BE 521: Homework 1 Questions

Spring 2025

33 points

Due: Jan 30th, 2025

Objective: Working with the IEEG Portal to explore different Neural Signals.

AI Usage Notice

The use of artificial intelligence tools (e.g., large language models, code assistants) is permitted. However, students must explicitly state the specific ways AI was used in completing their work. Failure to disclose AI usage may result in an oral examination to assess understanding, at the discretion of Dr. Litt.

If AI was used in the completion of this assignment, please provide a statement below:

[Enter your statement here]

In [157…
```
!jupyter nbconvert --to html /Prakriti_HW1.ipynb
```

```
[NbConvertApp] Converting notebook /Prakriti_HW1.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 7 image(s).
[NbConvertApp] Writing 1849618 bytes to /Prakriti_HW1.html
```

1. Seizure Activity (16 pts)

The dataset I521_A0001_D002 contains an example of human intracranial EEG (iEEG) data displaying seizure activity. It is recorded from a single channel (2 electrode contacts) implanted in the hippocampus of a patient with temporal lobe epilepsy being evaluated for surgery. In these patients, brain tissue where seizures are seen is often resected. You will do multiple comparisons with this iEEG data and the unit activity that you worked with in Homework 0 (I521_A0001_D001). You will have to refer to that homework and/or dataset for these questions.

In [141…
```
#Set up the notebook environment
!pip install git+https://github.com/ieeg-portal/ieegpy.git # Install ieegpy toolbox directly from github
from ieeg.auth import Session
from IPython.display import Image
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal as sig
```

```
Collecting git+https://github.com/ieeg-portal/ieegpy.git
  Cloning https://github.com/ieeg-portal/ieegpy.git to /tmp/pip-req-build-lq651r5u
  Running command git clone --filter=blob:none --quiet https://github.com/ieeg-portal/ieegpy.git /tmp/pip-req-bu
ild-lq651r5u
  Resolved https://github.com/ieeg-portal/ieegpy.git to commit 080bfa42a8503380ef164b5e7b116613f75073bb
  Preparing metadata (setup.py) ... done
Requirement already satisfied: deprecation in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.32.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.2)
Requirement already satisfied: pennprov==2.2.4 in /usr/local/lib/python3.11/dist-packages (from ieeg==1.6) (2.2.
4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2
.4->ieeg==1.6) (2024.12.14)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.11/dist-packages (from pennprov==2
.2.4->ieeg==1.6) (2.8.2)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->ieeg=
=1.6) (1.17.0)
Requirement already satisfied: urllib3>=1.23 in /usr/local/lib/python3.11/dist-packages (from pennprov==2.2.4->i
eeg==1.6) (2.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from deprecation->ieeg==1.6
) (24.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->ieeg==1.6
) (2025.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from request
s->ieeg==1.6) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->ieeg==1.6
) (3.10)
```

Retrieve the dataset using the IEEGToolbox and generate a session variable as before (nno need to report the output this time). What is the sampling rate of this data? What is the maximum frequency of the signal content that we can resolve? (2 pts)

In [142…
```
#Your code here

#password to open dataset
with open('/content/pra_ieeglogin(9).bin', 'r') as f:
  session = Session('prasadpr', f.read())

#retrieve the dataset
dataset = session.open_dataset('I521_A0001_D002')

key = list(dataset.ts_details.keys())[0]

#sampling rate of this data
time = dataset.get_time_series_details(key)
print(time)
print("Sampling rate of this data: ", time.sample_rate)

# maximum frequency that we can resolve
#To resolve a frequency" means to accurately identify and distinguish a specific frequency within a signal,
# essentially meaning you can clearly separate it from other nearby frequencies, allowing you to precisely meas
#The Nyquist wavenumber/frequency (k = length of time series /2) is the highest frequency that can be resolved
#https://kls2177.github.io/Climate-and-Geophysical-Data-Analysis/chapters/Week6/Harmonics.html

#length of time series = time period/ sampling rate
# Here 644995000 usec (644.995 seconds ) is the time period

time_period = time.duration/1000000
k = time_period /time.sample_rate
print("Maximum frequency of the signal that we can resolve: ", k)
```

```
None(seizure) spans 644995000.0 usec, range [-2216-2221] in 129000 samples. Starts @1 uUTC, ends @644995001 uUTC
with sample rate 200.0 Hz and voltage conv factor 1.0
Sampling rate of this data:  200.0
Maximum frequency of the signal that we can resolve:  3.224975
```

Your answer here

2

How does the duration of this recording compare with the recording from HW0 (I521_A0001_D001)? (2 pts)

In [143…
```
#Your code here

# Comparing two recordings:

# The first dataset I521_A0001_D001

dataset= session.open_dataset('I521_A0001_D001')
key = list(dataset.ts_details.keys())[0]
time1 = dataset.get_time_series_details(key)

print("Duration of the first recording: ", time1.duration)

# The first dataset I521_A0001_D002

dataset= session.open_dataset('I521_A0001_D002')
key = list(dataset.ts_details.keys())[0]
time2 = dataset.get_time_series_details(key)

print("Duration of the second recording: ", time2.duration)

# This recording is longer
```

```
Duration of the first recording:  10000000.0
Duration of the second recording:  644995000.0
```

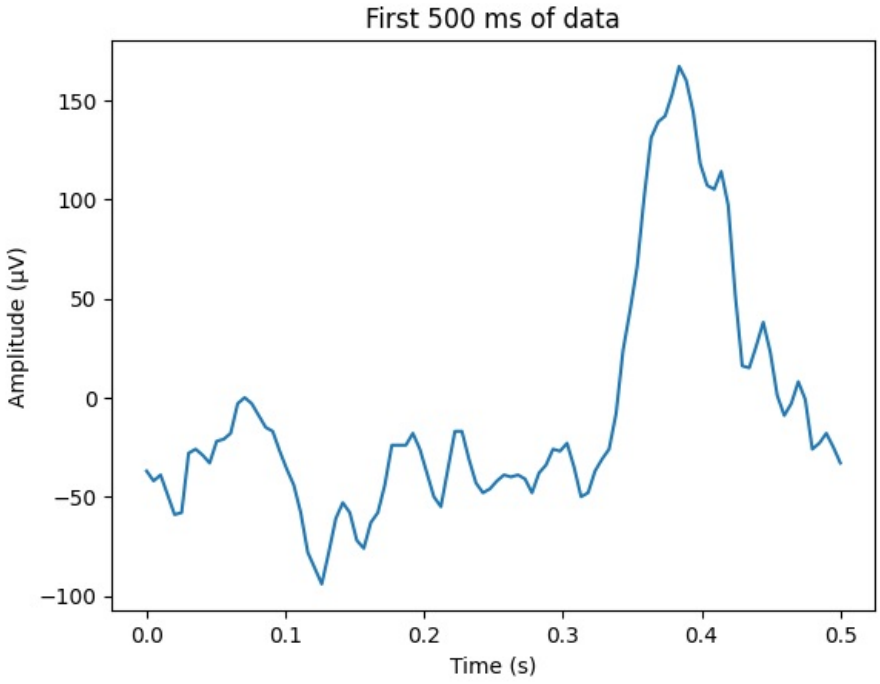
Your answer here

3

Using the time-series visualization functionality of the IEEG Portal, provide a plot of the first 500 ms of data from this recording. (2 pts)

