TECHNISCHE
UNIVERSITÄT
DARMSTADT

Clinical Applications of Brain Imaging, Stimulation, and Modeling

Exercise 4

Talha Bhatti (2667250)

Contents

List of Figures

1 Questions

1.1 Describe the relative advantages and disadvantages of EEG and cryogenic MEG (6 points).

Advantages of EEG:

- High signal magnitude (10 mV, easily detectable).
- Flexibility to move with the subject.
- Captures both tangential and radial dipoles.
- Affordable with 4–256 channels.

Disadvantages of EEG:

- Signal distorted by skull and scalp.
- Lower spatial resolution ($\sim$ 1 cm).

Advantages of MEG:

- High spatial resolution ($<$ 1 cm) and temporal resolution ($\sim$ 1 ms).
- Signal not distorted by skull/scalp.
- Measures primary currents and detects tangential dipoles better.

Disadvantages of MEG:

- Requires magnetic shielding due to weak signals (10 fT).
- Subject must remain stationary.
- Expensive with 275–306 channels.

1.2 What are evoked potentials and stimulus-induced activity, and how do they differ? (6 points)

Evoked Potentials can be described as the time and phase-locked brain responses to a stimulus, making them consistent across trials. They are typically observed as clear peaks in EEG recordings and are analyzed in the time domain.

On the other hand, Stimulus-Induced activity is not phase-locked to the stimulus and varies in timing and phase across different trials. It reflects changes in brain oscillations, such as power increases in specific frequency bands.

Differences:

- EPs are phase-locked, while stimulus-induced activity is not.
- EPs are time-domain signals, while stimulus-induced activity involves oscillations, so they are analyzed in time-frequency domain.

1.3 In EEG, why is it considered more important to have good impedance when recording subjects during task-based experiments (as opposed to resting state)? (4 points)

ERPs captured in task-based EEG recordings are more susceptible to noise, so a good impedance is important to maintain a high SNR for such recordings. Resting state focuses on continuous brain rhythms, which are not as affected by small impedance changes.

1.4 Describe voltage-clamp and current-clamp techniques. (4 points)

Voltage-clamp Technique: The voltage-clamp maintains the voltage at a fixed level and measures the resulting current across the membrane. It allows for studying ion channel activity and understanding how ion currents respond to changes in membrane voltage.
Current-clamp Technique: The current-clamp keeps the current constant and measures the resulting voltage changes. It is used to study how neurons respond to neurotransmitters and the opening of membrane ion channels by observing voltage fluctuations.

1.5 What contributes to linear mixing in EEG/MEG data? What are the problems arising from linear mixing? How can these problems be addressed during data processing and analysis? (10 points)

What contributes to linear mixing in EEG/MEG data?

- Instantaneous field spread - Signals from one source spread instantaneously to multiple sensors.
- Volume conduction - Signals from different brain sources mix due to the conductive properties of brain tissue, skull, and scalp.

What are the problems arising from linear mixing?

- Each source spreads to several sensors.
- Each sensor picks up signals from multiple nearby sources.
- False positives in connectivity estimation between nearby sensors/parcels.
- Particularly severe in resting-state data.
- Affects amplitude correlations but can be mitigated with signal orthogonalization.

How can these problems be addressed?

- **Source reconstruction:** Reduces linear mixing but does not eliminate it.
- **Use appropriate interaction metrics:** Metrics like iPLV discard zero-lag and multiple-of-π-lag interactions (mostly spurious, with some true ones).
- **Task-based comparisons:** Subtract false positives between conditions, though not optimal.

1.6 What steps in recording and preprocessing are required for good source reconstruction in EEG and MEG? (6 points)

1. Recording Stage

- **DAQ:** Collect electrophysiological signals (EEG/MEG) alongside MRI for anatomical information.
- Record behavioral data, demographics, and clinical status for clinical studies.

2. Handling Bad Channels

- Identify and remove bad channels (e.g., too noisy or empty).
- Interpolate missing data from neighboring channels if necessary.

3. Noise and Artifact Removal

- **Line Noise Removal:** Apply narrow band-stop (notch) filters (e.g., 50 Hz in Europe, 60 Hz in the USA).
- **High-Pass Filtering:** Remove slow drifts in data caused by movement (slow means < 0.1 Hz).
- **Artifact Rejection:** Remove major artifacts, including:
  - Eye blinks and movements.
  - Heartbeat.
  - Muscle activity (e.g., neck and jaw).
- **Using ICA (Independent Component Analysis):**
  - Identify and remove artifacts by inspecting spatial distributions and time courses.
  - Record EOG/ECG signals to aid detection, though ICA can often identify eye and heartbeat artifacts without them.

4. Coregistration

- Align EEG/MEG sensor positions with the 3D head model derived from MRI.
- Use software to compute transformation operators between coordinate systems.
- Automatic fitting exists, but manual adjustment may be needed.

5. Source Reconstruction

- Perform source modeling after coregistration to localize brain activity.
- Use cortical parcellations to collapse source time series into meaningful parcels.

1.7 How are local and inter-areal synchronization related to each other? (4 points)

Local synchronization involves locally coordinated activity that produces parallel currents in neurons, resulting in higher amplitude signals. These signals reflect summed neuronal activity, as seen in meso and macroscale recordings.
Inter-areal synchronization refers to phase synchrony between distinct populations. It supports feature binding of related information and enables fast communication between brain regions.
Local synchronization generates coherent activity within a population, which can contribute to inter-areal synchronization between distinct brain regions.

1.8 Which brain states are typically dominated by delta, theta, and alpha rhythm, respectively? (6 points)

- **Delta rhythm (1–4 Hz):** Dominates during deep sleep and is associated with slow-wave sleep states.
- **Theta rhythm (4–8 Hz):** Common during drowsiness, light sleep, and meditative states.
- **Alpha rhythm (8–15 Hz):** Dominates during awake but relaxed states. At rest, it differs between eyes-open and eyes-closed states.

1.9 Discuss if the alpha peak can be diagnosed to diagnose neuro(psychological) disorders. (4 points)

Alpha peak can be affected in many disorders. For example, people with Parkinson's disease tend to have a less pronounced eyes-closed alpha. Weakened alpha peak has also been observed in patients with diseases like epilepsy, Alzheimer's, schizophrenia, and sometimnes mood disorders.

Some studies suggest that across widely distributed brain regions, patients often show a negative correlation between alpha peak amplitudes and depressive scores, while a positive correlation between alpha peak frequencies and depressive scores.

