TECHNISCHE
UNIVERSITÄT
DARMSTADT

Clinical Applications of Brain Imaging, Stimulation, and Modeling

Exercise 5

Talha Bhatti (2667250)

Contents

List of Figures

1 Questions

1.1 What is the analytic signal and why is it important in the analysis of electrophysiological data? (5 points)

- The analytic signal is a complex extension of a real-valued signal, obtained via the Hilbert transform or wavelet convolution.
- It provides both amplitude and phase information, crucial for assessing neuronal oscillations.
- Essential for analyzing phase synchronization and phase-amplitude coupling.
- No negative frequency components, making it useful for EEG/MEG time-frequency analysis.
- Helps understand rhythmic neuronal interactions and connectivity.

1.2 Describe two ways to obtain a narrowband analytical signal. (5 points)

- **Bandpass filtering + Hilbert transform:** First filter the signal in a specific frequency band, then apply the Hilbert transform to get the analytic signal.
- **Wavelet convolution (e.g Morlet wavelets):** Convolute the signal with a wavelet function to extract phase and amplitude within a specific frequency range.

1.3 Which type of noise in electrophysiological recordings varies with the country in which the recording was carried out? (3 points)

Line noise from electrical power grids. In Europe, it is 50 Hz while in the US and some other regions it is 60 Hz. It can be removed using a notch filter.

1.4 Which type of 'noise' can be removed with a high-pass filter? (2 points)

"Slow drifts" in the signal can be removed by using a highpass filter. These may include muscle, movement artifacts.

1.5 What are the two major applications of independent component analysis in neuroimaging? (5 points)

- **Artefact removal:** Identifies and removes eye blinks, muscle activity and heartbeat noise from EEG/MEG.
- **Source separation:** Decomposes mixed signals into statistically independent components which are useful for detecting brain networks.

1.6 What has been suggested to be the relationship between phase synchrony and amplitude coupling? Does this relationship vary between frequency bands? (5 points)

- Phase synchrony and amplitude coupling are two distinct mechanisms for long range neuronal communication.
- Phase synchrony enables precise timing of neural interactions, while amplitude coupling operates over longer timescales.
- The relationship varies by frequency, meaning lower frequencies (delta/theta) show stronger amplitude coupling while higher frequencies (beta/gamma) favor phase synchrony.

1.7 How could analysis of phase synchrony be used with epileptic patients? (5 points)

Seizures are characterized by hypersynchrony of neuronal populations. So we can observe increased phase synchrony in the seizure-onset zone before a seizure starts. Synchrony mapping helps identify the epileptogenic zone (EZ) for surgical planning and it can be used to monitor disease progression or predict seizures.

1.8 Which are the two main types of cross-frequency coupling that have been studied and how are they thought to "connect" oscillations of different frequencies? (5 points)

- **Phase-amplitude coupling (PAC):** The phase of a slow oscillation (e.g theta) modulates the amplitude of a faster one (e.g gamma).
- **Cross-frequency phase synchrony (CFS):** Two oscillations at different frequencies maintain a consistent phase relationship (e.g theta-gamma coupling).

They are thought to integrate processing across spatial and temporal scales.

1.9 What is power-law scaling and how has it been observed in the brain? (5 points)

- Power-law scaling means that brain signals follow a $1/f$ frequency distribution, where lower frequencies have higher power.
- Seen in EEG/MEG power spectra, where activity does not have a single dominant frequency but instead follows a continuous spectrum.
- Suggests self-organized criticality and efficient information processing.

1.10 Which is the range of DFA exponents typically observed in neuronal time series and what are the characteristics of processes with exponents in this range? (5 points)

DFA exponents in brain signals typically range between 0.5 – 1.5.

- 0.5 – 1.0: Correlated noise, scale-free fluctuations.
- >1.0: Long-range temporal correlations, possibly indicating criticality in brain dynamics.

1.11 What are some of the proposed benefits of the brain operating near a critical phase transition? (5 points)

- Maximized information transfer and efficient communication across brain regions
- Optimal balance between stability and flexibility, allowing adaptive responses to stimuli
- Enhanced computational capacity, improving learning and memory
- Increased sensitivity to stimuli, beneficial for perception and cognition
- More diverse network configurations, enabling complex neural dynamics

