Selected Topics in Biomedical Signal Processing: Data-driven filter design for biomedical sensor arrays

General Instructions The report of this exercise should be divided into 2 sections with numbering and titles of sections as in this document. The PDF of the report and the MATLAB scripts are submitted through Toledo. Also provide a printed version of the report. Name your scripts as described in in each exercise's **What to submit** sub-section.

1 EEG artifact removal

Electroencephalography (EEG) measures electrical potentials generated by the brain using electrodes placed on the scalp. However, EEG is highly susceptible to artifacts generated by either technical factors, such as electrode movement, or by biological factors, such as eye blinks, muscle activity, etc. To reduce the impact of these artifacts on EEG signal quality, the signals are typically post-processed to remove the artifacts. In this exercise, we will use two different approaches to remove artifacts, 'unsupervised' and 'semi-supervised' methods.

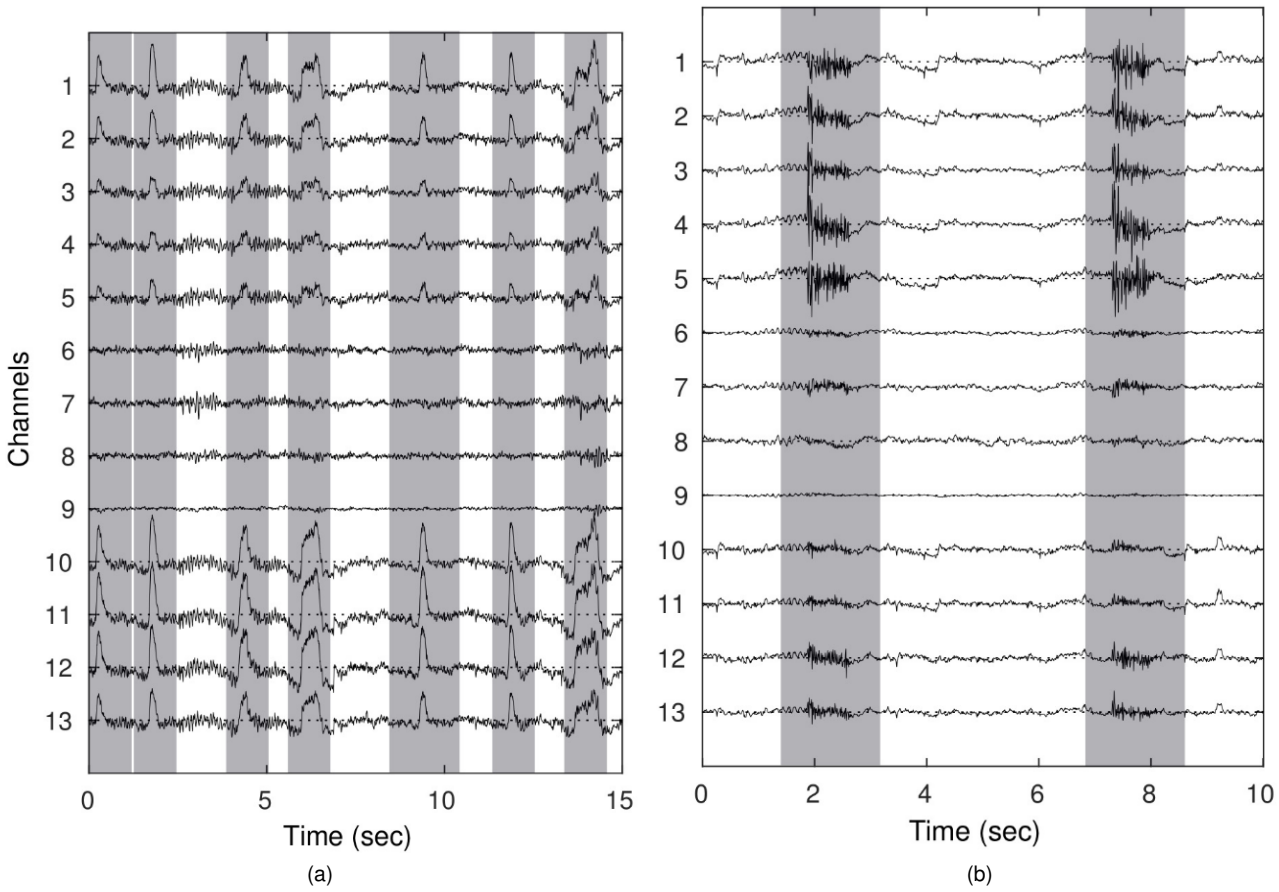

Figure 1: Examples of EEG contaminated by: (a) Eye-blink artifacts (b) Muscle artifacts

In this exercise, we will address the removal of two types of EEG artifacts, eye blink artifacts and muscle artifacts. Eye blink artifacts are one of the most commonly observed artifacts in EEG. In 1a, an example EEG signal contaminated with eye blink artifacts is shown. These artifacts are of high amplitude compared to background EEG and easily noticeable within the signal. In 1b, an example of a muscle artifact in EEG is shown. These are high frequency artifacts which occur in bursts due to contraction of cheek or jaw muscles.

1.1 Source files

- In Toledo, you can find the *MWF_artifact.zip* file which contains the following:
  - *eegdata_artifacts.mat* containing:
    1. **eegdata**: 2 minutes of 64-channel EEG data containing both eye blink and muscle artifacts.
    2. **fs**: Sampling frequency of the EEG data.
    3. **channelnames**: Names of EEG channels corresponding to the channel number (row of **eegdata**)
  - An *eegplot_simple.m* function for EEG visualization.
  - **channel_layout.jpg**: The layout of the EEG channels on the scalp.
- The latest EEGLAB toolbox is required for this exercise. EEGLAB is an interactive Matlab toolbox for processing continuous and event-related EEG. Instructions to install EEGLAB toolbox:
  1. The toolbox can be downloaded from: `https://sccn.ucsd.edu/eeglab/download.php` (or Google: 'EEGLAB').
