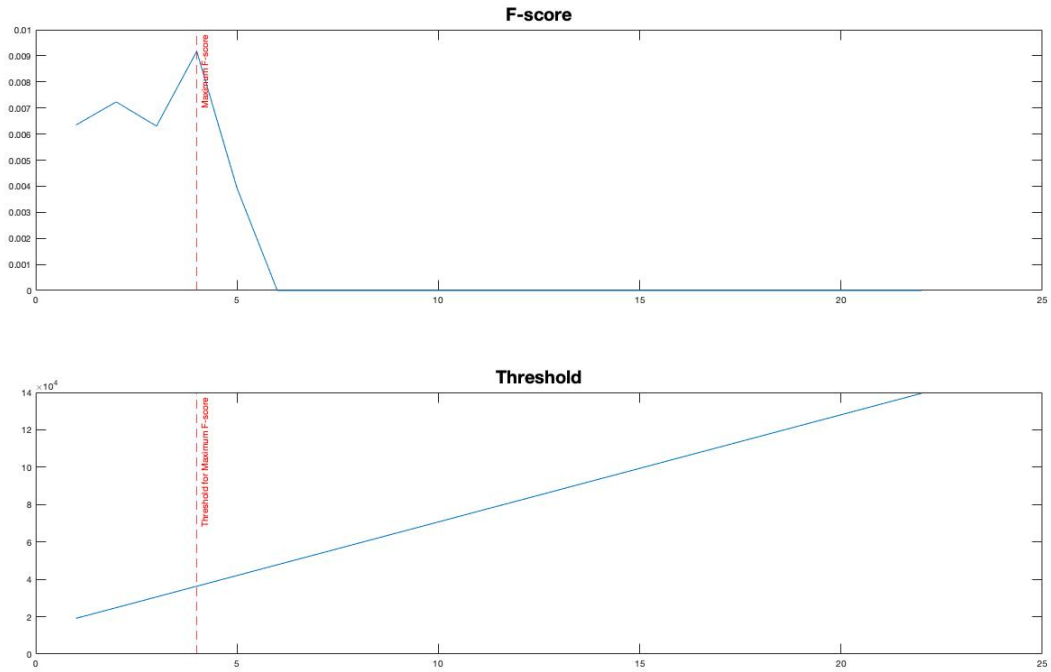

Figure 21: F-score and threshold for the matched-filter

Our optimal threshold will be 3.6316e+04.

We now have your matched filter and threshold to validate threshold-based spike sorting on the test data. Again, the code for the same is implemented in the script in the section titled VALIDATE MATCHED FILTER ON TESTING DATA. Comment on the results that are printed in the command window at the end of this section. Is it better or worse than item 4? Why? Comment also on the filter output plot, which the script plots along with this.