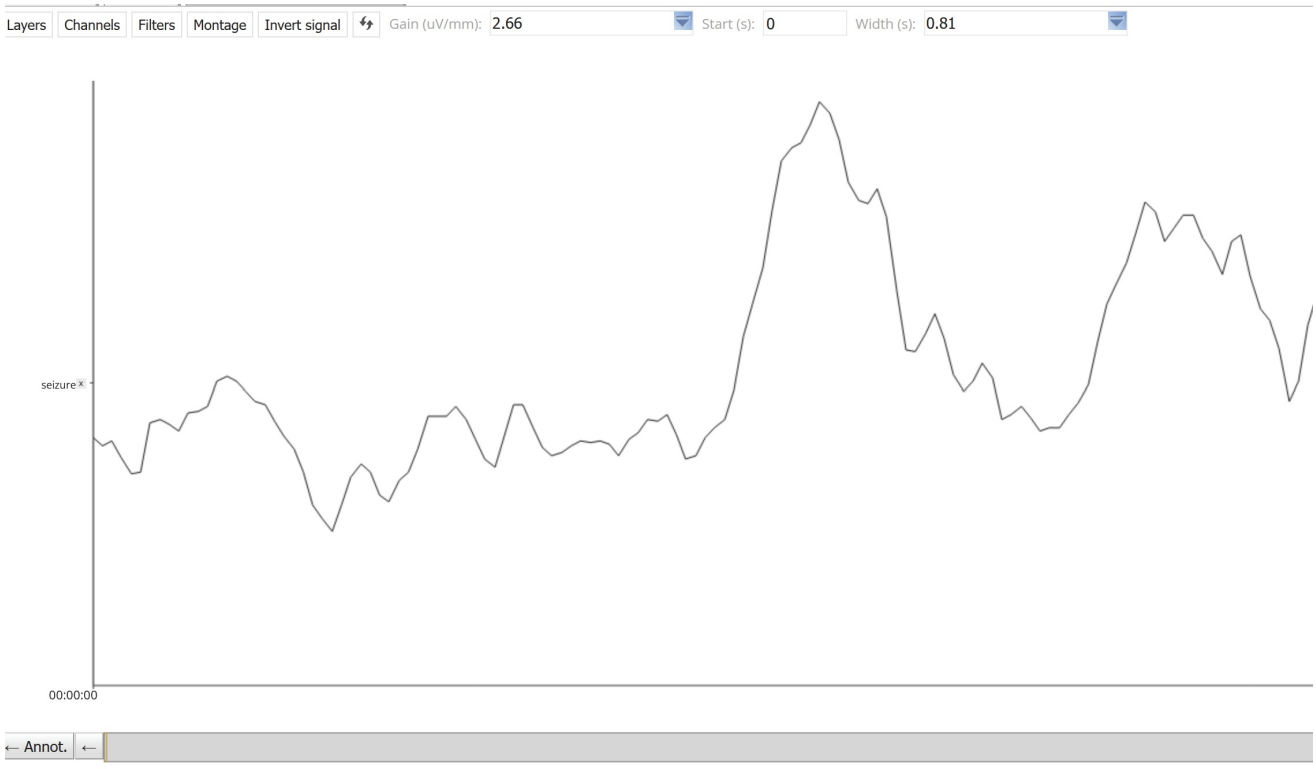
In [144…
```
#Your code here
import matplotlib.pyplot as plt

# Get the first 500 ms of data
key = list(dataset.ts_details.keys())[0]
```

```
channels = dataset.get_channel_indices([key])[0]

start_time = 0
duration = 0.5 #seconds
data_y = dataset.get_data(start_time, duration* 1e6, [channels])

# Plot the first 500 ms of data
time_axis = np.linspace(start_time, start_time + duration, data_y.shape[0])
plt.plot(time_axis, data_y)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (µV)')
plt.title('First 500 ms of data')
plt.show()
plt.tight_layout()
```




```
<Figure size 640x480 with 0 Axes>
```

Your answer here



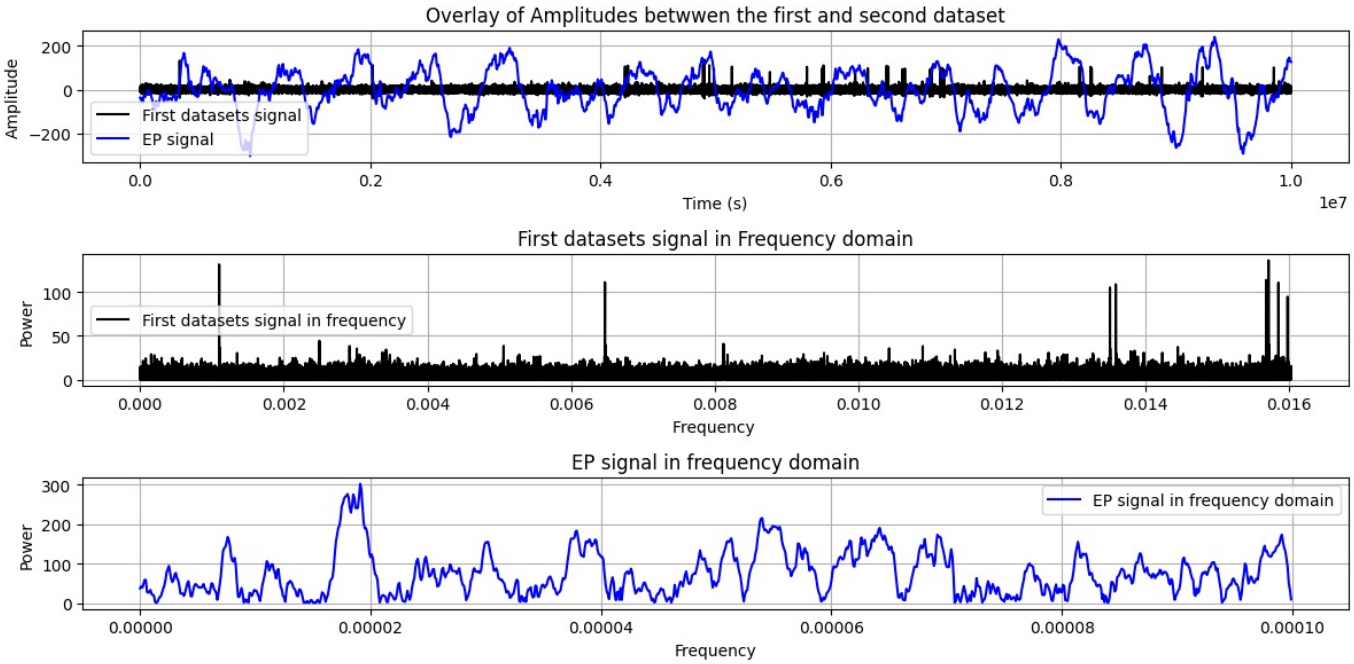
4

Compare the activity in this sample with the data from HW0. What differences do you notice in the amplitude and frequency characteristics? (2 pts)

In [145…

```
import matplotlib.pyplot as plt
```

```python
from scipy.fft import fft, fftfreq

# first dataset
dataset= session.open_dataset('I521_A0001_D001')
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 10000000
first_dataset_signal = dataset.get_data(start_time, duration, [channels])

dataset2 = session.open_dataset('I521_A0001_D002')
key2 = list(dataset2.ts_details.keys())[0]
channel2 = dataset2.get_channel_indices([key2])[0]
second_dataset_signal = dataset2.get_data(start_time, duration, [channel2])

time_axis1 = np.linspace(start_time, duration, first_dataset_signal.shape[0])
time_axis2 = np.linspace(start_time, duration, second_dataset_signal.shape[0])

# Create the plot
plt.figure(figsize=(12, 6))


plt.subplot(3, 1, 1)
plt.plot(time_axis1, first_dataset_signal, color='black', label='First datasets signal')
plt.plot(time_axis2, second_dataset_signal, color='blue', label='EP signal')
plt.title('Overlay of Amplitudes between the first and second dataset')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()
plt.tight_layout()


# # plt.subplot(4, 1, 2)
# plt.plot(time_axis2, second_dataset_signal, color='blue', label='EP signal')
# plt.title('EP signal')
# plt.xlabel('Time (s)')
# plt.ylabel('Amplitude')
# plt.grid(True)
# plt.legend()


N1 = first_dataset_signal.shape[0]
N2 = second_dataset_signal.shape[0]


# Compute FFT
first_dataset_fft = fft(first_dataset_signal)
second_dataset_fft = fft(second_dataset_signal)
y1 =  np.abs(first_dataset_fft[:N1 // 2])
y2 =  np.abs(second_dataset_fft[:N2 // 2])


# Compute frequency axes
sampling_rate1 = N1 / (duration - start_time)
sampling_rate2 = N2 / (duration - start_time)
freqs1 = fftfreq(N1, 1 / sampling_rate1)
freqs2 = fftfreq(N2, 1 / sampling_rate2)
x1 = freqs1[:N1 // 2]
x2 = freqs2[:N2 // 2]

plt.subplot(3, 1, 2)
plt.plot(x1, y1, color='black', label='First datasets signal in frequency')
plt.title('First datasets signal in Frequency domain')
plt.xlabel('Frequency ')
plt.ylabel('Power')
plt.grid(True)
plt.legend()
plt.tight_layout()


plt.subplot(3, 1, 3)
plt.plot(x2, y2, color='blue', label='EP signal in frequency domain')
plt.title('EP signal in frequency domain')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid(True)
plt.legend()
plt.tight_layout()
```



To compare the Amplitudes:

Comparing the first and second datasets in the time domain, the amplitude of the first signal is relatively low with occasional sharp spikes.

The amplitude in the second signal(EP) is higher and varies over time. It also shows some pattern as compared to the first signal which has random spikes.

To compare the Frequencies:

In the first signal, the power specturm shows several sharp peaks, indicating distinct frequency components. Most power is at lower frequencies.

In the EP signal, The power spectrum is smoother and spread over. It is a more continuous frequency distribution.