2 Python Exercises

2.1 Task 1

```
#%% Import modules and functions

from fooof import FOOOF, FOOOFGroup
from fooof import Bands
from fooof.analysis import get_band_peak_fg, get_band_peak_fm
from mne.preprocessing import ICA
from mne.time_frequency import tfr_array_morlet

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

plt.style.use('default')

import sys
sys.path.append('L:/nttk-data3/palva/Felix/Python37/__exercises/ex05/')
sys.path.append('L:/nttk-data3/palva/Felix/Python37/Utilities/')

from cross_functions import cplv, dfa, get_dfa_parameters

# Define canonical frequency bands
bands = Bands({'theta' : [4, 8],
```

```
                'alpha' : [8, 15],
                'beta' : [15, 30]})



#%% Task 1 (5 points)

# This time, we're gonna look at all 4 subject's EEG datasets.
# Load these datasets with MNE.
# Inspect the timeseries of each dataset and find the approximate time points
    where the subjects close their eyes.
# Report these time points.

raw1_path = 'EEG data_1-4/VP01/VP1_BrainImaging.vhdr'
raw2_path = 'EEG data_1-4/VP02/VP2_BrainImaging.vhdr'
raw3_path = 'EEG data_1-4/VP03/VP3_BrainImaging.vhdr'
raw4_path = 'EEG data_1-4/VP04/VP4_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)
raw3 = mne.io.read_raw_brainvision(raw3_path, preload=True, verbose=False)
raw4 = mne.io.read_raw_brainvision(raw4_path, preload=True, verbose=False)

#raw1.plot(block=True)
#raw2.plot(block=True)
#raw3.plot(block=True)
#raw4.plot(block=True)

print('Subject 1 closes eyes after 6-minutes (aprrox. 360 seconds mark). That
    is when the eye-blinking artefacts diminish from the frontal channels and
    alpha peaks dominate in the occipital channels')
print('Subject 2 shows the similar trend, however it seems eyes close around
    the 330 seconds mark so at around 5 and half minutes')
print('Subject 3 also closes eyes around 360 seconds, which is at 6-min mark')
print('Subject 4 closes eyes at around 5-min mark at 310 seconds')
```

```
Subject 1 closes eyes after 6-minutes (aprrox. 360 seconds mark).
That is when the eye-blinking artefacts diminish from the frontal
channels and alpha peaks dominate in the occipital channels
Subject 2 shows the similar trend, however it seems eyes close around
the 330 seconds mark so at around 5 and half minutes
Subject 3 also closes eyes around 360 seconds, which is at 6-min mark
Subject 4 closes eyes at around 5-min mark at 310 seconds
```

2.2 Task 2

```
#%% Task 2 (10 points)

# As with exercise 4, let's preprocess the data.
# Create copies of the raw datasets, keeping only EEG channels.
# Apply notch filters for line noise and bandpass (1-100 Hz).
```

```
# Execute ICA for each subject on the copied data.
# Identify bad components and do ICA reconstruction from raw data with bad
    components excluded.
# To the ICA-reconstructed data, apply notch and bandpass filters as
    previously.
# Report which components you excluded and what kind of artifact(s) they
    likely represent.


raw1_filtered = raw1.copy().notch_filter(50).filter(l_freq=1, h_freq=100)
raw2_filtered = raw2.copy().notch_filter(50).filter(l_freq=1, h_freq=100)
raw3_filtered = raw3.copy().notch_filter(50).filter(l_freq=1, h_freq=100)
raw4_filtered = raw4.copy().notch_filter(50).filter(l_freq=1, h_freq=100)

fig1n = raw1_filtered.compute_psd(fmax=120).plot()
fig1n.suptitle('Power Spectra - Subject 1')
fig2n = raw2_filtered.compute_psd(fmax=120).plot()
fig2n.suptitle('Power Spectra - Subject 2')
fig3n = raw3_filtered.compute_psd(fmax=120).plot()
fig3n.suptitle('Power Spectra - Subject 3')
fig4n = raw4_filtered.compute_psd(fmax=120).plot()
fig4n.suptitle('Power Spectra - Subject 4')


ica = mne.preprocessing.ICA(n_components=16, max_iter="auto", random_state=97)
def perform_ica(raw_filtered, raw, bad_components):
    ica.fit(raw_filtered)
    ica
    ica.plot_sources(raw_filtered, show_scrollbars=False)
    ica.plot_components()
    ica.exclude=bad_components
    reconst_raw = raw.copy()
    ica.apply(reconst_raw)
    reconst_raw
    raw.plot(highpass=1,lowpass=99, block=True)
    reconst_raw.plot(highpass=1,lowpass=99, block=True)
    return  reconst_raw

exclude_dict = {
    "raw1": [0, 1], #blinking, eye movement
    "raw2": [0, 2], #blinking, eye movement
    "raw3": [0, 1], #blinking, eye movement
    "raw4": [0], #blinking
}
reconst_raw1 = perform_ica(raw1_filtered, raw1, exclude_dict["raw1"])
reconst_raw2 = perform_ica(raw2_filtered, raw2, exclude_dict["raw2"])
reconst_raw3 = perform_ica(raw3_filtered, raw3, exclude_dict["raw3"])
reconst_raw4 = perform_ica(raw4_filtered, raw4, exclude_dict["raw4"])

print('Each subject shows blinking artefact in ICA000 component. Also the eye
    movement artefact, ICA001 for subject 1 and 3, ICA002 for subject 2 are
    removed. Subject 4 does not show prominent eye movement')
```