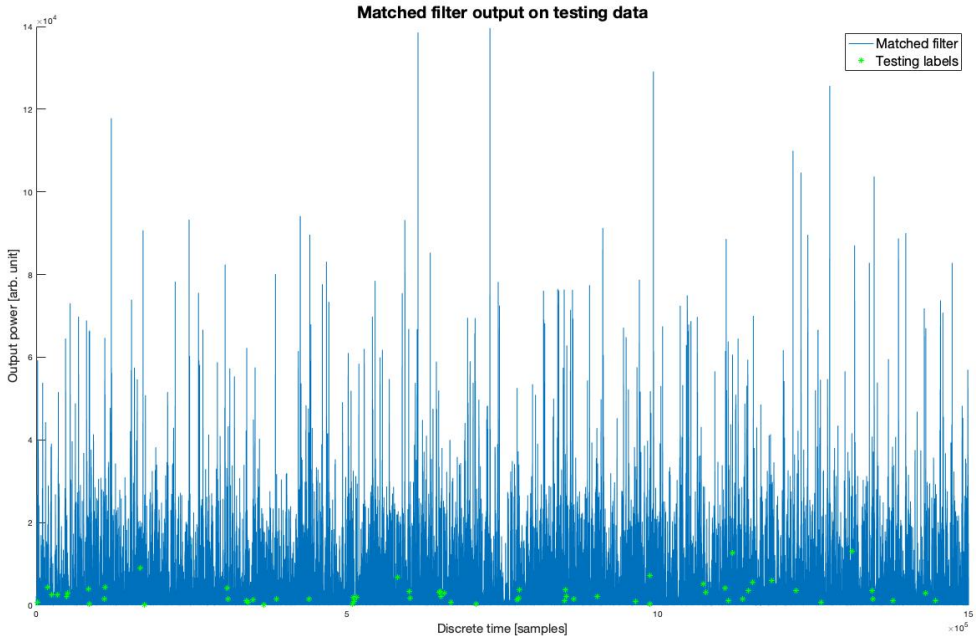

Figure 22: Matched-filter output on testing data

With our chosen threshold we get, in the command window, this message: *"matched-filter: for your threshold: recall: 0.049, precision: 0.009"*. These values are not optimal values, thus, we can be sure that we have made mistakes during the process.

Design a SPIR-optimal (max-SPIR) filter for the target neuron similar to the earlier matched filter design. Calculate the required covariance matrix using the function created in item 2. Note that the function should now estimate the peak-interference covariance matrix.

As we demonstrated during the lectures, the max-SPIR filter can obtained with the following expression:

$$\hat{w} = {R_{ii}}^{-1} * \tau$$

Where the Rii is obtained using the *calcCovMatrix.m* designed in previous tasks.

Similar to item 8, but using your max-SPIR filter presently, a threshold is to be selected. The script contains code which does a sweep of potential threshold values, calculating sensitivity and precision of spike sorting using corresponding thresholds. Within the same plot as item 8 (and in a different colour), plot the precisionrecall (P-R) curve on the training data for the threshold sweep on max-SPIR filtered data. Comment on its difference with respect to the matched filter's P-R curve, plotted in item. Select a threshold based on the plot.

Using the max-SPIR filter, *maxSPIR*, we obtain the next P-R curve and F-score with its corresponding threshold:

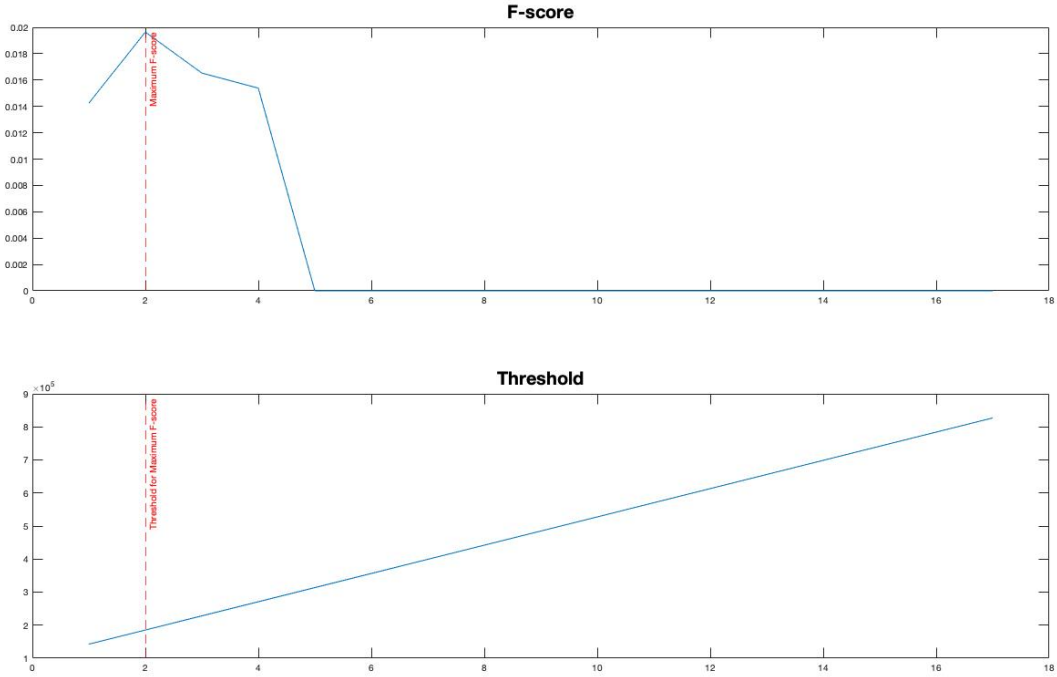

Figure 23: F-score and threshold for the max-SPIR filter

Our optimal threshold will be $1.85 * 10^5$.