Your answer here

5

The unit activity sample in (I521_A0001_D001) was high-pass filtered to remove low-frequency content. Assume that the seizure activity in (I521_A0001_D002) has not been high-pass filtered. Given that the power of a frequency band scales roughly as $1/f$, how might these di erences in preprocessing contribute to the differences you noted in the previous question? (There is no need to get into specific calculations here. We just want general ideas.) (3 pts)

Impact on Amplitude Differences:

First Signal (I521_A0001_D001, High-Pass Filtered): The removal of low-frequency components reduces the overall amplitude of the signal, except for occasional sharp spikes, which are likely brief high-frequency events that were preserved. This explains why the first signal appears to have lower amplitude with random spikes. Second Signal (I521_A0001_D002, Unfiltered): Since low-frequency components were retained, the overall amplitude remains higher and more variable over time, contributing to the smoother and more structured patterns you observed.

Impact on Frequency Differences:

First Signal (I521_A0001_D001, High-Pass Filtered): The power spectrum has sharp peaks at distinct frequencies because only higher-frequency components remain. The removal of low-frequency components makes the spectrum more selective. Second Signal (I521_A0001_D002, Unfiltered): The power spectrum appears smoother and more spread out, as lower-frequency components are still present. Given that power scales as 1/f, unfiltered signals tend to have more power at lower frequencies, creating a continuous distribution instead of distinct peaks.

Your answer here

6

Two common methods of human iEEG are known as electrocorticography (ECoG) and stereoelectroencephalography (SEEG). For either of these paradigms (please indicate which you choose), find and report at least two of the following electrode characteristics: shape, material, size. Please note that exact numbers aren't required, and please cite any sources used. (3 pts)

Ecog:

Electrode characteristics:

- Shape: An ECoG array consists of a strip or a grid of electrodes. In general, a strip has 2–12 electrodes arranged in a line, and a grid has 8–64 electrodes arranged in a grid.
- Material: Platinum electrodes are preferred for their superior resistance to electrolysis and stable electrochemical properties
- Size: The size of the metal electrodes ranges from 3 to 6 mm, and the size of the exposure is slightly smaller.

Source :https://onlinelibrary.wiley.com/doi/full/10.1002/admt.202301692 The above information can be found above figure 3 in this paper

SEEG:

Electrode characteristics:

- Shape: conical shape, or H6 hexagonal base
- Material: • Titanium alloy • Silicone • Polymer
- Size: 0.8mm or 37 microns to 18 microns

source: http://epi-for.org/wp-content/uploads/2020/11/ALCIS-NEURO-CATALOGUE.pdf

Your answer here

7

What is a local field potential? How might the characteristics of human iEEG electrodes cause them to record local field potentials as opposed to multiunit activity, which was the signal featured in HW0 as recorded from 40 micron Pt-Ir microwire electrodes? (2 pts)

A local field potential (LFP) is the low-frequency (<300 Hz) electrical activity recorded from a population of neurons. It represents the summed synaptic activity (both excitatory and inhibitory) in a localized brain region. LFPs mainly capture slow oscillatory activity rather than the rapid spiking of individual neurons.

The difference in recorded signals between human intracranial EEG (iEEG) electrodes and 40-micron Pt-Ir microwire electrodes (used in HW0) is largely due to electrode size and impedance:

1. Electrode Size & Spatial Averaging

   iEEG Electrodes (Macro-scale, mm-sized) These electrodes are relatively large (on the order of millimeters), meaning they collect signals over a large population of neurons. This results in a spatially averaged signal, where fast, local spiking activity cancels out, leaving only the slower oscillatory components (LFPs).

   Microwire Electrodes (Micro-scale, ~40 μm) These electrodes are much smaller, allowing them to detect spikes from individual neurons. Their high impedance (~100 kΩ) makes them sensitive to fast, high-frequency action potentials rather than low-frequency synaptic activity.

2. Impedance Differences

   ```
   Larger electrodes (iEEG) have lower impedance(~1-10 kΩ), making them more responsive to low-frequency signals (LFPs). Smaller electrodes (microwires) have high impedance (~100 k Ω), making them better suited for detecting high-frequency spikes from individual neurons.
   ```

Your answer here

2. Evoked Potentials (17 pts)

The data in I521 A0001 D003 contains an example of a very common type of experiment and neuronal signal, the evoked potential (EP). The data show the response of the whisker barrel cortex region of rat brain to an air puff stimulation of the whiskers. The stim channel shows the stimulation pattern, where the falling edge of the stimulus indicates the start of the air puff, and the rising edge indicates the end. The ep channel shows the corresponding evoked potential. Once again, play around with the data on the IEEG Portal, in particular paying attention to the effects of stimulation on EPs. You should observe the data with window widths of 60 secs as well as 1 sec. Again, be sure to explore the signal gain to get a more accurate picture. Finally, get a sense for how long the trials are (a constant duration), and how long the entire set of stimuli and responses are.

1

Based on your observations, should we use all of the data or omit some of it? (There's no right answer, here, just make your case either way in a few sentences.) (2 pts)

In [158…

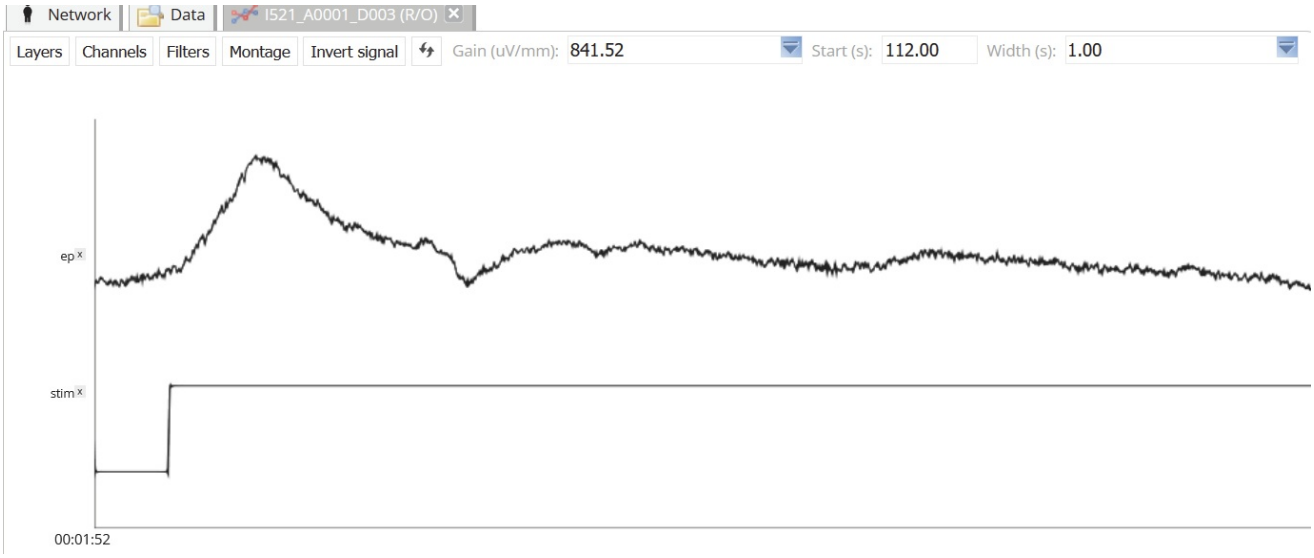
```
#Your code here
# An evoked potential (EP) is a record of the electrical activity in the brain in response to a stimulus.
# Evoked potentials are used to assess sensory and motor nerve function, and to diagnose neurological condition
```

```
# You should observe the data with window widths of 60 secs as well as 1 sec.
# Again, be sure to explore the signal gain to get a more accurate picture.
# Finally, get a sense for how long the trials are (a constant duration), and how long the entire set of stimul


dataset= session.open_dataset('I521_A0001_D003')
print(dataset.ts_details.keys())

key = list(dataset.ts_details.keys())[0]
time_ep = dataset.get_time_series_details(key)

key1 = list(dataset.ts_details.keys())[1]
Time_sim = dataset.get_time_series_details(key1)

# This is how long the entire set of stimuli and responses are:
print("Duration of the ep: ", time_ep.duration)
print("Sampling rate of ep", time_ep.sample_rate)
print("Duration of stim", Time_sim.duration)
print("Sampling rate of stim", Time_sim.sample_rate)
```

