# ELEC 472 - Artificial Intelligence: Lab 3

## Objectives

This laboratory activity is focused on data pre-processing.

## Submission and Deliverable

Please submit the following through onQ: ***one report*** (with ***13*** parts) – the report must be created in MS Word (or Latex) and converted to PDF prior to submission + one **Python file (.py)** containing all your work.

You have been given a python file (lab_3.py) which contains various functions used in this lab.

### 1. Reading and Plotting the Time-series

The goal of the first step is to read and visualize the data. Download the file named "ecg.csv" from onQ. The file contains an Electrocardiogram (ECG) signal, which records the electrical activity of the heart. If interested, you can learn more about ECG signals at:
https://www.heartandstroke.ca/heart-disease/tests/electrocardiogram
https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983
https://en.wikipedia.org/wiki/Electrocardiography

Now run the following code to visulaize the input:

```
import matplotlib.pyplot as plt

ecg_timeseries_full=[]
with open('ecg.csv') as f:
    for line in f.readlines():
        ecg_timeseries_full.append(float(line.strip()))

# from the hardware specs used for recording this data, we know that
# the sampling frequency is 500Hz
sampling_freq = 500

plt.figure()
plt.title('Entire ECG signal')
plt.plot(ecg_timeseries_full)
plt.show()

# the time-series is too long, so we're going to take a small segment
x = ecg_timeseries_full[71740:81060]

plt.figure()
plt.plot(range(len(x)), x)
plt.title('Cropped ECG signal')
plt.show()
```

***Report - Part 1:*** Take a screenshot of both plots and include them in the report.

***Report - Part 2:*** Now zoom on an individual heart beat, take a screenshot, and include it in the report. You can zoom in by either using the Zoom option on the plot depending on your IDE or by changing the range of x so that only a single beat is displayed.

## 2. Power Spectral Density (PSD) Analysis

***Periodogram Method:***
The goal of the first step is to perform PSD (periodogram) analysis. The *psd* function is in the given Python file. Now, run the following code and generate the plot:

```
freq1, pow1 = psd(x, sampling_freq)
# Remember, x is the same data created earlier, covering samples 71,740 to 81,060

plt.figure()
plt.plot(freq1,pow1)
plt.ylim([0, 0.005])
plt.title('Periodogram')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid()
plt.show()
```

Note that the 'ylim' command is limiting the y axis to allow better visualization since the first few points have very large values (you can investigate this by commenting out the 'plt.ylim' command). A good way to visualize very large signals is to take the log10 of the *y* values. Try the following code:

```
plt.figure()
plt.plot(freq1,10*np.log10(pow1))
plt.title('Periodogram Log scale')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid()
plt.show()
```

***Report - Part 3:*** Take screenshots of the Periodogram plots (both without and with the log10), and include them in the report.

***Welch Method:***
Now, let's use the Welch algorithm. You can find the function in the given Python file.

```
freq, pxx_welch = pwelch(x, fs=sampling_freq, window='hamming',)

plt.figure()
plt.plot(freq1,pow1)
plt.ylim([0, .007])
```

```
plt.title('Periodogram')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid()
plt.show()

plt.figure()
plt.plot(freq1,10*np.log10(pow1))
plt.title('Periodogram Log scale')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid()
plt.show()
```

***Report - Part 4:*** Take screenshots of the Welch plots (both without and with the log10), and include them in the report.

***Report - Part 5:*** In 2-3 sentences, mention your observation regarding the Periodogram vs. the Welch method. (hint: how do they compare in terms of smoothness/variance?)

***Report - Part 6:*** When we plotted the non-log versions of the PSDs, we had to limit the *y* axis using 'ylim'. When ylim is removed, we see a huge spike at the beginning of the PSDs. What do you think this spike corresponds to? (hint: If you're not sure about the answer to this question, the answer will become clear in the following sections; so continue with the rest of the lab and revisit this question later.)

## 3. Filter Design

***Report - Part 7a:*** To create FIR filters, there exist a function called **`firwin`** in the **`scipy.signal`** library which operates as follows:

```
coefficients = firwin(window_size, cutoff, window='hamming', pass_zero=True,
scale=True, fs=sampling_freq)
```

Here is a detailed list of how these input arguments work:
**window_size**:

- The number of filter coefficients (taps).
- Should be an **odd number** for symmetric filters (default behavior).

**cutoff**:

- Cutoff frequency/frequencies (normalized if fs is not specified, in Hz otherwise).
- For **lowpass/highpass** filters: a single value.
- For **bandpass/bandstop** filters: a list/array with two values.

**window** (default: 'hamming'):

- The window function used to shape the filter response.

- Examples: 'hamming', 'hann', 'blackman', 'kaiser', 'rectangular'.