2 Python Exercises

2.1 Task 1

```
import mne
import numpy as np
import matplotlib.pyplot as plt

from numpy import random
from scipy import signal
```

```
#%% Task 1 (4 points)

# Create some oscillation time series as simple sine or cosine waves.
# Use frequencies 6, 10, and 45 Hz.
# Length should be 2 sec and sampling frequency 1000 Hz.
# Plot them individually and their sum

frequencies = [6, 10, 45]
sampling_freq = 1000
duration = 2

t =  np.linspace(0,duration,int(sampling_freq*duration), endpoint=False)
#print(t.shape)
sine_oscillations = [np.sin(2*np.pi*f*t) for f in frequencies]

plt.figure(figsize=(10,6))
for i, (sine_oscillation, f) in enumerate(zip(sine_oscillations,frequencies)):
    plt.subplot(3,1,i+1)
    plt.plot(t, sine_oscillation)
    plt.title(f'{f} Hz Oscillation')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()
plt.tight_layout()
plt.show()

sum_oscillations = sum(sine_oscillations)

plt.figure(figsize=(10, 6))
plt.plot(t, sum_oscillations, label='Sum of Oscillations')
plt.title('Sum of Oscillations (6 Hz, 10 Hz, 45 Hz)')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()
```

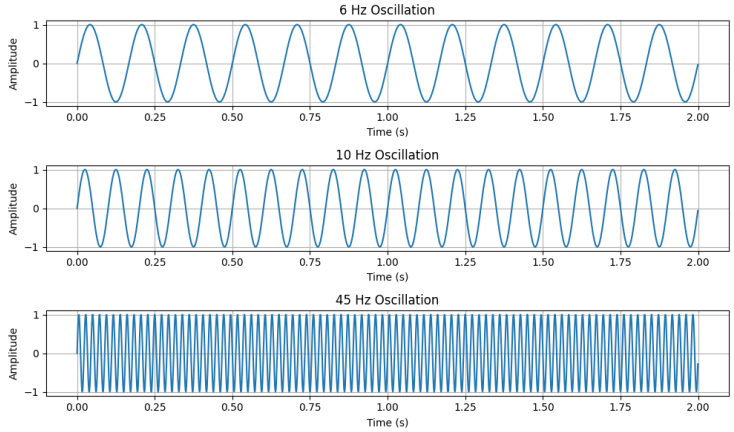

Figure 1: Oscillations

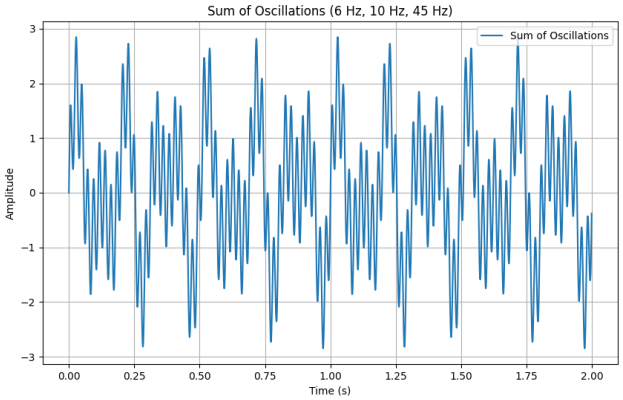

Figure 2: Sum of Oscillations (6, 10, 45 Hz)

2.2 Task 2

```
#%% Task 2 (4 points)

# Now, get the PSD (power spectral density) of the summed signal with the
    Welch method.
# Implementations can be found e.g. in scipy.signal or MNE.
# Play around with the nperseg parameter until you can see all oscillations as
     peaks.
# What do you notice?

#nperseg = 256
#nperseg = 512
nperseg = 1024
freqs, psd = signal.welch(sum_oscillations,sampling_freq,nperseg=nperseg)
plt.semilogy(freqs, psd )
plt.title(f'Power Spectral Density (PSD) using Welch Method (nperseg = {
    nperseg})')
```

```
plt.xlabel('Frequency (Hz)')
plt.ylabel('PSD')
plt.grid()
plt.show()

peaks, _ = signal.find_peaks(psd, height=0)
peak_freqs = freqs[peaks]
print("Frequencies of the peaks (Hz):", peak_freqs)
```

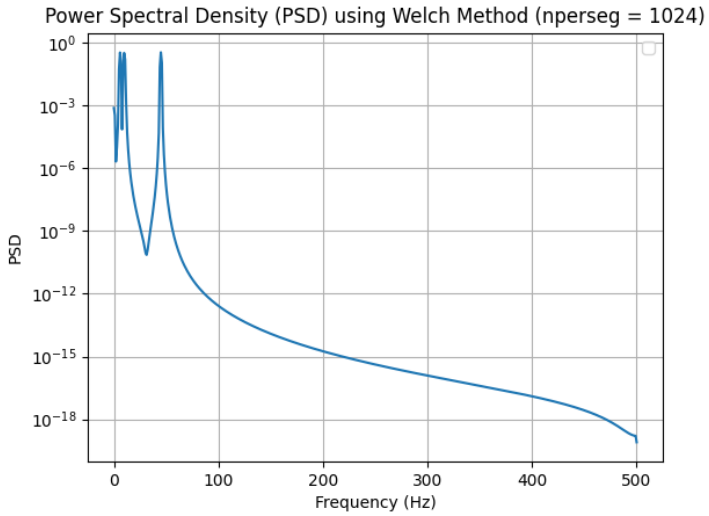

Figure 3: PSD using Welch Method

```
# Output
Frequencies of the peaks (Hz): [ 5.859375  9.765625 44.921875]
```

Better frequency resolution is obtained using higher values of nperseg.
1024 values a decent estimate of the frequencies of the peaks in the PSD.
When I run the code with nperseg = 2000, which is the max allowed value for this signal length, it gives a bunch of additonal peaks.
This is possibly because of thenoise artefacts in the signal at such a high resolution.

2.3 Task 3

```
#%% Task 3 (5 points)

# Now, let's create some white noise (several ways to do this is numpy or
    scipy)
# Again, use f_samp = 1000 Hz and length of 2 sec.
# Plot both the signal and its PSD (computed with Welch).

np.random.seed(42)

white_noise = np.random.normal(0,1, int(duration*sampling_freq))
```

```
plt.figure(figsize=(10, 6))
plt.plot(t, white_noise)
plt.title('White Noise Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.grid()
plt.show()

freqs_noise, psd_noise = signal.welch(white_noise, fs=sampling_freq, nperseg=
    nperseg)

plt.figure(figsize=(10, 6))
plt.semilogy(freqs_noise, psd_noise)
plt.title('PSD of White Noise')
plt.xlabel('Frequency (Hz)')
plt.ylabel('PSD')
plt.grid()
plt.show()
```