  2. Extract the downloaded *zip* file into your MATLAB home folder.
  3. Add the path of unzipped EEGLAB folder to MATLAB's path. Use the 'Add Folder' option after clicking 'Set Path' of MATLAB. Do not use the 'Add with subfolders' option.
  4. Run the command **eeglab** from the MATLAB command-line. This will automatically add necessary functions of EEGLAB to the MATLAB path.
- The latest MWF based artifact removal toolbox is required for this exercise.
  1. The toolbox can be downloaded from: `https://github.com/exporl/mwf-artifact-removal`.
  2. Extract the downloaded *zip* file into your MATLAB home folder.
  3. Add the *mwf* folder present in the zip to your path.

1.2 Unsupervised artifact removal using CCA

In this section we will use canonical correlation analysis (CCA) to remove artifacts from EEG data.

1.2.1 Tasks

1. 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.
2. 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?
3. 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).
4. 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.
5. Could we also remove eye-blink artifacts from EEG using CCA? If so, describe the strategy to achieve eye-blink artifact removal.

1.3 Supervised artifact removal using MWF

In section 1.2, you have used an 'unsupervised' artifact removal method using CCA to remove artifacts. In this task, we will use the multi-channel Wiener filter (MWF) instead.

The method is semi-automatic and semi-supervised, as it requires the user to mark a few examples of artifact segments in the time domain EEG data. The algorithm will use these examples to train an MWF that is then applied to the full EEG recording to automatically clean the EEG signals by filtering out the artifacts in each channel. We will use the MWF-based EEG artifact removal toolbox for removing the artifacts from the EEG signal.

1.3.1 Tasks

1. 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*.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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?

8. Take a union of the masks generated in item 1 and item 6. Make sure that possible artifact segments are not indicated as clean segments in this union. Consult the toolbox manual on what each mask sample signify. Use this mask to generate an MWF filter and filter the EEG. Use the parameters of item 3. Which artifacts are removed now? Provide illustrations to support your answer.