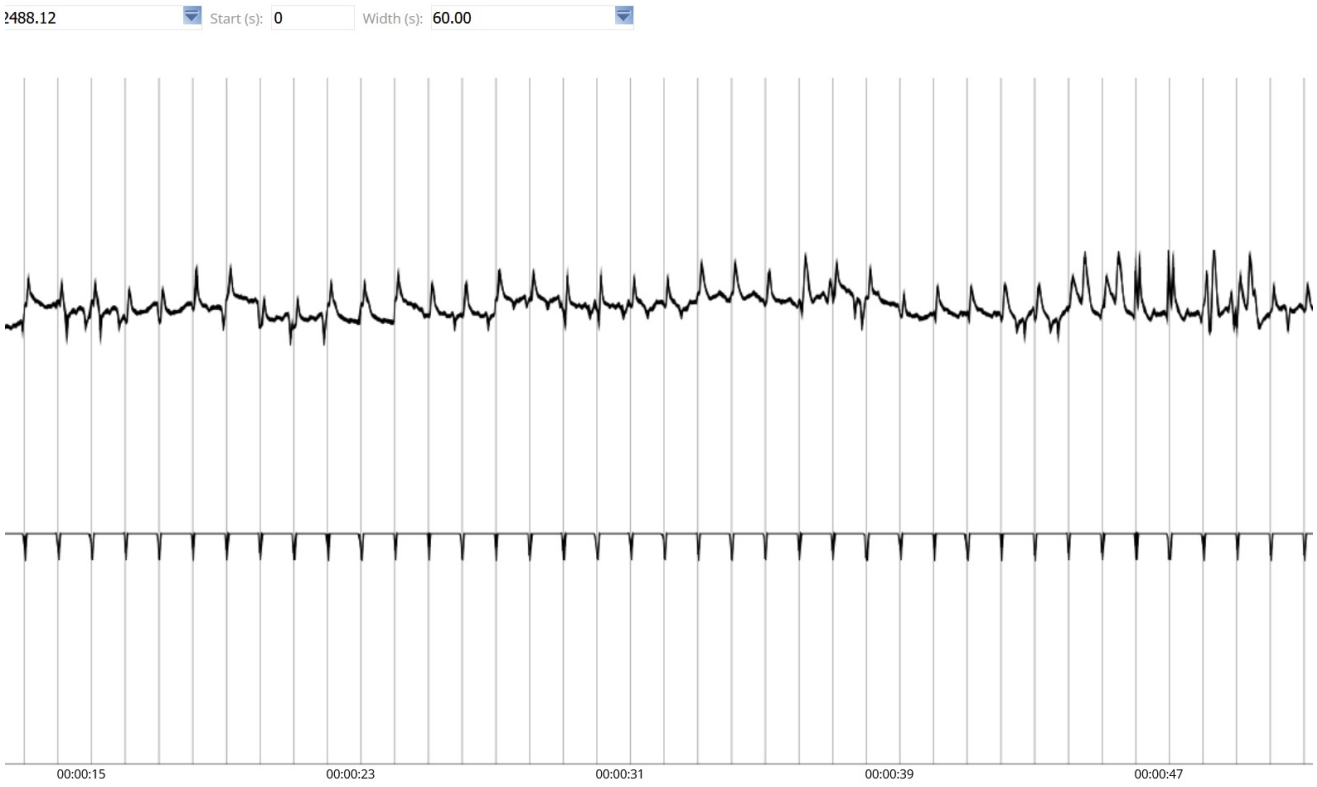
```
dict_keys(['ep', 'stim'])
Duration of the ep:  117999631.0
Sampling rate of ep 2713.0
Duration of stim 117999631.0
Sampling rate of stim 2713.0
```

Observing it at a width of 1 second:

*The usual trend that I observed is that: in the rising edge of stim, after a lag, the EP signal starts to rise to form a peak. Except for 3-4 times when it doesn't.



Observing it at a width of 60 seconds:



Based on my observation, I think we can use all of the data as it is pretty consistent, that is on rising edges of stim the EP starts to begin and forms a peak after a little lag.

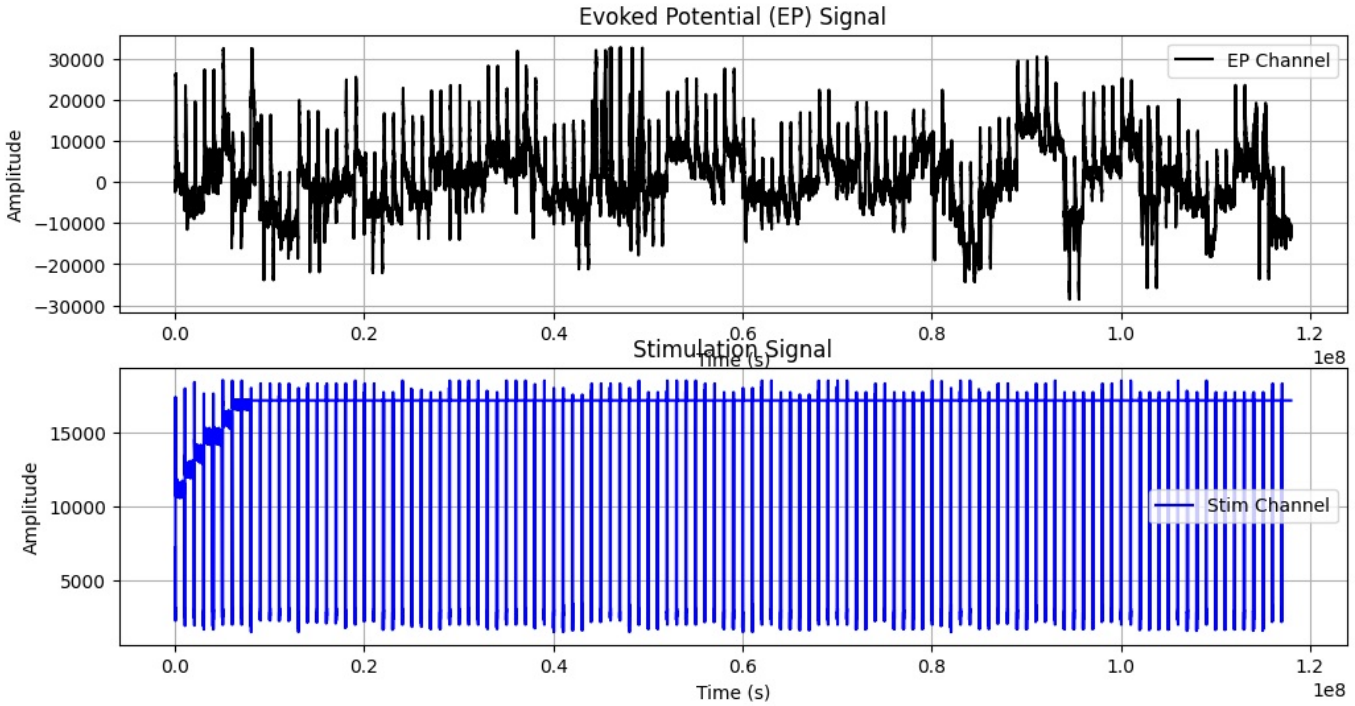
2

Retrieve the ep and stim channel data in Colab. What is the average latency (in ms) of the peak response to the stimulus onset over all trials? (Assume stimuli occurs at exactly 1 second intervals.) (3 pts)

In [147…

```
from scipy.signal import find_peaks
import numpy as np
import matplotlib.pyplot as plt

dataset = session.open_dataset('I521_A0001_D003')

# Extract EP and Stim channel information
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 117999631
data_ep = dataset.get_data(start_time, duration, [channels]).flatten()

key1 = list(dataset.ts_details.keys())[1]
channels1 = dataset.get_channel_indices([key1])[0]
data_stim = dataset.get_data(start_time, duration, [channels1])

# Generate time axis
time_axis = np.linspace(start_time, duration, len(data_ep))

# Plot EP and Stim channels
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(time_axis, data_ep, color='black', label='EP Channel')
plt.title('Evoked Potential (EP) Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(time_axis, data_stim, color='blue', label='Stim Channel')
plt.title('Stimulation Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()

plt.show()

# Spike-Triggered Analysis
```

```
time_domain = 500000  # 500 ms window
interval = 1e6 / 2713  # Sampling interval
latencies = []

for start in range(60000, duration, 1000000):

    # Extract a window of data
    ep_data_window = dataset.get_data(start, time_domain, [channels]).flatten()

    # Find the peak index
    peak_sample_index = ep_data_window.argmax()
    time_of_this_peak = start + peak_sample_index * interval

    # Compute latency
    latency = time_of_this_peak - start
    latencies.append(latency)

# Compute final latency and average response
average_latency_ms = np.mean(latencies) * 0.001  # Convert to ms

print(f"Average Latency: {average_latency_ms:.3f} ms")
plt.tight_layout()
```




```
Average Latency: 88.394 ms
<Figure size 640x480 with 0 Axes>
```