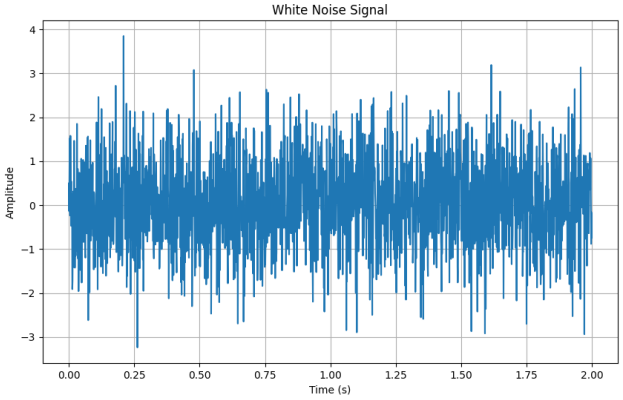

Figure 4: White Noise Signal

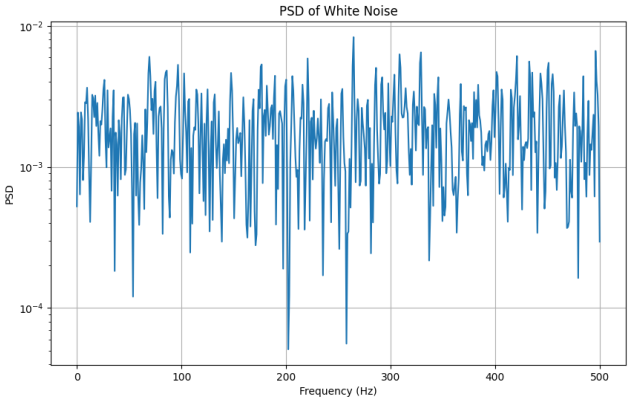

Figure 5: White Noise Signal PSD

2.4 Task 4

```
#%% Task 4 (5 points)

# Create a Morlet wavelet (e.g. with scipy.signal or PyWavelets) at 10 Hz.
# Filter the white noise data by convolution with this wavelet.
# Plot 2 seconds of both real and imaginary part of the filtered time series
    on the same plot.
# Compute and plot the PSD for the real part of the filtered time series.
# What do you notice when you change the width parameter of the wavelet?

center_freq = 10
width = 5
#scaling factor s should be 2 according to the formula f = 2*s*w*r / M in the
    documentation
M = int(duration * sampling_freq)
s = (center_freq * M) / (2 * width * sampling_freq)
print(s)
morlet_wavelet = signal.morlet(M=M, w=width, s = s)

filtered_signal = np.convolve(white_noise, morlet_wavelet, mode='same')

real_part = np.real(filtered_signal)
imag_part = np.imag(filtered_signal)

plt.figure(figsize=(10, 6))
plt.plot(t, real_part, label='Real Part')
plt.plot(t, imag_part, label='Imaginary Part')
plt.title('Filtered Signal (Real and Imaginary Parts)')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()
```

```
freqs_filtered, psd_filtered = signal.welch(real_part, fs=sampling_freq,
    nperseg=nperseg)

plt.figure(figsize=(10, 6))
plt.semilogy(freqs_filtered, psd_filtered)
plt.title('PSD of the Real Part of the Filtered Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('PSD')
plt.grid()
plt.show()

peak_index = np.argmax(psd_filtered)
peak_frequency = freqs_filtered[peak_index]

# Print the peak frequency
print(f"The peak frequency in the PSD is approximately: {peak_frequency:.2f}
    Hz")
```

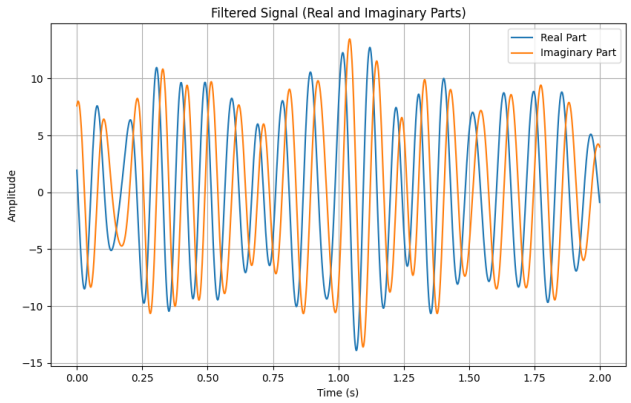

Figure 6: Filtered Signal (Real and Imaginary Parts)

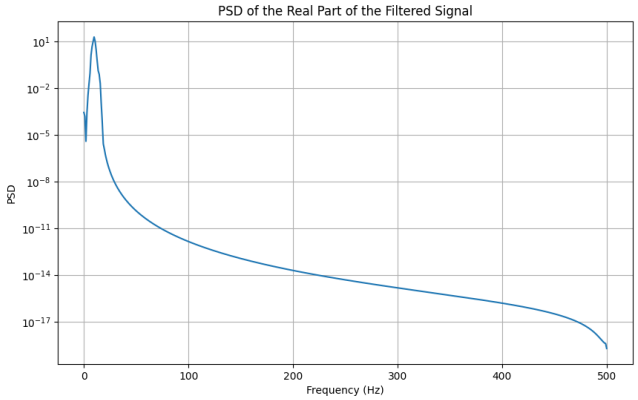

Figure 7: PSD of the Real Part of the Filtered Signal

```
# Output
Changing the width controls the trade-off between time and frequency resolution.
Small width -> less frequency resolution but narrow time domain wavelet
Larger width -> Sharper peak at center frequency but wider wavelet in time domain
```

2.5 Task 5

```python
#%% Task 5 (4 bonus points)

# Let's create a time-frequency plot.
# Create an array of 50 (ideally log-spaced) frequencies from 1 to 30 Hz.
# Create another white noise time series and add it to your artifical 10-Hz
# time series from the previous task.
# Compute the convolution of the summed signal with the array of frequencies.
# (you can do this at once e.g. with signal.cwt)
# You should get a 2D data array.
# Use this to create a time-frequency plot of the time series amplitude.
# Label your axes correctly.
# Describe what you see.

log_spaced_frequencies = np.logspace(np.log10(1), np.log10(30), 50)

np.random.seed(43)
new_white_noise = np.random.normal(0, 1, int(duration * sampling_freq))
summed_signal = real_part + new_white_noise

wavelet_widths = np.round(sampling_freq / (log_spaced_frequencies * 2 * np.pi)
    ).astype(int)
cwt_result = signal.cwt(summed_signal, signal.morlet2, wavelet_widths)
amplitude = np.abs(cwt_result)

plt.figure(figsize=(12, 6))
plt.imshow(
    amplitude,
    aspect='auto',
    extent=[0, duration, log_spaced_frequencies[-1], log_spaced_frequencies
    [0]],
    cmap='viridis'
)
plt.colorbar(label='Amplitude')
plt.title('Time-Frequency Representation of the Summed Signal')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')
plt.gca().invert_yaxis()
plt.show()
```