1.4 What to submit

The following items are expected to be present in the report. Any additional details, figures to support or illustrate your arguments can be added.

1. All the plots generated during the execution of the tasks in section 1.2 and section 1.3. Provide enough information in captions to figures to identify each plot.

2. Interpretations of the plots and descriptive answers to the questions in section 1.2 and section 1.3.

MATLAB code to be submitted separately:

1. A script named *ex_1_artifact_removal.m* with code which was used to run the tasks of section 1.2 and section 1.3.

2 Threshold-based spike sorting in neural probes

The spiking activity of neurons can be measured and recorded using extracellular electrodes placed in close proximity to those neurons. The spikes measured on a certain electrode typically originate from multiple neurons, and the spikes from a specific neuron are often picked up by multiple electrodes. Resolving such a multi-electrode recording into the spike times of its individual contributing neurons is known as spike sorting. Threshold-based spike sorting is the process of assigning each detected spike in an extracellular recording to its putative neuron, using only linear filters and simple thresholding operations. Due to the computational simplicity of performing threshold-based spike sorting, it is an interesting technique for real-time spike sorting applications. One possible application is a single-unit activity controlled brain-machine interface (BMI).

In section 2.1 you will first explore the filter design in threshold-based spike sorting. You will design both a matched filter and max-SPIR filter and compare their performances in threshold-based spike sorting. In section 2.2, we will address the issue of artifacts present in extracellular recordings which affect spike sorting and how to remove them.

2.1 Filter design in threshold-based spike sorting

2.1.1 Source files

- Training and test data in two mat files:
  1. *trainingData.mat* (Duration: 120s)
  2. *testingData.mat* (Duration: 60s).

  Each mat files contain:
  - A multi-channel neural recording, from a neural probe, originally of 3 min duration which has been split into training and test data. The data is sampled at a frequency $f_s = 25$kHz and high-pass filtered with a cut-off frequency of 300Hz. Only a local spatial region of the probe is given (19-channels) which captures most of the signal energy of the target neuron.
  - Ground truth spike times for the neuron of interest for validation of test data.
- The primary script for the exercise called *spike_sorting.m*. The script contains code required to perform tasks in section 2.1.2 along with place holders to insert your own code to perform tasks. Comments are added in the script for better clarity.
- Other important functions used in the exercise are given below. In section 2.1.2, more details on how some of the following functions are to be completed is provided:
  - applyMultiChannelFilter.m : Performs the filtering of neural data. *To be completed yourself.*
  - findInterferenceSegments.m : Finds and returns peak interferer segments using an interference threshold and target safe zone.
  - findNoiseSegments.m : Finds and returns noise segments.
  - mat2stacked.m : Converts a matrix to a stacked vector.
  - plotMultiChannel.m : Visualizes multi-channel neural data.
  - stacked2mat.m : Converts stacked vectors to matrix form.

2.1.2 Tasks

The tasks of this exercise are carried out by individual sections of the script, *spike_sorting.m*. You are instructed to complete portions of the script or functions called by the script at various points. Proceed sequentially through the script, since results of each task is needed for the succeeding task. Consider the use of the 'Run Section' option in MATLAB for running a specific subset of the code.

1. 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 *plotMultiChannel* function. Read the help documentation of this function in *plotMultiChannel.m*.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

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

8. 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.

9. 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.

10. 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.

11. 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 precision-recall (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 8. Select a threshold based on the plot.

12. 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?

13. 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.

2.1.3 What to submit

The following items are expected to be present in the report. Any additional details, figures to support or illustrate your arguments can be added:

1. All the plots generated during the tasks.

2. Descriptive answers to the questions asked in section 2.1.2.

MATLAB code to be submitted separately:

1. The script titled *spike_sorting.m* with code required to implement the tasks in section 2.1.2.

2. The following 2 functions:

    - Function to estimate lagged covariance matrices.
    - *applyMultiChannelFilter.m* (completed)

2.2 Stimulation artifact removal to improve threshold-based spike sorting

To study the effect of neural stimulations on cortical neurons, extracellular recordings are carried out along with external stimulations. When stimuli are applied directly to the brain, the EEG recordings can contain artifacts of an order of magnitude larger than the neuronal spikes. Consequently, the spike sorting process fails on these recordings since artifacts overlapping with spikes dominate the signal. It is therefore necessary to first perform artifact removal on the recordings before spike sorting techniques can be applied.