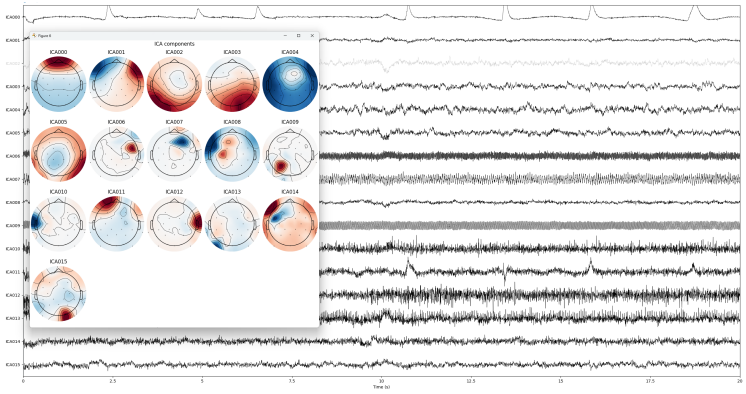

Figure 1: ICA Components for Subject 1

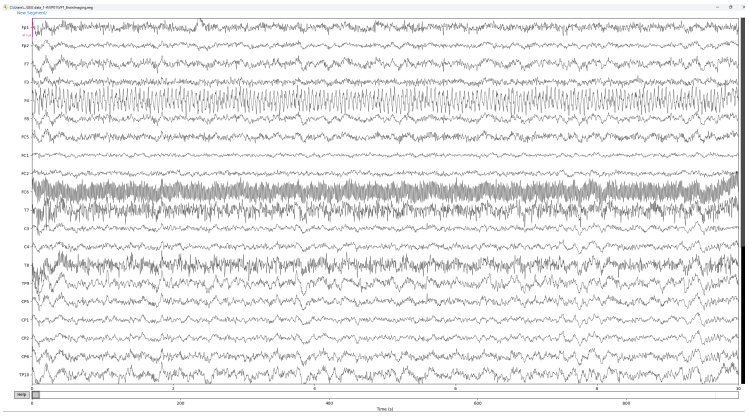

Figure 2: Reconstructed Time Series for Subject 1

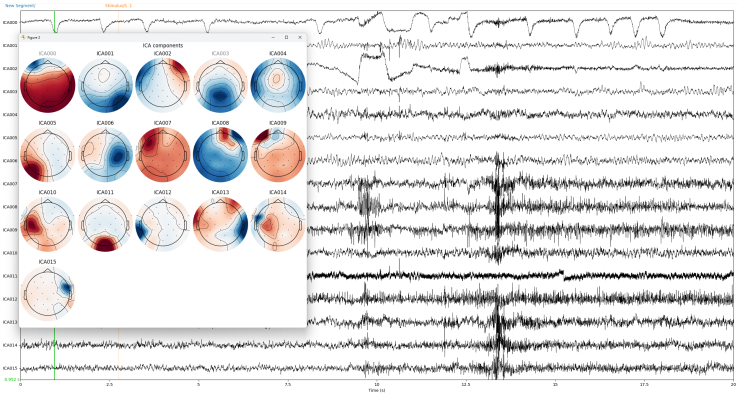

Figure 3: ICA Components for Subject 2

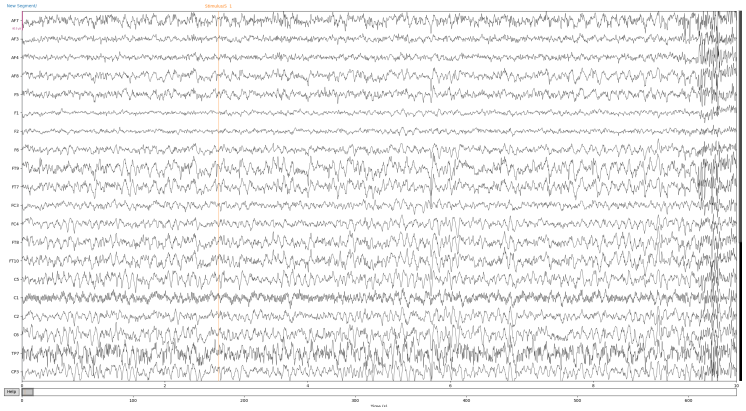

Figure 4: Reconstructed Time Series for Subject 2

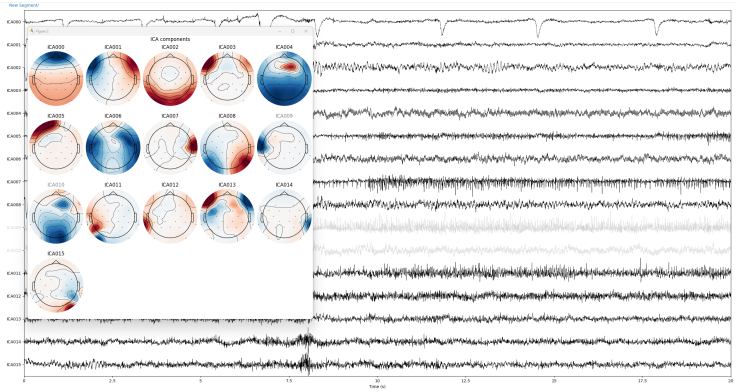

Figure 5: ICA Components for Subject 3

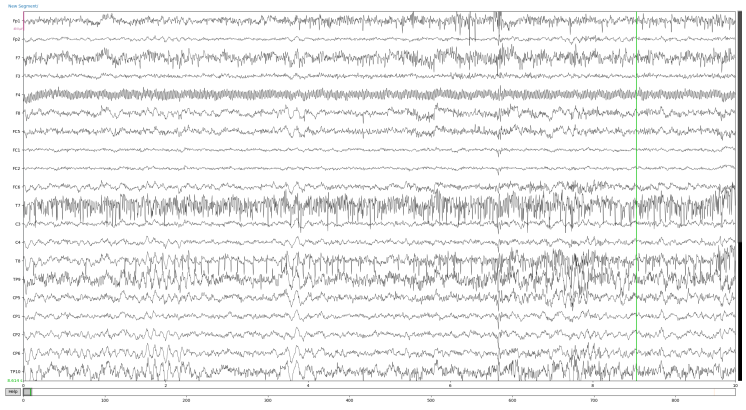

Figure 6: Reconstructed Time Series for Subject 3

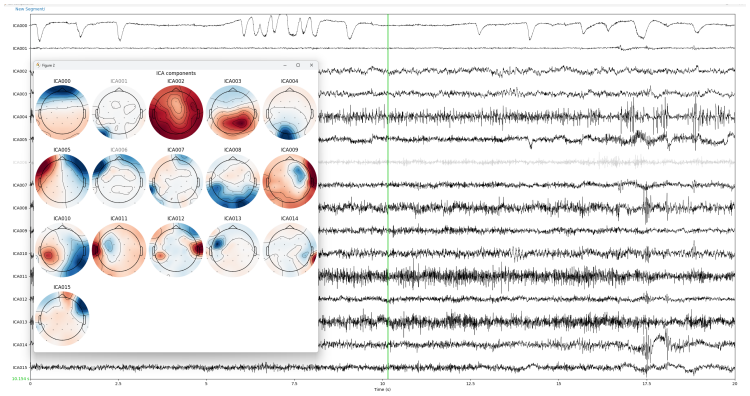

Figure 7: ICA Components for Subject 4

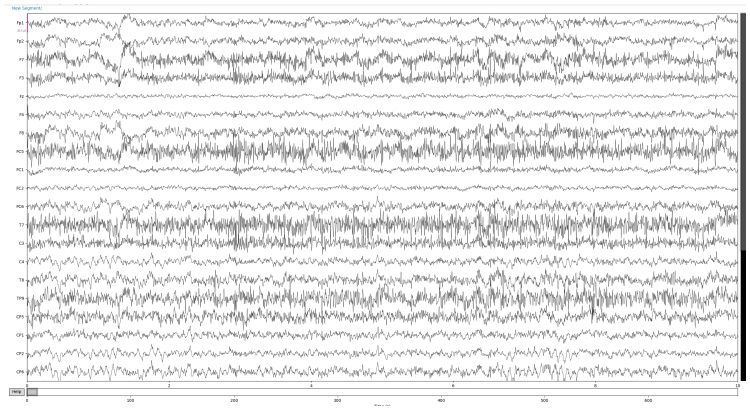

Figure 8: Reconstructed Time Series for Subject 4

Each subject shows blinking artefact in ICA000 component.
Also the eye movement artefact, ICA001 for subject 1 and 3,
ICA002 for subject 2 are removed.
Subject 4 does not show prominent eye movement.

2.3 Task 3

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

# For each subject, compute the power spectra in:
# a 4-minute eyes-open time period and
# a 4-minute eyes-closed time period.
# The compute_psd() function returns a 'Power Spectrum' object which contains
    two attributes that we will need, "freqs" and "data".
# The frequencies should be the same in all spectra, so we'll only need to
    save this once.
# Save the data for each spectrum.

# For each subject, see if you can identify from the spectra channels that
    seem to contain noise in the beta - gamma bands and remove them from the
```

```
    spectra.
# If you remove channels for a subject, you should remove the same channels
    from both eyes-open and -closed spectra.
# If you cannot solve this part of the task, continue with the spectra as they
     are.