Your answer here

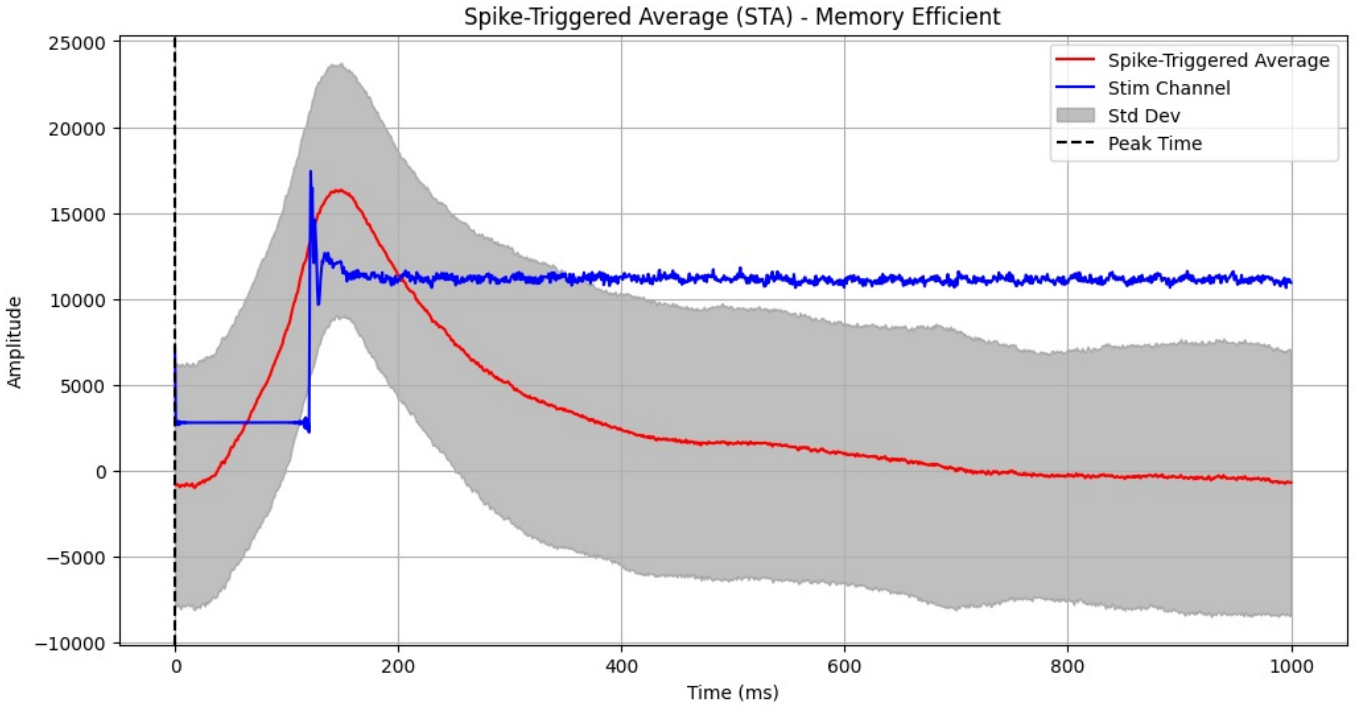
3

In neuroscience, we often need to isolate a small neural signal buried under an appreciable amount of noise. One technique to accomplish this is called the "spike triggered average", sometimes called "signal averaging". This technique assumes that the neural response to a repetitive stimulus is constant (or nearly so), while the noise fluctuates from trial to trial - therefore averaging the evoked response over many trials will isolate the signal and average out the noise. Construct a spike triggered average plot for the data in I521 A0001 D003. Plot the average EP in red. Overlay the standard deviation of the responses in grey. You may find `plt.fill_between` and `np.linspace` to be helpful. Be sure to include an informative legend. You may also find this to be helpful for what colors plyplot natively supports (4 pts).

In [148…

```
from scipy.signal import find_peaks
import numpy as np
import matplotlib.pyplot as plt

dataset = session.open_dataset('I521_A0001_D003')

# Extract EP and Stim channel information
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 117999631
data_ep = dataset.get_data(start_time, duration, [channels]).flatten()

key1 = list(dataset.ts_details.keys())[1]
channels1 = dataset.get_channel_indices([key1])[0]
```

```
data_stim = dataset.get_data(start_time, duration, [channels1])

# Spike-Triggered Analysis
time_domain = 500000  # 500 ms window
interval = 1e6 / 2713  # Sampling interval
meann = []

for start in range(60000, duration, 1000000):

    # Extract a window of data
    ep_data_window = dataset.get_data(start, time_domain, [channels]).flatten()
    meann.append(ep_data_window)


# Compute final latency and average response
meann = np.array(meann)
mean_waveform = np.mean(meann, axis=0)  # Compute mean waveform
std_dev = np.std(meann, axis=0)  # Compute standard deviation



# Create time axis for STA plot
time_axis_mean = np.linspace(0, 1000, len(mean_waveform))
# Create time axis for stim data (same as time_axis_mean for alignment)
time_axis_stim = np.linspace(0, 1000, len(data_stim[:len(time_axis_mean)]))

# Plot Spike-Triggered Average
plt.figure(figsize=(12, 6))
plt.plot(time_axis_mean, mean_waveform, color='red', label='Spike-Triggered Average')
# Plot the stimulation data (overlay)
plt.plot(time_axis_stim, data_stim[:len(time_axis_mean)], color='blue', label='Stim Channel')
plt.fill_between(time_axis_mean, mean_waveform - std_dev, mean_waveform + std_dev, color='grey', alpha=0.5, lab
plt.axvline(0, color='black', linestyle='--', label='Peak Time')
plt.title('Spike-Triggered Average (STA) ')
plt.xlabel('Time (ms)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()
```



In [149…

```
# # #Your code here - Please ignore this

# # # signal averaging - construct a spike triggered average plot for the data
# # # plot the average EP in red
# # # sd from orginal in grey
# # # include legend

# # # https://medium.com/@baxterbarlow/spike-trigger-averages-sta-understanding-neural-spike-stimuli-via-mathema
# # # Each time a spike appears, the stimulus in a time window preceding the spike is recorded.
# # # The recorded stimuli for all spikes are then summed and the procedure is repeated over multiple trials.
# # # In other words, for a spike occurring at time tᵢ, we determine s(tᵢ − τ), and
# # # then we sum over all n spikes in a trial, i = 1, 2, …, n, and divide the total by n, then average over tr
# import numpy as np
# import matplotlib.pyplot as plt
# from scipy.signal import find_peaks
```

```
# # Load dataset
# dataset = session.open_dataset('I521_A0001_D003')
# key = list(dataset.ts_details.keys())[0]
# channels = dataset.get_channel_indices([key])[0]
# start_time = 0
# duration = 117999631
# data_ep = dataset.get_data(start_time, duration, [channels]).flatten()

# # Find peaks
# peaks, _ = find_peaks(data_ep, height=-40000)  # Adjust height threshold if needed

# # Define window size (100 samples after each peak)
# window_size = 50
# avg = np.zeros(window_size)  # Initialize accumulator array
# resp = np.zeros(window_size)  # Counter for valid contributions

# # Iterate over all detected peaks
# for peak in peaks:
#     start = peak
#     end = min(len(data_ep), peak + window_size)

#     if end - start == window_size:  # Ensure full window size
#         avg += data_ep[start:end]  # Accumulate signals
#         resp += 1  # Count contributions per sample

# # Compute final average (element-wise division)
# avg /= np.maximum(resp, 1)  # Avoid division by zero

# # Compute standard deviation in a memory-efficient way
# std_accumulator = np.zeros(window_size)

# for peak in peaks:
#     start = peak
#     end = min(len(data_ep), peak + window_size)

#     if end - start == window_size:
#         std_accumulator += (data_ep[start:end] - avg) ** 2

# std_dev = np.sqrt(std_accumulator / np.maximum(resp, 1))

# # Create time axis from 0 to window_size
# time_axis = np.arange(0, window_size)

# # Plot results
# plt.figure(figsize=(12, 6))
# plt.plot(time_axis, avg, color='red', label='Spike-Triggered Average')
# plt.fill_between(time_axis, avg - std_dev, avg + std_dev, color='grey', alpha=0.5, label='Std Dev')
# plt.axvline(0, color='black', linestyle='--', label='Peak Time')
# plt.title('Spike-Triggered Average (STA) - Memory Efficient')
# plt.xlabel('Time (Samples)')
# plt.ylabel('Amplitude')
# plt.legend()
# plt.grid()
# plt.show()
```

Your answer here

4.