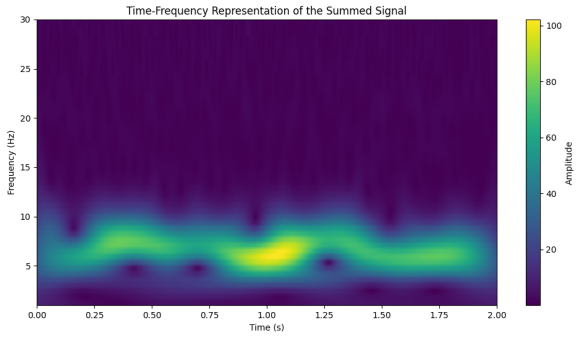

Figure 8: Time-Frequency Representation of the Summed Signal

2.6 Task 6

```
#%% Task 6 (5 points)


# For this following tasks, we will use the MNE library.
# If you have problems installing and/or using MNE library, please let me know
    .
# All the following tasks here can be done using MNE functions.
# Use the online documentation at mne.tools.stable to figure out how to use
    them.
# You can search for functions directly on the website; also googling them
    often works.

# We are gonna look at two EEG data files that were recorded last week at
    Galuske lab from students of this course.
# Download the files from Moodle.
# Load the two raw student data files into variables raw1, raw2 with mne.io.
    read_brainvision()
# Use the preload option.
# Get the sampling frequency and the indices of the channels marked as 'misc'
    from the info metadata.
# Plot the power sepctra in the range to 120 Hz.
# What do you notice regarding peaks, noise?
# How do the two datasets differ?

raw1_path = 'EEG data_1-4/VP01/VP1_BrainImaging.vhdr'
raw2_path = 'EEG data_1-4/VP02/VP2_BrainImaging.vhdr'
raw1 = mne.io.read_raw_brainvision(raw1_path, preload=True, verbose=False)
raw2 = mne.io.read_raw_brainvision(raw2_path, preload=True, verbose=False)
#print(raw1.info)
#print(raw2.info)

sfreq1 = raw1.info['sfreq']
sfreq2 = raw2.info['sfreq']
misc_channels1 = mne.pick_types(raw1.info, misc=True)
misc_channels2 = mne.pick_types(raw2.info, misc=True)
```

```
print(f"Dataset 1: Sampling frequency = {sfreq1} Hz, Misc channels = {
    misc_channels1}")
print(f"Dataset 2: Sampling frequency = {sfreq2} Hz, Misc channels = {
    misc_channels2}")

raw1.compute_psd(fmax=120, verbose=False).plot(show=False)
plt.title('Power Spectra - Dataset 1')
plt.show(block=True)

raw2.compute_psd(fmax=120, verbose=False).plot(show=False)
plt.title('Power Spectra - Dataset 2')
plt.show(block=True)
```

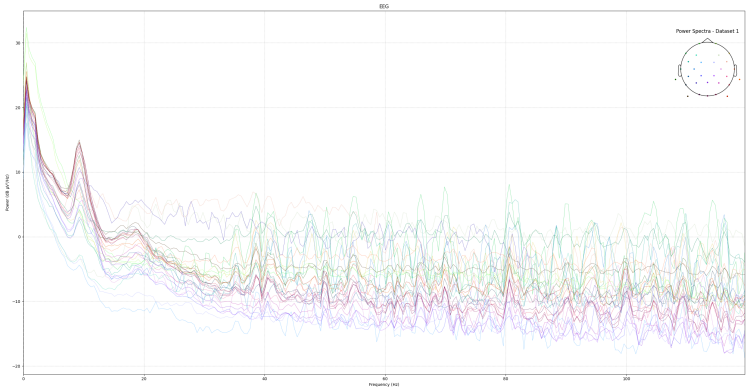

Figure 9: Power Spectra - Dataset 1

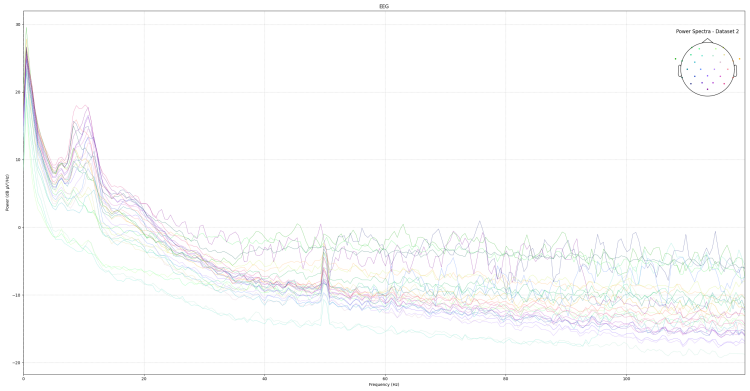

Figure 10: Power Spectra - Dataset 2

```
# Output
Dataset 1: Sampling frequency = 1000.0 Hz, Misc channels = [12 31]
Dataset 2: Sampling frequency = 1000.0 Hz, Misc channels = [27 29 30 31]

Both datasets have clear peaks at lower frequencies and low power at gamma+ freqs.
Dataset 1 has stronger peaks.
Although it shows more noise at higher frequencies than Dataset 2
```

2.7 Task 7

```
# %% Task 7 (2 points)

# For each dataset, plot the sensor locations with names.
raw1.plot_sensors(show_names=True, show=False)
plt.title('Sensor Locations - Dataset 1')
plt.show(block=True)

raw2.plot_sensors(show_names=True, show=False)
plt.title('Sensor Locations - Dataset 2')
plt.show(block=True)
```