**pass_zero**:

- Defines the type of filter:
  - True → **Low-pass** filter.
  - False → **High-pass** filter.
  - None → Band-pass or Band-stop filter (determined by cutoff length).

**scale**:

- If True, normalizes the filter to have unit gain at DC (0 Hz) for lowpass/highpass or center frequency for bandpass.

**fs**:

- Sampling frequency in Hz (defaults to 1.0 if not provided).
- If provided, cutoff should be in Hz.

* *If your version of Python or the library differs slightly from this implementation of the function (e.g., in terms of input-output arguments), please refer to the function's documentation for the correct details.*

Now, use this function to create a low-pass Hamming filter with a size of 401 coefficients and a cutoff frequency of 5 Hz. Recall that the sampling frequency of our ECG is 500 Hz. Name the coefficients of the filter **`filter_coefficients_low`**. Then, plot the created window, and provide the plot as well as a screenshot of your code in the report.

***Report - Part 7b:*** Repeat the process, but this time for a high-pass filter. Name the coefficients of the filter **`filter_coefficients_high`**.

***Report - Part 8:*** Now, we can convolve the filter with the signal using the following code

```
x_filtered = np.convolve(x, filter_coefficients)
```

But before filtering the ECG signal, remember when we plotted the PSD without the 'ylim' command. The huge spike at the beginning was because of the very strong *bias/offset* in the data. To address this, apply one of the filters we designed earlier (**`filter_coefficients_low`** or **`filter_coefficients_high`**) to filter the ECG signal. Choose the appropriate filter based on the characteristics of the signal and the type of noise present. After filtering, take a screenshot of the filtered ECG signal and include it in the report. Additionally, explain why you selected that specific filter and how it affects the signal.

***Report - Part 9:*** Compare x_filtered with the original signal (x) and state your observarion.

***Report - Part 10:*** Now, perform the welch algorithm on x_filtered. Take a screenshot of the new Welch PSD and include it in the report.

***Report - Part 11:*** State your observation regarding how the PSD has changed before and after filtering (by comparing it to the original PSD). Notice that we don't need the 'ylim' anymore.

***Report - Part 12:*** To create Butterworth IIR filters, there exist a function called **`butter`** in the **`scipy.signal`** library which operates as follows:

```
b, a = butter(filter_order, Wn, btype='low', analog=False, output='ba', fs=None)
```

where **b** are the coefficients of the input signal x[n] and **a** are the coefficients of the output signal y[n].

Here is a detailed list of how these input arguments work:

- **filter_order**:
  - The **order** of the filter.
  - Higher values provide a **steeper cutoff** but may introduce **phase distortion**.
- **Wn**:
  - The **cutoff frequency** as a single value if the filter is lowpass or highpass. In case the filter is bandpass or bandstop, Wn = [low_cutoff, high_cutoff] is used.
  - If fs is provided, Wn is in **Hz**.
  - If fs is not provided, Wn is **normalized (0 to 1)**, where 1 corresponds to the Nyquist frequency (fs/2).
- **btype**:
  - Specifies the filter type:
    - 'low' → **Low-pass** filter.
    - 'high' → **High-pass** filter.
    - 'bandpass' → **Band-pass** filter.
    - 'bandstop' → **Band-stop (notch)** filter.
- **analog** (default: False):
  - If True, designs an **analog** filter instead of a digital filter.
- **output** (default: 'ba'):
  - Format of the output coefficients:
    - 'ba' → Returns numerator (b) and denominator (a) coefficients.
    - 'zpk' → Returns zeros, poles, and gain (not used in this course).
    - 'sos' → Returns second-order sections (recommended for numerical stability) (not used in this course).
- **fs**:
  - The **sampling frequency** in Hz.
  - If provided, Wn is specified in **Hz** instead of being normalized.

** If your version of Python or the library differs slightly from this implementation of the function (e.g., in terms of input-output arguments), please refer to the function's documentation for the correct details.*

Now, use this function to create a 7$^{th}$ order high-pass Butterworth (IIR) filter with a cutoff frequency of 5 Hz. Take a screenshot of the filter coefficients as well as the code, and include them in the report.

***Report - Part 13:*** Once the filter is created, you can use the following to filter the data. Note that **`lfilter`** belongs to **`scipy.signal`**.

```
filtered_signal = lfilter(b, a, x)
```

Now, filter the ECG (x) and plot it using the following. Take a screenshot of the output plot and include it in the report.

```
# Plot original and filtered signal
plt.figure(figsize=(8, 4))
t = np.linspace(0, 1, len(x), endpoint=False)
plt.plot(t, x, label="Original Signal", alpha=0.5)
plt.plot(t, filtered_signal, label="Filtered Signal", linewidth=2)
plt.legend()
plt.title("Butterworth Filter")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```