4a

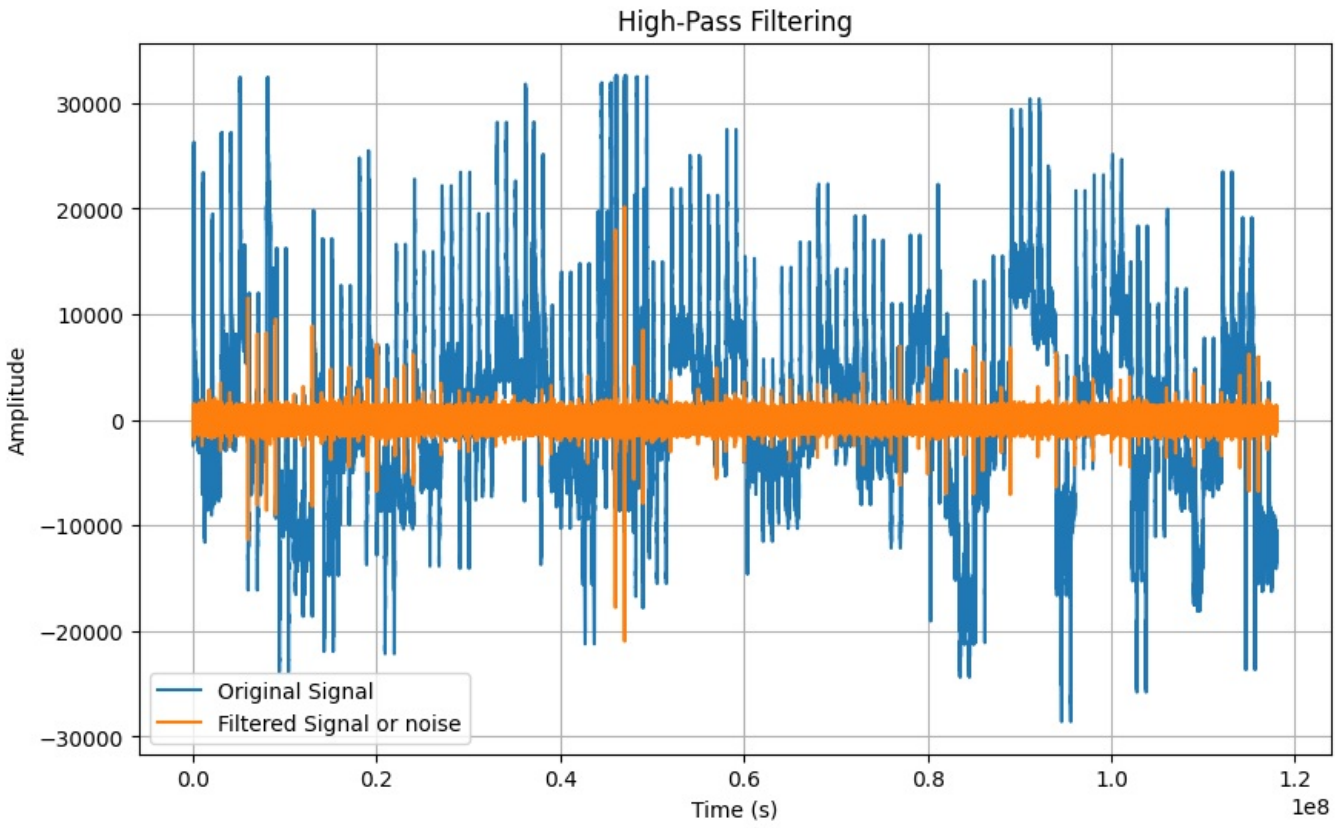
We often want to get a sense for the amplitude of the noise in a single trial. Propose a method to do this (there are a few reasonably simple methods, so no need to get too complicated). Note: do not assume that the signal averaged EP is the "true" signal and just subtract it from that of each trial, because whatever method you propose should be able to work on the signal from a single trial or from the average of the trials. (4 pts)

In [150…

```
# Two methods:
# 1 . Standard Deviation of the High-Frequency Components
# Assume that the low-frequency content of the EP signal contains the actual evoked response, while the high-fr
# Steps:
#     Apply a high-pass filter (e.g., >30 Hz) to isolate noise.
#     Compute the standard deviation of the filtered signal as a measure of noise amplitude.

# 2. Root Mean Square (RMS) of the Residual Signal

#     If we assume the signal is relatively smooth, you can measure how much it fluctuates by computing the RMS
#     Steps:
#         Apply a low-pass filter to get a smooth version of the signal.
```

```
#         Subtract the low-pass signal from the original to get the "residual" noise.
#         Compute the RMS of the residual as a noise estimate.
# Source: https://medium.com/@ChanakaDev/low-pass-high-pass-and-band-pass-filters-with-scipy-python-a87b2332ce2

import numpy as np
from scipy.signal import butter, filtfilt
import matplotlib.pyplot as plt


# Load dataset
dataset = session.open_dataset('I521_A0001_D003')
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 117999631
data_ep = dataset.get_data(start_time, duration, [channels]).flatten()


# Filter parameters
cutoff_freq = 50  # Cutoff frequency in Hz - This is the variable we need to adjust
fs = 2713  # Sampling rate in Hz
order = 4  # Filter order

# Design the high-pass filter
nyq = 0.5 * fs
normal_cutoff = cutoff_freq / nyq
b, a = butter(order, normal_cutoff, btype='highpass')


time_axis = np.linspace(0, duration, len(data_ep))
# time_axis = np.arange(len(data_ep)) / fs

# Apply the filter
filtered_data = filtfilt(b, a, data_ep)
std_dev = np.std(filtered_data)

print(f"Standard Deviation of High-Frequency Components or amplitude of noise is: {std_dev}")

# Plot the original and filtered data
plt.figure(figsize=(10, 6))

plt.plot(time_axis, data_ep, label='Original Signal')
plt.plot(time_axis, filtered_data, label='Filtered Signal or noise')

plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('High-Pass Filtering')
plt.legend()
plt.grid(True)

plt.show()
```

```
Standard Deviation of High-Frequency Components or amplitude of noise is: 551.4879963539308
```



Your answer here

4b

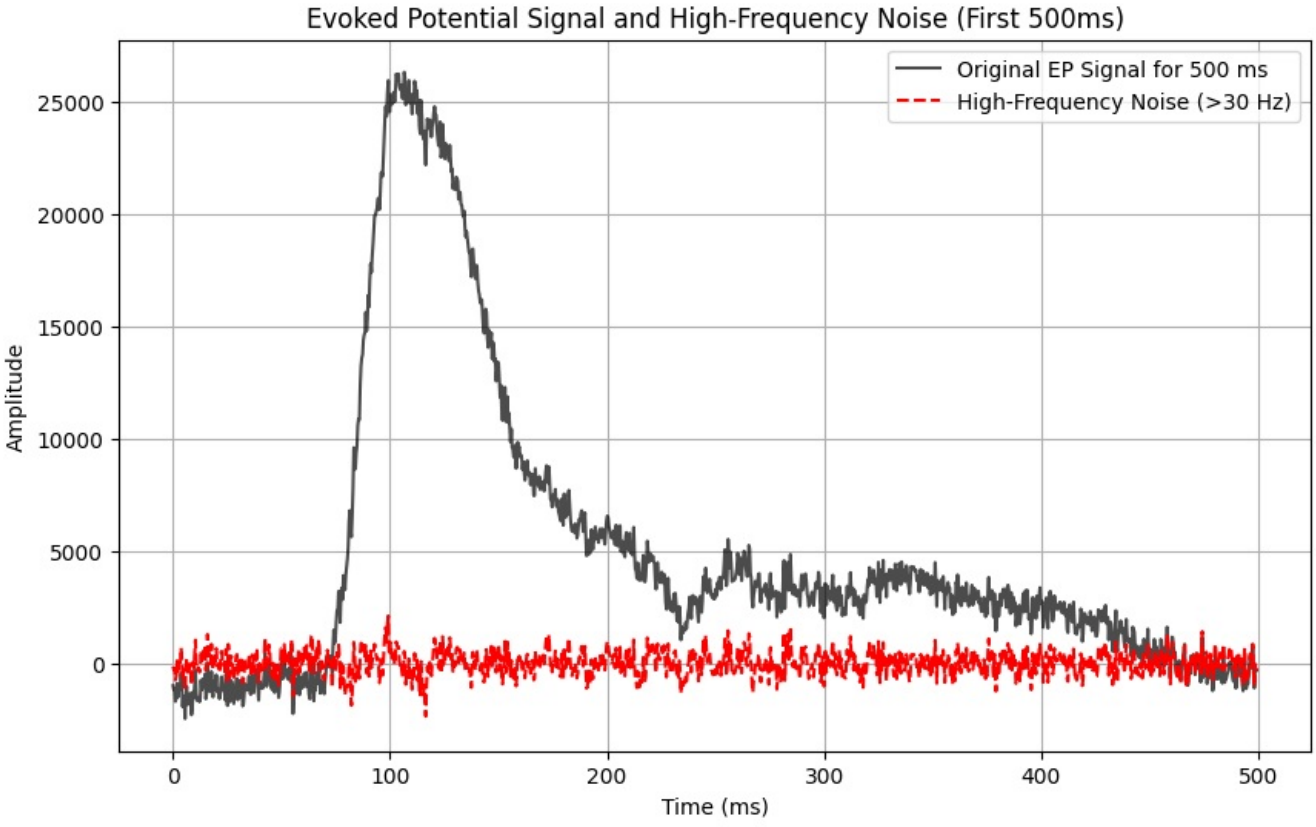
Show with a few of the EPs (plots and/or otherwise) that your method gives reasonable results. You may find `sig.butter` and `sig.filtfilt` helpful. (1 pt)