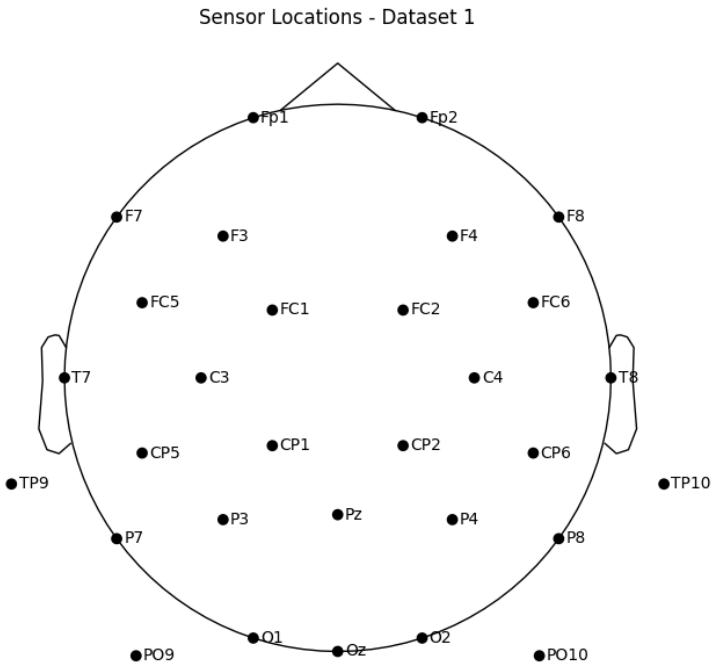

Figure 11: Sensor Locations - Dataset 1

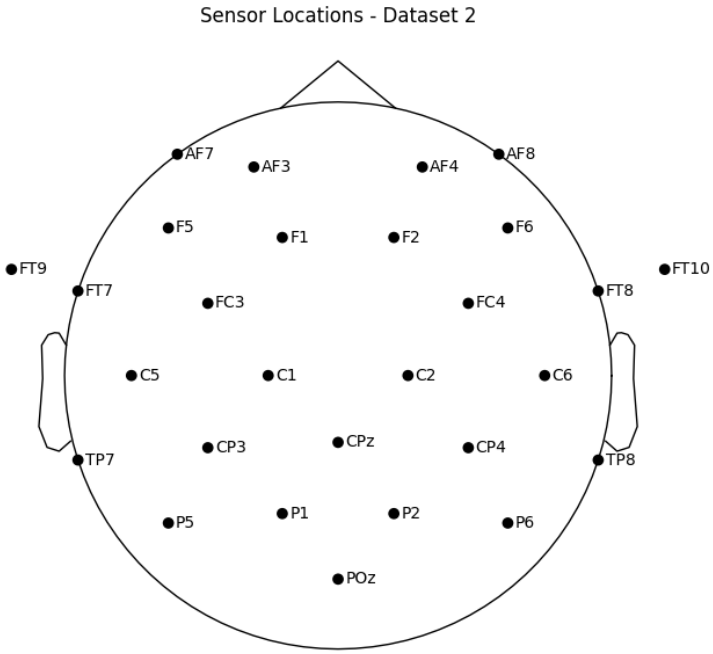

Figure 12: Sensor Locations - Dataset 2

2.8 Task 8

```
# %% Task 8 (4 points)

# Then, use .plot() on your raw objects to open interactive plots.
# If the plot isn't interactive, try changing your matplotlib backend.
# https://matplotlib.org/stable/users/explain/figure/backends.html
# In an interactive plot, you can scroll through the whole time series with
    arrow buttons.
# If you can't get interactive plots to work, you can also specify start and
# duration for static plots with the plot() function.
#
# In which subject can you easily identify periods with eyes open and eyes
    closed?
# Why?
# Make a plot or screenshot of 10 sec eyes-open brain activity, and one of 10
    sec eyes-closed.


raw1.plot(show=False)
plt.show(block=True)

raw2.plot(show=False)
plt.show(block=True)

def plot_alpha_power(raw, channels, alpha_band=(8, 12), window_length=2,
    step_size=0.5, title=None):
```

```
    with mne.utils.use_log_level("error"):
        picks = mne.pick_channels(raw.info['ch_names'], include=channels)

        sfreq = raw.info['sfreq']
        window_samples = int(window_length * sfreq)
        step_samples = int(step_size * sfreq)
        n_samples = raw.n_times
        n_windows = (n_samples - window_samples) // step_samples + 1

        alpha_power = []
        for i in range(n_windows):
            start = i * step_samples
            stop = start + window_samples
            data = raw.get_data(picks=picks, start=start, stop=stop)
            psd, freqs = mne.time_frequency.psd_array_welch(data, sfreq=sfreq,
     fmin=alpha_band[0], fmax=alpha_band[1], n_overlap=0)
            alpha_power.append(psd.mean())

    time_axis = np.arange(0, n_windows) * step_size

    plt.figure(figsize=(10, 4))
    plt.plot(time_axis, alpha_power, label=f'Alpha Power ({alpha_band[0]}   {
    alpha_band[1]} Hz)')
    plt.title(title)
    plt.xlabel('Time (s)')
    plt.ylabel('Power')
    plt.legend()
    plt.grid()
    plt.show()

alpha_band = (8, 12)

occipital_channels = ['O1', 'O2', 'Oz']
plot_alpha_power(raw1, channels=occipital_channels, alpha_band=alpha_band,
    title = 'Alpha Power Spectra - Subject 1')

parietal_channels = ['P5', 'P1', 'P2', 'P6', 'POz']
plot_alpha_power(raw2, channels=parietal_channels, alpha_band=alpha_band,
    title = 'Alpha Power Spectra - Subject 2')


def plot_10s_segment(raw, start_time, title):
    raw.plot(start=start_time, duration=10, title=title, show=False)
    plt.show(block=True)

eye_open = 100
eye_closed = 500

#subject 1
plot_10s_segment(raw1, eye_open, title='Eyes-Open Brain Activity - Subject 1')
plot_10s_segment(raw1, eye_closed, title='Eyes-Closed Brain Activity - Subject
     1')
#subject 2
plot_10s_segment(raw2, eye_open, title='Eyes-Open Brain Activity - Subject 2')
```

```
plot_10s_segment(raw2, eye_closed, title='Eyes-Closed Brain Activity - Subject
    2')
```