# It is recommended to use lists and for loops here, to keep the code short
    and easy to read.

subjects = {
    'Subject1': {
        'raw': reconst_raw1,
        'open': (0, 240),
        'closed': (400, 640)
    },
    'Subject2': {
        'raw': reconst_raw2,
        'open': (0, 240),
        'closed': (360, 600)
    },
    'Subject3': {
        'raw': reconst_raw3,
        'open': (0, 240),
        'closed': (400, 640)
    },
    'Subject4': {
        'raw': reconst_raw4,
        'open': (0, 240),
        'closed': (360, 600)
    },
}

fmin_bg, fmax_bg = 15, 50
threshold_factor = 2.0

# Dictionary to store final cleaned PSDs for each subject & condition
psd_clean_dict = {}

for subj_name, info in subjects.items():
    open_start, open_end = info['open']
    closed_start, closed_end = info['closed']

    raw_open = info['raw'].copy().crop(tmin=open_start, tmax=open_end)
    raw_closed = info['raw'].copy().crop(tmin=closed_start, tmax=closed_end)

    psd_open = raw_open.compute_psd(fmin=1, fmax=100)
    psd_closed = raw_closed.compute_psd(fmin=1, fmax=100)

    # Mean power in 15  50   Hz range
    data_open_bg = psd_open.get_data(fmin=fmin_bg, fmax=fmax_bg)
    mean_open = data_open_bg.mean(axis=1)
    data_closed_bg = psd_closed.get_data(fmin=fmin_bg, fmax=fmax_bg)
    mean_closed = data_closed_bg.mean(axis=1)
```

```
    # Threshold-based detection
    thresh_open = threshold_factor * mean_open.mean()
    thresh_closed = threshold_factor * mean_closed.mean()
    idx_open = np.where(mean_open > thresh_open)[0]
    idx_closed = np.where(mean_closed > thresh_closed)[0]

    all_noisy_idx = set(idx_open.tolist() + idx_closed.tolist())
    noisy_channels = [psd_open.ch_names[i] for i in all_noisy_idx]

    if noisy_channels:
        print(f"{subj_name} - Noisy channels: {noisy_channels}")
    else:
        print(f"{subj_name} - No obviously noisy channels.")

    psd_open_clean = psd_open.drop_channels(noisy_channels)
    psd_closed_clean = psd_closed.drop_channels(noisy_channels)

    # Store cleaned PSDs
    psd_clean_dict[subj_name] = {
        'open': psd_open_clean,
        'closed': psd_closed_clean
    }

    fig1 = psd_open.plot()
    fig1.suptitle(f"{subj_name} - Eyes Open (Original)")
    fig2 = psd_closed.plot()
    fig2.suptitle(f"{subj_name} - Eyes Closed (Original)")
    fig3 = psd_open_clean.plot()
    fig3.suptitle(f"{subj_name} - Eyes Open (Cleaned)")
    fig4 = psd_closed_clean.plot()
    fig4.suptitle(f"{subj_name} - Eyes Closed (Cleaned)")
    plt.show()
```

```
Subject1 - Noisy channels: ['FC2', 'F3', 'P7']
Subject2 - Noisy channels: ['POz', 'TP7', 'P6']
Subject3 - Noisy channels: ['P8', 'FC6', 'F3', 'T8']
Subject4 - Noisy channels: ['O1', 'Oz']
```

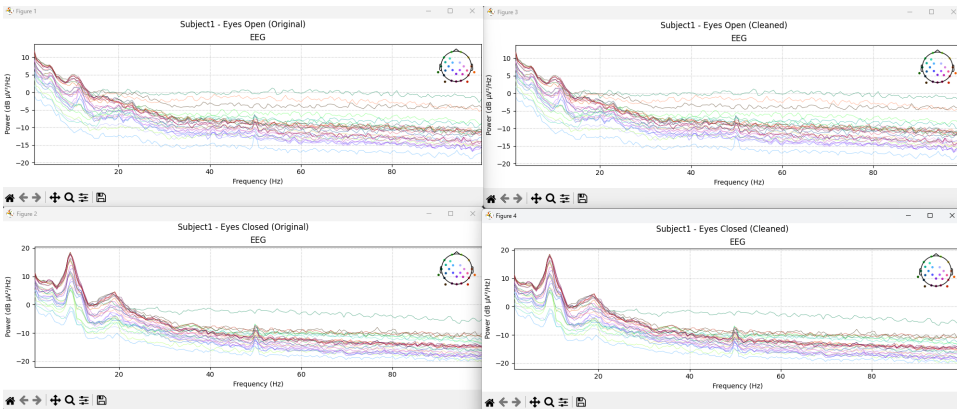

Figure 9: Original and Cleaned Power Spectra of Subject 1

2.4 Task 4

```
#%% Task 4 (10 points)

# For each subject, compute the mean over all channels of the PSD data, both
    for the eyes-open and closed data.

# Now, we'll use the package FOOOF to fit peaks and 1/f background "aperiodic"
     activity to the mean spectra.
# Description and documentation can be found at https://fooof-tools.github.io/
    fooof/index.html.
# Initialize an instance of FOOOF with the parameters.
# Try to set the parameters so that you get a good fit in all spectra.
# Execute FOOOF fits on each subject's eyes-open and eyes-closed data.
# You can use fm.report() to get plots of the fits and check the fit quality,
    you don't have to show these plots though.