In [151…

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Load dataset
dataset = session.open_dataset('I521_A0001_D003')
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 117999631
data_ep = dataset.get_data(start_time, duration, [channels]).flatten()
```

```python
# Sampling rate (Hz)
fs = 2713

# Select only the first 500 ms of data
time_window = int(0.5 * fs)  # Convert 500 ms to samples
data_ep = data_ep[:time_window]  # Keep only first 500 ms

# High-pass filter parameters
cutoff_freq = 30  # Cutoff frequency in Hz
order = 4

# Design high-pass Butterworth filter
nyq = 0.5 * fs  # Nyquist frequency
normal_cutoff = cutoff_freq / nyq
b, a = signal.butter(order, normal_cutoff, btype='highpass')

# Apply high-pass filter
filtered_data = signal.filtfilt(b, a, data_ep)  # Isolated high-frequency noise
print(filtered_data.size)


# Compute standard deviation of noise
std_dev = np.std(filtered_data)
print(f"Standard Deviation of High-Frequency Noise: {std_dev:.2f}")

# Create time axis in milliseconds
time_axis = np.arange(len(data_ep)) / fs * 1000  # Convert to milliseconds

# ---------- Plot the Results ----------
plt.figure(figsize=(10, 6))

# Plot original EP signal
plt.plot(time_axis, data_ep, label='Original EP Signal for 500 ms', color='black', alpha=0.7)

# Plot high-frequency noise (filtered signal)
plt.plot(time_axis, filtered_data, label='High-Frequency Noise (>30 Hz)', color='red', linestyle='dashed')

plt.xlabel('Time (ms)')
plt.ylabel('Amplitude')
plt.title('Evoked Potential Signal and High-Frequency Noise (First 500ms)')
plt.legend()
plt.grid(True)

plt.show()
```

```
1356
Standard Deviation of High-Frequency Noise: 523.66
```



Your answer here

4c

Apply your method on each individual trial and report the mean noise amplitude across all trials. (1 pt)

In [152…

```
# mean noise amplitude across all trials

# Sampling rate (Hz)
fs = 2713
time_window = 1000000  # Convert 500 ms to samples
# High-pass filter parameters
cutoff_freq = 30  # Cutoff frequency in Hz
order = 4

# Design high-pass Butterworth filter
nyq = 0.5 * fs  # Nyquist frequency
normal_cutoff = cutoff_freq / nyq
b, a = signal.butter(order, normal_cutoff, btype='highpass')  # Correctly defined
data_ep = dataset.get_data(start, duration, [channels]).flatten()


# # Loop over time segments of 500 ms
# for i in range(start_time, duration, time_window):

#     trial_data = data_ep[i : i + time_window]  # Extract 500 ms segment
#     # Apply high-pass filter
#     filtered_data = signal.filtfilt(b, a, trials)

#     # Compute mean noise amplitude for this trial
#     mean_noise = np.mean(np.abs(filtered_data))  # Take absolute mean
#     # mean_noise_per_trial.append(mean_noise)

#     print(f"Mean noise amplitude for trial starting at {i} is {mean_noise:.2f}")

# # Compute overall mean noise amplitude across all trials
# overall_mean_noise = np.mean(mean_noise)
# print(f"\nOverall Mean Noise Amplitude Across All Trials: {overall_mean_noise:.2f}")


all_trials = []
stimulus_interval = 1000000
window = 500000
for i in range(0, 117999631, stimulus_interval):

    # Extracting signal data for a window of 500 ms around the stimulus onset
    signal_data = dataset.get_data(i, window, [channels]).flatten()
    all_trials.append(signal_data)

    # Applying low-pass filter to extract EP
    filtered_signal = sig.filtfilt(b, a, signal_data)

    noise_amplitude = np.abs(np.mean(filtered_signal))

    print(f"Mean noise amplitude for trial starting at {i} is {noise_amplitude:.2f}")
```

```
Mean noise amplitude for trial starting at 0 is 1.99
Mean noise amplitude for trial starting at 1000000 is 2.63
Mean noise amplitude for trial starting at 2000000 is 2.08
Mean noise amplitude for trial starting at 3000000 is 2.69
Mean noise amplitude for trial starting at 4000000 is 0.63
Mean noise amplitude for trial starting at 5000000 is 8.32
Mean noise amplitude for trial starting at 6000000 is 2.78
Mean noise amplitude for trial starting at 7000000 is 3.93
Mean noise amplitude for trial starting at 8000000 is 3.81
Mean noise amplitude for trial starting at 9000000 is 3.36
Mean noise amplitude for trial starting at 10000000 is 9.11
Mean noise amplitude for trial starting at 11000000 is 5.08
Mean noise amplitude for trial starting at 12000000 is 5.08
Mean noise amplitude for trial starting at 13000000 is 1.64
Mean noise amplitude for trial starting at 14000000 is 7.83
Mean noise amplitude for trial starting at 15000000 is 5.91
Mean noise amplitude for trial starting at 16000000 is 1.96
Mean noise amplitude for trial starting at 17000000 is 1.64
Mean noise amplitude for trial starting at 18000000 is 4.11
Mean noise amplitude for trial starting at 19000000 is 4.47
Mean noise amplitude for trial starting at 20000000 is 1.36
Mean noise amplitude for trial starting at 21000000 is 1.36
Mean noise amplitude for trial starting at 22000000 is 0.92
Mean noise amplitude for trial starting at 23000000 is 2.15
Mean noise amplitude for trial starting at 24000000 is 3.33
Mean noise amplitude for trial starting at 25000000 is 7.79
Mean noise amplitude for trial starting at 26000000 is 1.28
Mean noise amplitude for trial starting at 27000000 is 2.05
Mean noise amplitude for trial starting at 28000000 is 2.05
```

```
Mean noise amplitude for trial starting at 29000000 is 9.76
Mean noise amplitude for trial starting at 30000000 is 0.21
Mean noise amplitude for trial starting at 31000000 is 3.58
Mean noise amplitude for trial starting at 32000000 is 1.16
Mean noise amplitude for trial starting at 33000000 is 3.57
Mean noise amplitude for trial starting at 34000000 is 8.27
Mean noise amplitude for trial starting at 35000000 is 3.29
Mean noise amplitude for trial starting at 36000000 is 3.35
Mean noise amplitude for trial starting at 37000000 is 2.63
Mean noise amplitude for trial starting at 38000000 is 2.69
Mean noise amplitude for trial starting at 39000000 is 0.83
Mean noise amplitude for trial starting at 40000000 is 3.94
Mean noise amplitude for trial starting at 41000000 is 4.62
Mean noise amplitude for trial starting at 42000000 is 8.16
Mean noise amplitude for trial starting at 43000000 is 3.25
Mean noise amplitude for trial starting at 44000000 is 6.60
Mean noise amplitude for trial starting at 45000000 is 6.60
Mean noise amplitude for trial starting at 46000000 is 2.52
Mean noise amplitude for trial starting at 47000000 is 4.03
Mean noise amplitude for trial starting at 48000000 is 3.18
Mean noise amplitude for trial starting at 49000000 is 1.19
Mean noise amplitude for trial starting at 50000000 is 0.18
Mean noise amplitude for trial starting at 51000000 is 0.18
Mean noise amplitude for trial starting at 52000000 is 3.15
Mean noise amplitude for trial starting at 53000000 is 12.90
Mean noise amplitude for trial starting at 54000000 is 6.58
Mean noise amplitude for trial starting at 55000000 is 0.73
Mean noise amplitude for trial starting at 56000000 is 2.49
Mean noise amplitude for trial starting at 57000000 is 10.21
Mean noise amplitude for trial starting at 58000000 is 1.31
Mean noise amplitude for trial starting at 59000000 is 1.71
Mean noise amplitude for trial starting at 60000000 is 0.46
Mean noise amplitude for trial starting at 61000000 is 0.56
Mean noise amplitude for trial starting at 62000000 is 2.52
Mean noise amplitude for trial starting at 63000000 is 1.06
Mean noise amplitude for trial starting at 64000000 is 3.19
Mean noise amplitude for trial starting at 65000000 is 0.20
Mean noise amplitude for trial starting at 66000000 is 4.45
Mean noise amplitude for trial starting at 67000000 is 6.27
Mean noise amplitude for trial starting at 68000000 is 1.84
Mean noise amplitude for trial starting at 69000000 is 1.84
Mean noise amplitude for trial starting at 70000000 is 8.02
Mean noise amplitude for trial starting at 71000000 is 1.44
Mean noise amplitude for trial starting at 72000000 is 1.85
Mean noise amplitude for trial starting at 73000000 is 0.47
Mean noise amplitude for trial starting at 74000000 is 1.57
Mean noise amplitude for trial starting at 75000000 is 1.57
Mean noise amplitude for trial starting at 76000000 is 2.47
Mean noise amplitude for trial starting at 77000000 is 2.28
Mean noise amplitude for trial starting at 78000000 is 0.73
Mean noise amplitude for trial starting at 79000000 is 4.58
Mean noise amplitude for trial starting at 80000000 is 0.50
Mean noise amplitude for trial starting at 81000000 is 1.94
Mean noise amplitude for trial starting at 82000000 is 4.11
Mean noise amplitude for trial starting at 83000000 is 3.38
Mean noise amplitude for trial starting at 84000000 is 3.38
Mean noise amplitude for trial starting at 85000000 is 0.61
Mean noise amplitude for trial starting at 86000000 is 2.47
Mean noise amplitude for trial starting at 87000000 is 0.65
Mean noise amplitude for trial starting at 88000000 is 0.51
Mean noise amplitude for trial starting at 89000000 is 2.96
Mean noise amplitude for trial starting at 90000000 is 0.13
Mean noise amplitude for trial starting at 91000000 is 1.33
Mean noise amplitude for trial starting at 92000000 is 11.34
Mean noise amplitude for trial starting at 93000000 is 5.03
Mean noise amplitude for trial starting at 94000000 is 1.22
Mean noise amplitude for trial starting at 95000000 is 1.76
Mean noise amplitude for trial starting at 96000000 is 11.83
Mean noise amplitude for trial starting at 97000000 is 3.62
Mean noise amplitude for trial starting at 98000000 is 5.51
Mean noise amplitude for trial starting at 99000000 is 0.51
Mean noise amplitude for trial starting at 100000000 is 4.90
Mean noise amplitude for trial starting at 101000000 is 0.17
Mean noise amplitude for trial starting at 102000000 is 3.55
Mean noise amplitude for trial starting at 103000000 is 1.94
Mean noise amplitude for trial starting at 104000000 is 3.91
Mean noise amplitude for trial starting at 105000000 is 5.50
Mean noise amplitude for trial starting at 106000000 is 1.18
Mean noise amplitude for trial starting at 107000000 is 5.89
Mean noise amplitude for trial starting at 108000000 is 0.69
Mean noise amplitude for trial starting at 109000000 is 1.61
Mean noise amplitude for trial starting at 110000000 is 1.51
Mean noise amplitude for trial starting at 111000000 is 3.09
```