In this section, we explore a stimulation artifact removal technique based on linear regression. As, the spike waveform is only observed in a few channels close to the firing neuron (contrarily to the artifact waveform which is present in all channels), we can estimate the artifacts in each channel using a spatio-temporal filter computed using the artifact signals from far-away channels. The EEG signal is then cleaned by subtracting the estimated artifact signal from the raw signals.

Assume we have an $N-$channel extracellular recording contaminated by stimulation artifacts. Let $\mathbf{x}_{-k}(t)$ contain the signal data recorded in $N_k < N$ channels excluding channel-k and its adjacent channels defined as:

$$\mathbf{x}_{-k}(t) = \begin{bmatrix} \underline{\mathbf{x}}_1(t) \\ \vdots \\ \underline{\mathbf{x}}_{N_k}(t) \end{bmatrix} \tag{1}$$

with

$$\underline{\mathbf{x}}_i(t) = \begin{bmatrix} x_i(t) \\ x_i(t-1) \\ \vdots \\ x_i(t-L+1) \end{bmatrix}, \tag{2}$$

the data in channel-i at $t \in \mathcal{A}$, the set of all time instances during artifacts, and its L sample delays. The objective in linear regression based artifact removal is to obtain the artifact removed signal $y_k(t)$ from $x_k(t)$, the signal at channel-k, using:

$$y_k(t) = x_k(t) - \hat{\mathbf{w}}_k^T \mathbf{x}_{-k}(t) \ \forall t \in \mathcal{A} \tag{3}$$

$$y_k(t) = x_k(t) \qquad t \notin \mathcal{A} \tag{4}$$

where $\hat{\mathbf{w}}_k$ is a spatio-temporal filter computed as the solution of the following optimization problem for each channel-k:

$$\hat{\mathbf{w}}_k = \arg\min_{\mathbf{w}_k} \frac{1}{2} ||\mathbf{x}_k - \mathbf{X}_{-k}\mathbf{w}_k||_2^2, \tag{5}$$

Here,

$$\mathbf{x}_k = [x_k(t_1), x_k(t_2), \ldots, x_k(t_M)]^T \tag{6}$$

and

$$\mathbf{X}_{-k} = \begin{bmatrix} \mathbf{x}_{-k}^T(t_1) \\ \mathbf{x}_{-k}^T(t_2) \\ \vdots \\ \mathbf{x}_{-k}^T(t_M) \end{bmatrix}. \tag{7}$$

with $\{t_m\}_{m=1}^M \in \mathcal{A}$.

2.2.1 Source Files

- *artifactData.mat* a mat file containing the following variables:
  - **data**: A multi-channel neural recording contaminated by stimulations artifacts
  - **events**: A vector of indices corresponding to time instants of stimulation artifacts
  - **fs**: Sampling frequency of the neural recording
  - **labels**: Ground truth time indices of the spikes of a neuron in the recording
  - **adjacent_channels**: A cell containing spatially adjacent channels corresponding to each channel in the recording

2.2.2 Tasks

1. 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.

2. 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)?

3. Now, estimate the spatio-temporal filter $\hat{\mathbf{w}}_k$ 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**.

4. Estimate the artifact signal for each channel k.

5. 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?

2.2.3 What to submit

The following items are expected to be present in the report. Any additional details, figures to support or illustrate your arguments can be added:

1. All the plots generated during the tasks.

2. Descriptive answers to the questions asked in section 2.2.2.

MATLAB code to be submitted separately:

1. A script titled *stimulation.m* with code required to implement the tasks in section 2.2.2.