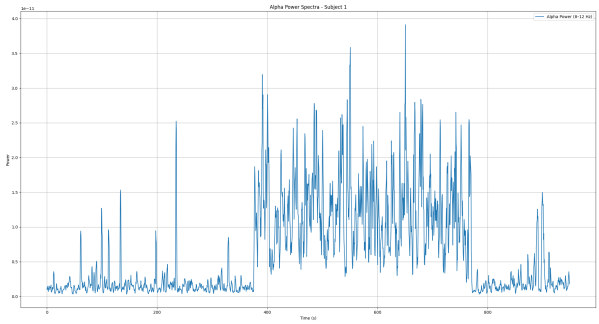

Figure 13: Alpha Power Spectra - Subject 1

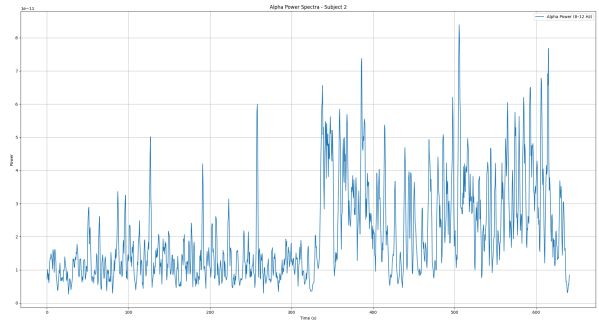

Figure 14: Alpha Power Spectra - Subject 2

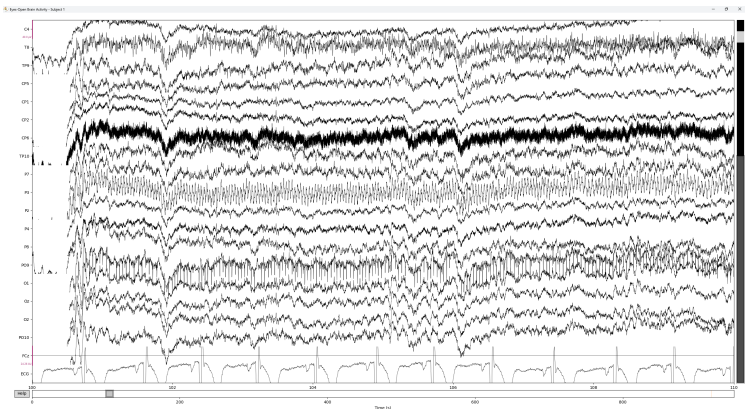

Figure 15: Eyes-Open Brain Activity - Subject 1

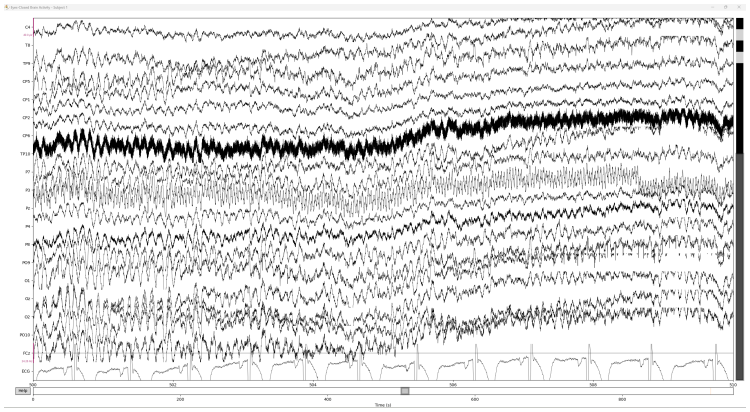

Figure 16: Eyes-Closed Brain Activity - Subject 1

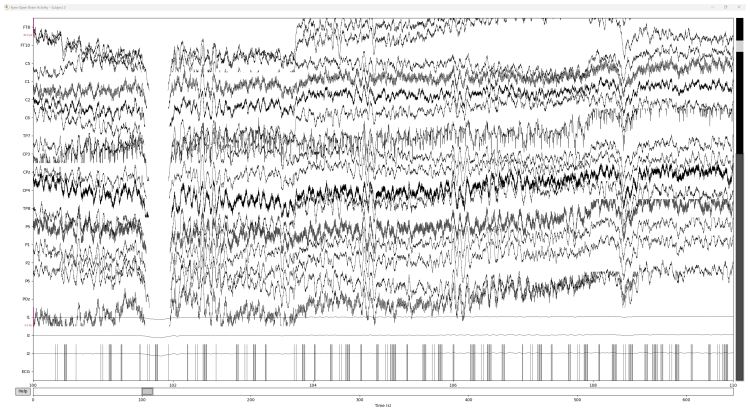

Figure 17: Eyes-Open Brain Activity - Subject 2

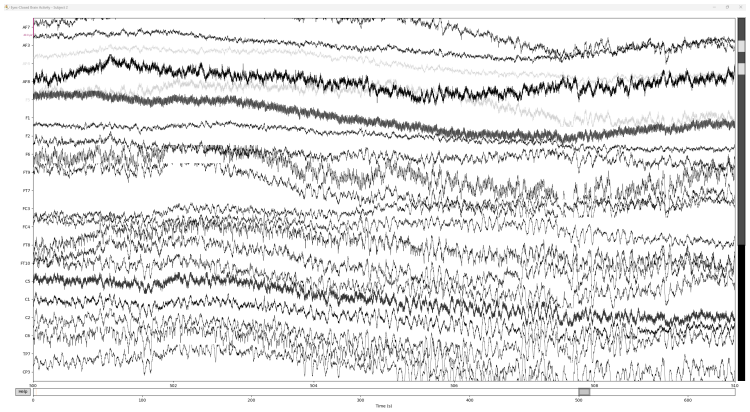

Figure 18: Eyes-Closed Brain Activity - Subject 2

```
# Output
In the interactive plots, looking at the occipital (subject 1) and parietal (subject 2)
channels helps identify eye open and eye closed activity.
I believe it is rather easier to identify in the first subject
```

due to the presence of occipital channels.
I have made the a function to plot power spectra of both subjects for alpha frequencies.
Higher activity in alpha freq means eye-closed period.
For subject 1: 400-800 seconds is eye closed period.
For subject 2: 350-600 seconds seem to be the eye closed period.

2.9 Task 9

```
# %% Task 9 (4 points)

# Make copies of the raw data objects.
# To these copies, apply a notch filter to remove line noise,
# and bandpass-filter them in the range 1 to 100 Hz.
# Plot the power spectra of the filtered data and describe what has changed.
def filter_psd(raw, notch_freq=50, bandpass_range=(1, 100), title=None):
    raw_filtered = raw.copy()
    raw_filtered.notch_filter(freqs=notch_freq, verbose=False)
    raw_filtered.filter(l_freq=bandpass_range[0], h_freq=bandpass_range[1],
    verbose=False)
    psd = raw_filtered.compute_psd(fmax=100, verbose=False)
    psd.plot(show=False)
    plt.title(title)
    plt.tight_layout()
    plt.show(block=True)
    return raw_filtered

raw1_filtered = filter_psd(raw1, title="Power Spectrum - Filtered Subject 1")

raw2_filtered = filter_psd(raw2, title="Power Spectrum - Filtered Subject 2")
```

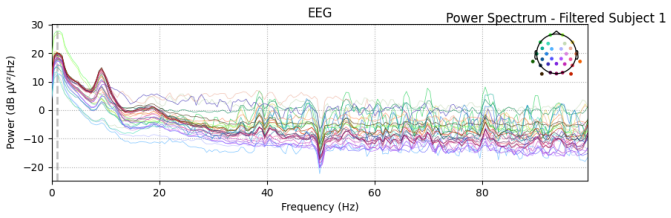

Figure 19: Power Spectrum - Filtered Subject 1

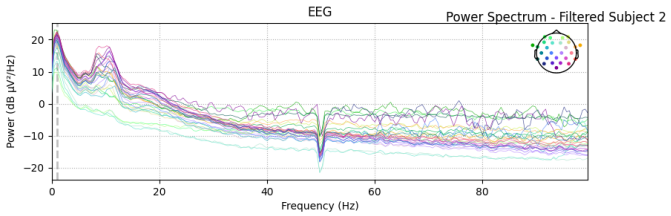

Figure 20: Power Spectrum - Filtered Subject 2

2.10 Task 10

```
# %% Task 10 (3 points)

# Plot and inspect the filtered time series. What has changed?
raw1_filtered.plot(start=100,duration=10,title='Filtered Time series - Subject
    1', show = False)
plt.show(block=True)
raw2_filtered.plot(start=100,duration=10,title='Filtered Time series - Subject
    2', show = False)
plt.show(block=True)
```