Validation of a threshold-based spike sorting, now using the max-SPIR filter and the threshold you selected in the previous step, is carried out by the script next. Comment on the new results that are displayed within the command window. Is it different from the results obtained in item 9? Why?

With our chosen threshold we get, in the command window, this message: *"max-SPIR: for the maximum F1-score threshold: recall: 0.066, precision: 0.006"*. Just as in the previous filter, we are aware that these are not optimal results and, as it is a similar situation as in the matched filter, we can say that the error has been made in the design of the calcCovMatrix.m function used to calculate the noise covariance matrix (matched-filter) and the interference covariance matrix (max-SPIR filter). Even if we change the value of the threshold we obtain low values for the recall and precision parameters, which confirms that the error is placed on the covariance matrix calculation.

The script plots both the filter outputs (after normalization). The ground truth spike times are marked on the plot. Answer the following questions:

- *From the generated plot, describe the qualitative difference between both filter outputs. Zoom in on relevant segments to better demonstrate your point.*
- *Based on your qualitative findings, which filter output do you think is more suitable for threshold-based spike sorting?*
- *Relate your findings about the filter output differences to their respective filter design objective functions.*

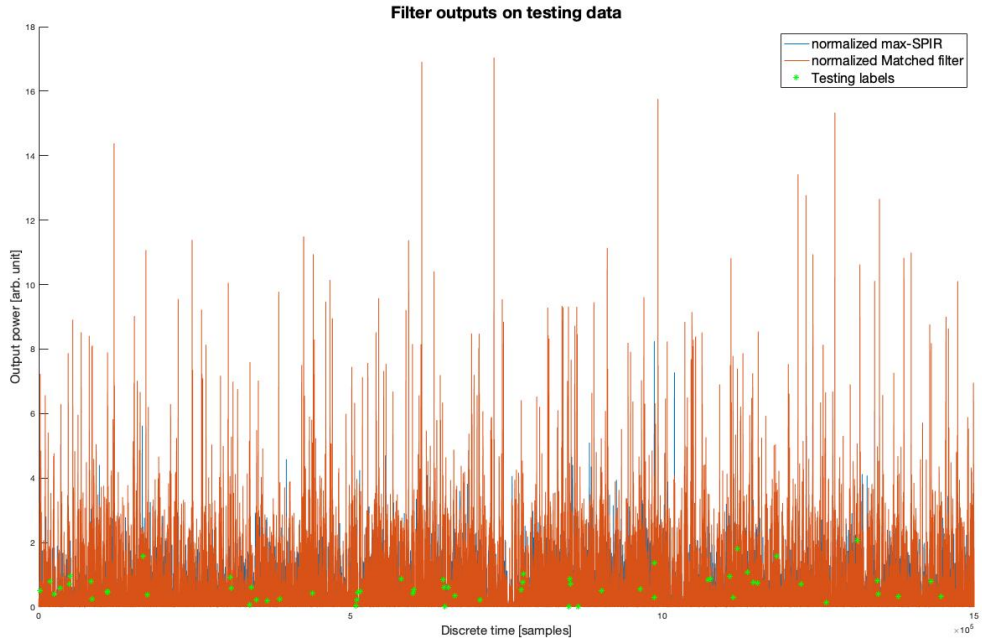

Figure 24: Filter output on testing data

2.2. Stimulation artifact removal to improve threshold-based spike sorting

Let us first visualize the data. Plot a representative segment of a single channel of data around a neuronal spike. Illustrate the position of the spike in the plot. Use the indices of neuronal spike in labels. Are you able to observe the spike? Why or why not? Hint: You are already familiar with the nature of a spike waveform from section 2.1.