```
Mean noise amplitude for trial starting at 112000000 is 9.78
Mean noise amplitude for trial starting at 113000000 is 6.02
Mean noise amplitude for trial starting at 114000000 is 3.91
Mean noise amplitude for trial starting at 115000000 is 6.93
Mean noise amplitude for trial starting at 116000000 is 6.72
Mean noise amplitude for trial starting at 117000000 is 1.53
```

Your answer here

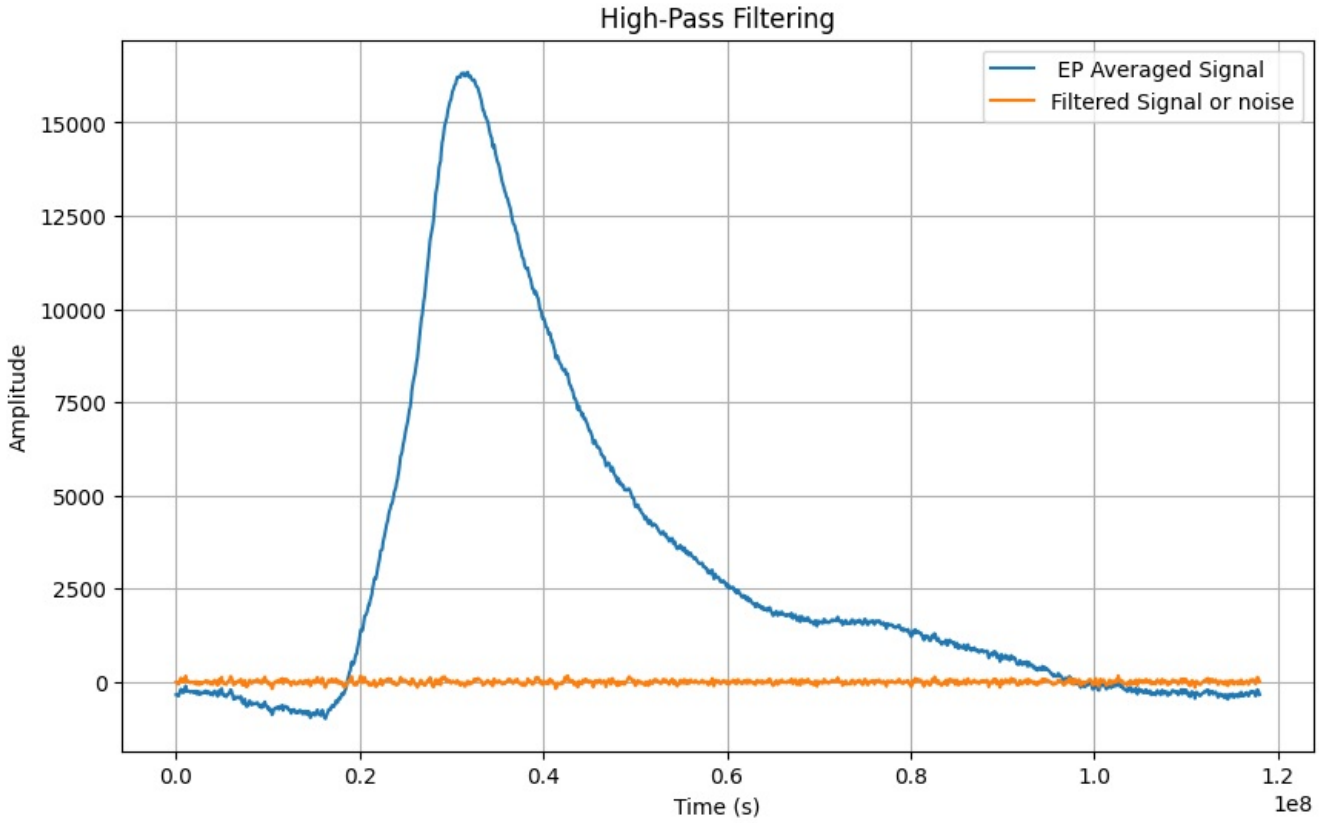
4d

Apply your method on the signal averaged EP and report its noise. (1 pt)

In [153…

```python
#Your code here
dataset = session.open_dataset('I521_A0001_D003')

# Extract EP and Stim channel information
key = list(dataset.ts_details.keys())[0]
channels = dataset.get_channel_indices([key])[0]
start_time = 0
duration = 117999631
data_ep = dataset.get_data(start_time, duration, [channels]).flatten()

# Spike-Triggered Analysis
time_domain = 500000  # 500 ms window
interval = 1e6 / 2713  # Sampling interval
latencies = []
meann = []

for start in range(0, duration, 1000000):

    # Extract a window of data
    ep_data_window = dataset.get_data(start, time_domain, [channels]).flatten()
    meann.append(ep_data_window)

# Compute final latency and average response
average_latency_ms = np.mean(latencies) * 0.001  # Convert to ms
meann = np.array(meann)
mean_waveform = np.mean(meann, axis=0)  # Compute mean waveform
std_dev = np.std(meann, axis=0)  # Compute standard deviation


# Filter parameters
cutoff_freq = 50  # Cutoff frequency in Hz - This is the variable we need to adjust
fs = 2713  # Sampling rate in Hz
order = 4  # Filter order

# Design the high-pass filter
nyq = 0.5 * fs
normal_cutoff = cutoff_freq / nyq
b, a = butter(order, normal_cutoff, btype='highpass')


time_axis = np.linspace(0, duration, len(mean_waveform))
# time_axis = np.arange(len(data_ep)) / fs

# Apply the filter
filtered_data = filtfilt(b, a, mean_waveform)
std_dev = np.std(filtered_data)

print(f"Standard Deviation of High-Frequency Components or amplitude of noise is: {std_dev}")

# Plot the original and filtered data
plt.figure(figsize=(10, 6))

plt.plot(time_axis, mean_waveform, label=' EP Averaged Signal')
plt.plot(time_axis, filtered_data, label='Filtered Signal or noise')

plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('High-Pass Filtering')
plt.legend()
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/numpy/core/fromnumeric.py:3504: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.11/dist-packages/numpy/core/_methods.py:129: RuntimeWarning: invalid value encountered in
scalar divide
  ret = ret.dtype.type(ret / rcount)
Standard Deviation of High-Frequency Components or amplitude of noise is: 52.00052592064035
```



Your answer here

4e

Do these two values make sense? Explain. (1 pt)

Yes, the amplitude of noise in the averaged EP signal is lower than the amplitude of noise in the EP signal.

Your answer here