# Extract and report the alpha peak frequency and the exponent of the 1/f
    slope.
# One set of parameters should work for all.
# How do alpha peaks and 1/f slopes differ between eyes-open and -closed data?

# Again, using loops and lists is highly recommended to keep the code short
    and easy to read.


peak_width = [1, 10]  # (peak_width_limits)
n_peaks = 6  # (max_n_peaks)
peak_height = 0.1  # (min_peak_height)
PSD_range = [1, 95]  # Freq range to fit

results = {}


# Function to find alpha peak (~8  13  Hz)
def find_alpha_peak(peak_params):
    alpha = np.nan
    for cf, amp, bw in peak_params:
        if 8 <= cf <= 15:
            alpha = cf
            break
    return alpha


for subj, psds in psd_clean_dict.items():
    psd_open = psds['open']
    psd_closed = psds['closed']

    freqs = psd_open.freqs  # same for open/closed if compute_psd uses same
    params

    # Mean over channels
    mean_open = psd_open.get_data().mean(axis=0)
    mean_closed = psd_closed.get_data().mean(axis=0)
```

```
    fm_open = FOOOF(peak_width_limits=peak_width,
                    max_n_peaks=n_peaks,
                    min_peak_height=peak_height)
    fm_open.fit(freqs, mean_open, freq_range=PSD_range)

    fm_closed = FOOOF(peak_width_limits=peak_width,
                      max_n_peaks=n_peaks,
                      min_peak_height=peak_height)
    fm_closed.fit(freqs, mean_closed, freq_range=PSD_range)

    alpha_open = find_alpha_peak(fm_open.peak_params_)
    alpha_closed = find_alpha_peak(fm_closed.peak_params_)
    slope_open = fm_open.aperiodic_params_[1]
    slope_closed = fm_closed.aperiodic_params_[1]

    results[subj] = {
        'open_alpha': alpha_open,
        'open_slope': slope_open,
        'closed_alpha': alpha_closed,
        'closed_slope': slope_closed
    }

    print(f"{subj} - Eyes Open:   alpha={alpha_open:.2f} Hz, slope={slope_open
    :.2f}")
    print(f"{subj} - Eyes Closed: alpha={alpha_closed:.2f} Hz, slope={
    slope_closed:.2f}")

print("\nSummary of FOOOF Fits:")
for subj, vals in results.items():
    print(f"{subj}: "
          f"Open -> alpha={vals['open_alpha']:.2f}, slope={vals['open_slope
    ']:.2f} | "
          f"Closed -> alpha={vals['closed_alpha']:.2f}, slope={vals['
    closed_slope']:.2f}")
```

```
Subject1 - Eyes Open:   alpha=nan Hz, slope=0.97
Subject1 - Eyes Closed: alpha=9.69 Hz, slope=1.36
Subject2 - Eyes Open:   alpha=8.42 Hz, slope=0.83
Subject2 - Eyes Closed: alpha=9.55 Hz, slope=1.40
Subject3 - Eyes Open:   alpha=10.03 Hz, slope=0.93
Subject3 - Eyes Closed: alpha=10.78 Hz, slope=1.32
Subject4 - Eyes Open:   alpha=11.11 Hz, slope=0.85
Subject4 - Eyes Closed: alpha=10.68 Hz, slope=0.94

Summary of FOOOF Fits:
Subject1: Open -> alpha=nan, slope=0.97 | Closed -> alpha=9.69, slope=1.36
Subject2: Open -> alpha=8.42, slope=0.83 | Closed -> alpha=9.55, slope=1.40
Subject3: Open -> alpha=10.03, slope=0.93 | Closed -> alpha=10.78, slope=1.32
Subject4: Open -> alpha=11.11, slope=0.85 | Closed -> alpha=10.68, slope=0.94
```

2.5 Task 5

```
#%% Task 5 (15 points)

# Now let's look at synchrony in the alpha band.
# For each subject and each eeg channel, get the alpha-band narrowband time
    series,
# either using a bandpass or (recommended) by convolving with a Morlet wavelet
     centered on their alpha frequency.
# If you did the previous exercise and found different alpha peaks, you can
    use the mean of the two alpha peak frequencies (eyes-open and -closed).
# If you didn't solve the previous exercise, you can try to visually identify
    alpha peak frequencies from the original power spectra, or just use 10 Hz.


# Now, for each subject, compute the complex phase locking value with the cplv
     function
# between all channels' alpha-band time series, both in the eyes-open and eyes
    -closed 4-min periods.