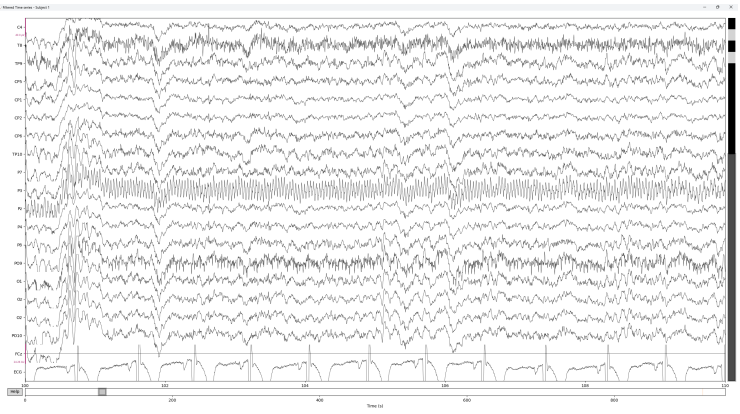

Figure 21: Filtered Time series - Subject 1

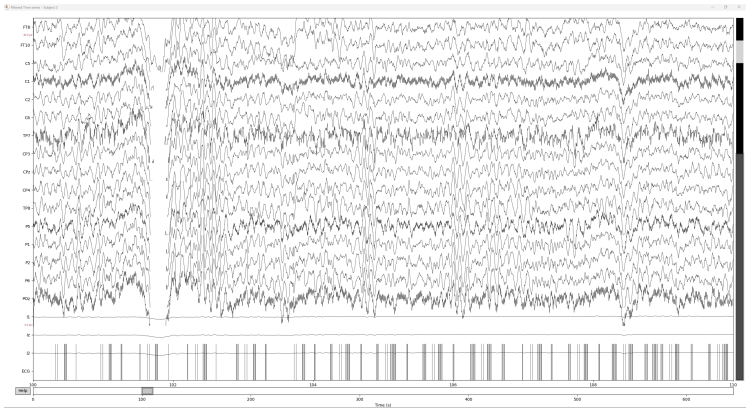

Figure 22: Filtered Time series - Subject 2

Output
With the removal of the line noise, the sharp oscillations around 50 Hz have been removed in th
Slow frequency drift seems to be removed and filtered series shows smoother waveform due to the
Neural oscillations are not only intact but seem rather clearer.

2.11 Task 11

```
# %% Task 11 (4 bonus points)

# Identify for one subject periods in the data when they had their eyes closed
    and open.
# Plot power spectra for these periods.
# What difference is observable in the power spectra?
# What is this effect called?

sfreq = raw1.info['sfreq']

eyes_closed_data = raw1.copy().crop(tmin=500, tmax=510).get_data()
psd_closed, freqs_closed = mne.time_frequency.psd_array_welch(eyes_closed_data
    ,sfreq, fmax=50, verbose=False)

eyes_open_data = raw1.copy().crop(tmin=100, tmax=110).get_data()
psd_open, freqs_open = mne.time_frequency.psd_array_welch(eyes_open_data,sfreq
    , fmax=50, verbose=False)

psd_closed_mean = psd_closed.mean(axis=0)
psd_open_mean = psd_open.mean(axis=0)

plt.figure(figsize=(10, 6))
plt.semilogy(freqs_closed, psd_closed_mean, label='Eyes Closed', alpha=0.8)
plt.semilogy(freqs_open, psd_open_mean, label='Eyes Open', alpha=0.8)
plt.title('Power Spectra - Eyes Closed vs Eyes Open')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power Spectral Density')
plt.legend()
plt.grid()
plt.show()
```

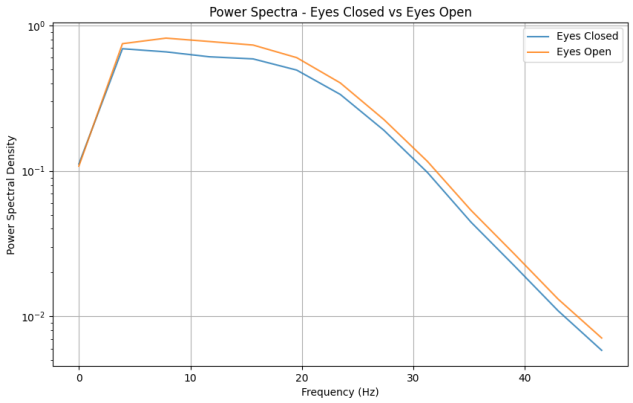

Figure 23: Power Spectra - Eyes Closed vs Eyes Open

```
# Output
For eyes closed, we can see a strong peak at 8 Hz (in Alpha range 8-12 Hz).
For eyes open, we can see some reduction in the Alpha power.
This is known as Alpha Blocking, where visual stimuli reduce alpha rhythm while eyes are open.
```

2.12 Task 12

```python
# %% Task 12 (4 points)

# Now let's look at some typical artefacts in EEG (or MEG) data.
# For the first subject, plot ECG and EOG events as described on
# https://mne.tools/stable/auto_tutorials/preprocessing/40
    _artifact_correction_ica.html
# Where are these artifacts mostly located?

raw1_filtered = mne.set_bipolar_reference(
    raw1_filtered, anode='Fp1', cathode='Fp2', ch_name='EOG_simulated'
)

ecg_epochs = mne.preprocessing.create_ecg_epochs(raw1_filtered, ch_name='ECG')
ecg_evoked = ecg_epochs.average()
ecg_evoked.apply_baseline(baseline=(None, -0.2))
ecg_evoked.plot_joint(title="ECG Artifacts")

eog_epochs = mne.preprocessing.create_eog_epochs(raw1_filtered, ch_name='
    EOG_simulated')
eog_evoked = eog_epochs.average()
eog_evoked.apply_baseline(baseline=(None, -0.2))
eog_evoked.plot_joint(title="Simulated EOG Artifacts")
```

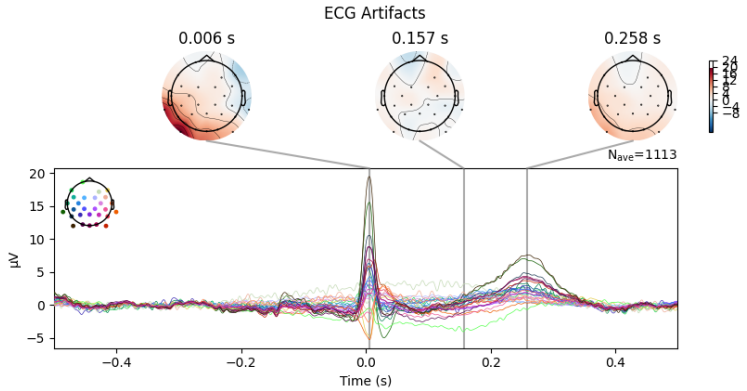

Figure 24: ECG Artifacts

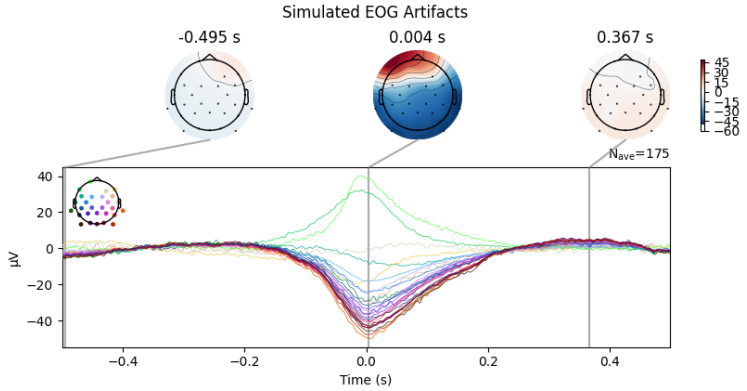

Figure 25: Simulated EOG Artifacts

```
# Output
The EOG artifacts from eye blinking are present in the frontal regions.
The ECG artifacts, mainly from heartbeat, are prominent in the central
and posterior regions, as seen in the topomaps at 0.006s and 0.157s.
```