First, we've represented some part of the data containing the neural spike located in 73836, which is marked with a red line. As we can observe, it is not visible the difference between this neural spike and other point when we are not supposed to observe anything. This is caused by some other artifacts that cover all the spike so we cannot see it. We have represented the following picture using the function *plotMultiChannel.m* provided in the previous section of the assignment.

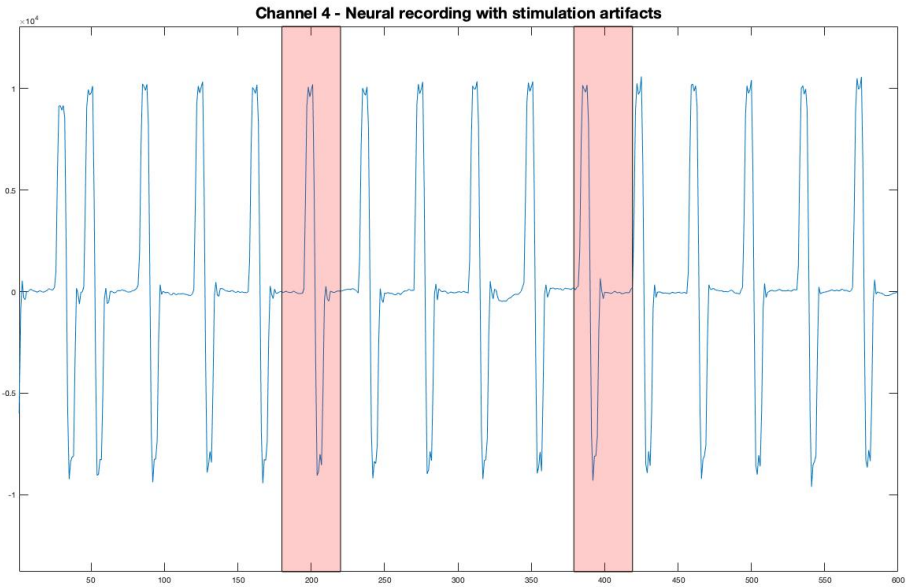

Figure 25: Neural recording with artifact

We will use linear regression to estimate stimulation artifacts in each channel using artifact as observed in non-adjacent channels. To this end, what is the solution of eq. (5)?

The optimal filter for this artifact removal is the solution of the next equation:

$$\widehat{w_k} = \min_{w_k} \|x_k - X_{-k} w_k\|_2^2$$

We end up solving a Least Squares problem with the following solution:

$$\widehat{w}_k = (X_{-k}^T X_{-k})^{-1} X_{-k}^T x_k$$

Now, estimate the spatio-temporal filter $\widehat{wk}$ for each channel-k. You might have to use regularization while constructing this filter. Note: The spatially adjacent channels for each channel-k are available in the variable adjacent channels. The artifact event time indices are available in the variable events.

In order to solve the LS problem with the MATLAB code y=A\b we first need to compute matrix A and vector b for each of the k channels.

- The A matrix corresponds to the X_{-k} mentioned before. It is a $t_M \times (N_k * L)$ matrix and has been formed making a distinction between those time intervals where the artifact was active and those where it was not.
- The b vector is the x_{-k} vector that contains the data for the no-adjacent channels of k.

Finally, we got one filter vector, w_k, for each of the channels.

Estimate the artifact signal for each channel k.

As it is mentioned in the assignment PDF, the signal after removing the artifact is the following one:

$$y(t) = x(t) - \widehat{w}_k{}^T x_{-k}(t) \rightarrow \forall t \in \mathcal{A}$$
$$y(t) = x(t) \qquad\qquad \rightarrow \forall t \notin \mathcal{A}$$

Thus, the artifact itself is calculated as $\widehat{w}_k{}^T x_{-k}(t)$. We computed it for every channel and, in concrete for the 4th channel, we got the following result. It is observed how similar this plot is with the one in the first task; we can know understand why we could not observe any spike on it, as they were masked by the artifact.