# Use fig,ax = plt.subplots() to create a figure with 3 rows and 4 columns.
# In the first row, plot eyes-open interaction matrices of iPLV with value
    range [0,.25]
# In the second row, do the same with eyes-closed.
# In the third row, plot the differences with an appropiately chosen value
    range and colormap.
# What do you notice?

# Then, do the same for PLV with value range [0,1] in the first two rows.
# How does this plot differ from that for iPLV? What could be the reason?


raw_cropped_dict = {}

for subj, info in subjects.items():
    raw_open = info['raw'].copy().crop(tmin=info['open'][0], tmax=info['open'
    ][1])
    raw_closed = info['raw'].copy().crop(tmin=info['closed'][0], tmax=info['
    closed'][1])
    raw_cropped_dict[subj] = {'open': raw_open, 'closed': raw_closed}

# Calculate mean alpha frequency per subject
alpha_freqs = {}
for subj, vals in results.items():
    open_alpha = vals.get('open_alpha', np.nan)
    closed_alpha = vals.get('closed_alpha', np.nan)

    if not np.isnan(open_alpha) and not np.isnan(closed_alpha):
        mean_alpha = (open_alpha + closed_alpha) / 2
    elif not np.isnan(open_alpha):
        mean_alpha = open_alpha
    elif not np.isnan(closed_alpha):
        mean_alpha = closed_alpha
    else:
```

```
        mean_alpha = 10  # Default value if both are NaN

    alpha_freqs[subj] = mean_alpha

# Parameters for Morlet wavelet
n_cycles = 7  # Number of cycles in Morlet wavelet

# Dictionaries to store synchrony measures
iplv_open_dict = {}
iplv_closed_dict = {}
plv_open_dict = {}
plv_closed_dict = {}

for subj in subjects.keys():
    mean_alpha = alpha_freqs[subj]

    # Retrieve cropped raw data
    raw_open = raw_cropped_dict[subj]['open']
    raw_closed = raw_cropped_dict[subj]['closed']

    # Get sampling frequency
    sfreq = raw_open.info['sfreq']

    # Get data as [n_channels, n_times]
    data_open = raw_open.get_data()
    data_closed = raw_closed.get_data()

    # Reshape data for Morlet wavelet convolution: [n_epochs, n_channels,
    n_times]
    data_open_reshaped = data_open[np.newaxis, :, :]
    data_closed_reshaped = data_closed[np.newaxis, :, :]

    # Define frequencies for Morlet wavelet (single frequency)
    frequencies = [mean_alpha]

    # Apply Morlet wavelet convolution to extract alpha-band
    power_open = tfr_array_morlet(
        data_open_reshaped,
        sfreq=sfreq,
        freqs=frequencies,
        n_cycles=n_cycles,
        output='complex'
    )
    power_closed = tfr_array_morlet(
        data_closed_reshaped,
        sfreq=sfreq,
        freqs=frequencies,
        n_cycles=n_cycles,
        output='complex'
    )

    # Remove epoch and frequency dimensions, resulting in [n_channels, n_times
    ]
    try:
```

```
        power_open = power_open.squeeze(axis=(0, 2))  # Shape: [n_channels,
    n_times]
        power_closed = power_closed.squeeze(axis=(0, 2))  # Shape: [n_channels
    , n_times]
    except ValueError as e:
        print(f"Error squeezing power arrays for {subj}: {e}")
        continue  # Skip this subject if there's an error

    # Verify shapes
    if power_open.ndim != 2 or power_closed.ndim != 2:
        print(f"Unexpected shape after squeezing for {subj}: Open {power_open.
    shape}, Closed {power_closed.shape}")
        continue  # Skip if shapes are not [n_channels, n_times]

    # Compute complex PLV using cplv function
    cplv_open = cplv(power_open, is_normed=False)
    cplv_closed = cplv(power_closed, is_normed=False)

    # Extract iPLV and PLV
    iplv_open = np.abs(np.imag(cplv_open))
    iplv_closed = np.abs(np.imag(cplv_closed))
    plv_open = np.abs(cplv_open)
    plv_closed = np.abs(cplv_closed)

    # Store in dictionaries
    iplv_open_dict[subj] = iplv_open
    iplv_closed_dict[subj] = iplv_closed
    plv_open_dict[subj] = plv_open
    plv_closed_dict[subj] = plv_closed


# Function to plot interaction matrices
def plot_interaction_matrices(measure_open, measure_closed, measure_diff,
    measure_name, vmin, vmax, cmap):
    fig, axes = plt.subplots(3, 4, figsize=(20, 15))
    fig.suptitle(f"{measure_name} (Alpha Band)", fontsize=20)

    subjects_list = list(subjects.keys())

    for col, subj in enumerate(subjects_list):
        # Eyes-Open
        im1 = axes[0, col].imshow(measure_open[subj], vmin=vmin, vmax=vmax,
    cmap=cmap)
        axes[0, col].set_title(f"{subj} - Open")
        fig.colorbar(im1, ax=axes[0, col], fraction=0.046, pad=0.04)