2.13 Task 13

```python
# %% Task 13 (10 points)

# Following the instruction on the same tutorial page,
# compute ICA with 16 components and fit to the filtered data of subject 1
# This may take a while depending on your computer.
# Plot the components and their source time series (but applied to the
    filtered data)
# Which components look obviously like they represent artefacts, and what kind
    ?
# Exclude the artifactual components and apply the ICA to a copy of the raw
    data.
# Did the ICA do what it was supposed to?

ica = mne.preprocessing.ICA(n_components=16, max_iter="auto", random_state=97)
```

```
ica.fit(raw1_filtered)

ica.plot_components(title="ICA Components (Scalp Maps)")  #visualize scalp
    maps
ica.plot_sources(raw1_filtered, show_scrollbars=False, title="ICA Component
    Time Series", show=False)  #time series
plt.show(block=True)

eog_inds, eog_scores = ica.find_bads_eog(raw1_filtered, ch_name='EOG_simulated
    ')
print(f"EOG-related components: {eog_inds}")

ecg_inds, ecg_scores = ica.find_bads_ecg(raw1_filtered, ch_name='ECG')
print(f"ECG-related components: {ecg_inds}")

ica.exclude = eog_inds + ecg_inds
print(f"Components marked for exclusion: {ica.exclude}")

reconst_raw1 = raw1_filtered.copy()
ica.apply(reconst_raw1)

raw1_filtered.plot(title="Original Data", show=False)
plt.show(block=True)

reconst_raw1.plot(title="Reconstructed Data After ICA", show=False)
plt.show(block=True)
```

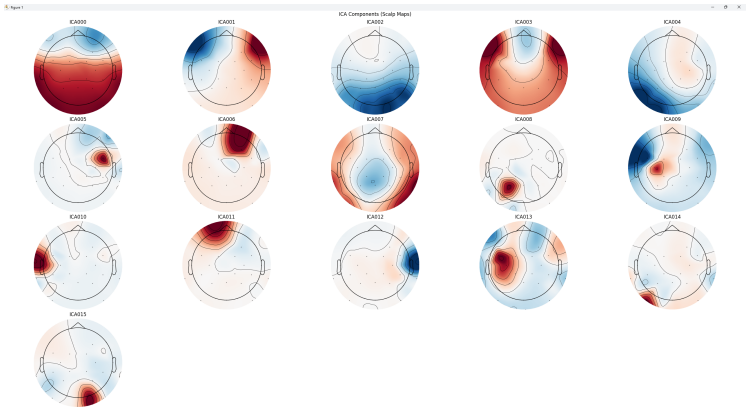

Figure 26: ICA Components (Scalp Maps)

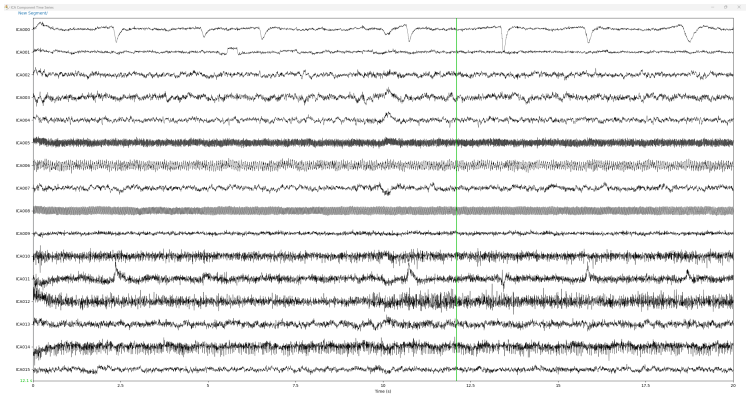

Figure 27: ICA Component Time Series

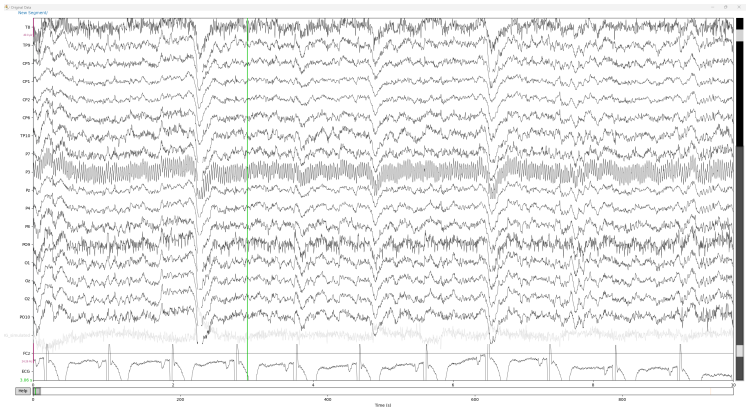

Figure 28: Original Data

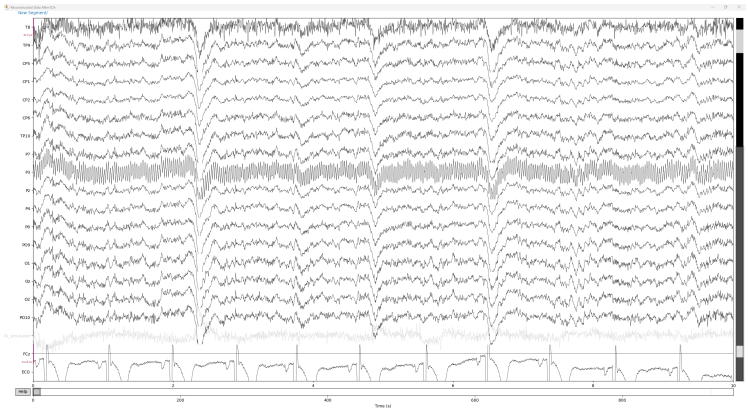

Figure 29: Reconstructed Data After ICA

```
# Output
ICA000 seems to have a large ECG artefact.
Main EOG related artefact is simulated in ICA001 due to eye blink.
ICA003, 005, 013, 014 are major ECG related artefact,
```

```
likely due to heartbeat or some muscle artefact.
After reconstruction, we can see ICA has reduced eye blink artefacts
and also minimized some ECG related artefacts.
```