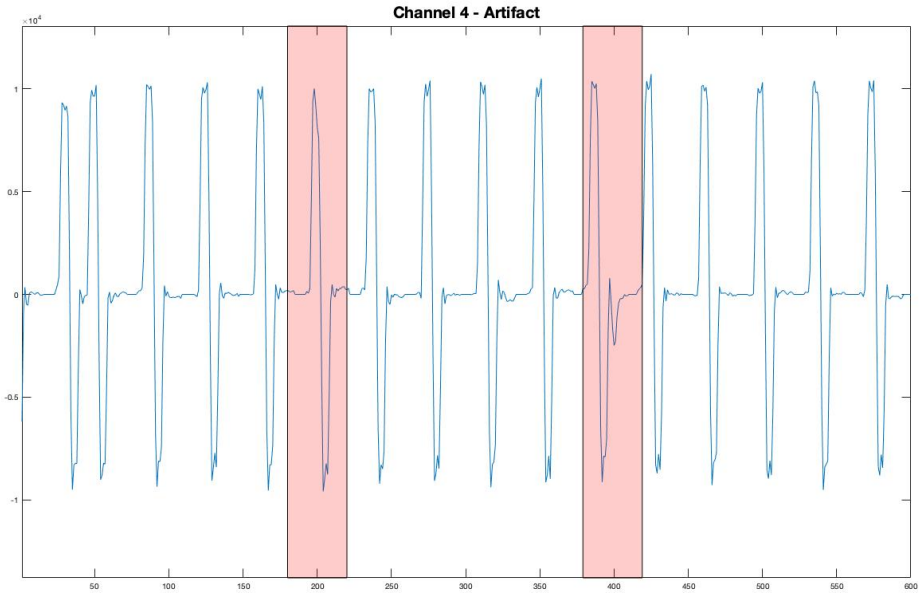

Figure 26: Artifact in channel 4

Remove the estimated artifacts from the corresponding channels. Now, plot the same section of data as plotted in item 1, after artifact removal. Can you better observe the neuronal spike?

In this last image we can see how the stimulation artifact was perfectly removed; now we can observe the first two neural spikes located in the red area, just as in the first task. Thus, we can confirm that the filter implemented in this section has correctly removed the artifact present in the signal.

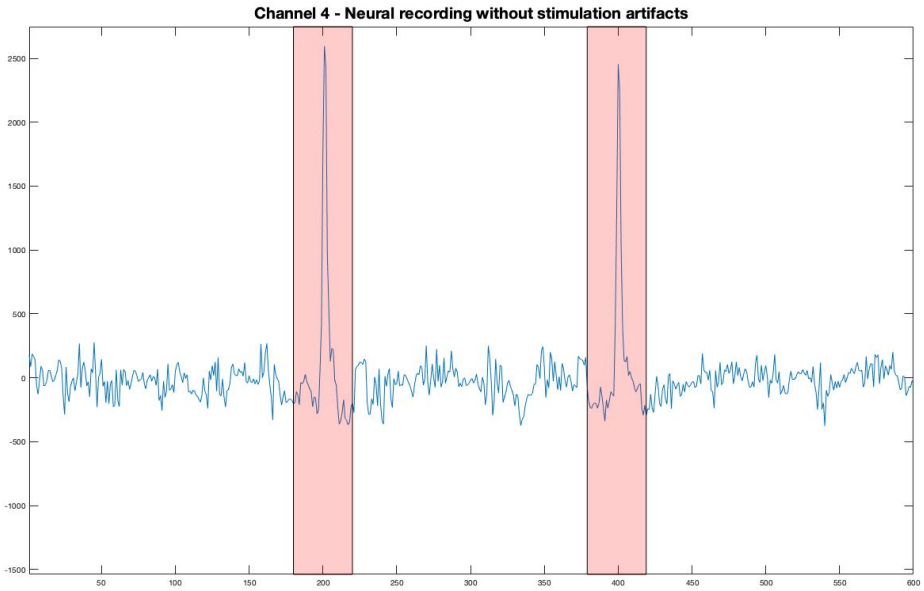

Figure 27: Clean signal after artifact removal