        # Eyes-Closed
        im2 = axes[1, col].imshow(measure_closed[subj], vmin=vmin, vmax=vmax,
    cmap=cmap)
        axes[1, col].set_title(f"{subj} - Closed")
        fig.colorbar(im2, ax=axes[1, col], fraction=0.046, pad=0.04)

        # Difference
        diff = measure_diff[subj]
```

```
        im3 = axes[2, col].imshow(diff, cmap='bwr')
        vmax_diff = np.max(np.abs(diff))
        im3.set_clim(-vmax_diff, vmax_diff)
        axes[2, col].set_title(f"{subj} - Diff (Closed - Open)")
        fig.colorbar(im3, ax=axes[2, col], fraction=0.046, pad=0.04)

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()


# Prepare difference dictionaries
iplv_diff_dict = {subj: iplv_closed_dict[subj] - iplv_open_dict[subj] for subj
     in subjects.keys()}
plv_diff_dict = {subj: plv_closed_dict[subj] - plv_open_dict[subj] for subj in
     subjects.keys()}

# Plot iPLV
plot_interaction_matrices(
    iplv_open_dict,
    iplv_closed_dict,
    iplv_diff_dict,
    "iPLV",
    vmin=0,
    vmax=0.25,
    cmap='Reds'
)

# Plot PLV
plot_interaction_matrices(
    plv_open_dict,
    plv_closed_dict,
    plv_diff_dict,
    "PLV",
    vmin=0,
    vmax=1,
    cmap='Reds'
)
```

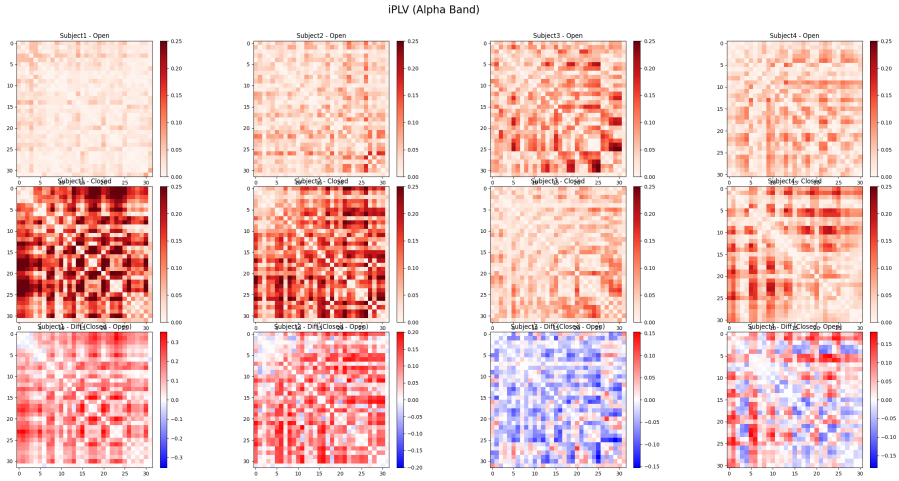

Figure 10: iPLV Alpha Band

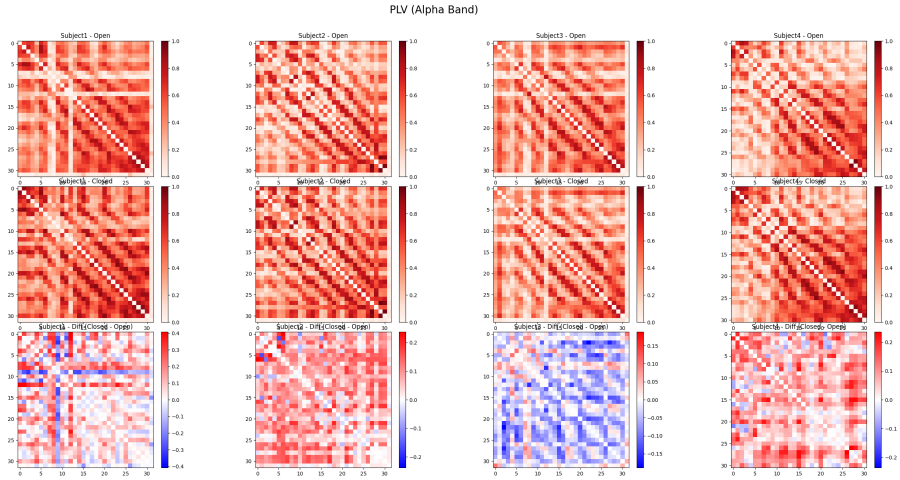

Figure 11: PLV Alpha Band