Neuroscience of Learning, Memory, Cognition
Project Report

Instructor: Prof. Aghajan

Sharif University of Technology

An Analysis of Depressed Individuals EEG signals

By Reza Nayeb Habib - Alireza Jahanifar
Student ID# 401102694 - 401101568

Contents

1 Introduction

1.1 The Goal of The Project

Depression is one of the most common psychological disorders in todays world that affects the feelings and thoughts of the individual. Its clinical signs include sadness, hopelessness, feeling empty, loss of interest or pleasure in most or all normal activities, trouble in thinking, making decisions and concentrating and etc.

Due to the complex nature of this disorder, researchers still can not introduce an accurate model for the brain of the individuals diagnosed with this disorder, but we may be able to analyze the brain of these people in response to a learning task with the knowledge that the disorder has an effect on learning and decision making.

In this project we analyze the response of depressed subjects to a Reinforcement Learning (RL) task especially their response to reward and punishment feedbacks received in the task.

2 Dataset Description

The dataset used in this project is from a research done to find the difference of the response of individuals with anxiety disorder and depressed. The subjects that we will use are Control and Depressed subjects. The subjects state (being depressed or not) is indicated by BDI test and the subjects aren't necessarily clinically diagnosed with depression.

Both subject groups are subjected to a statistical learning task in which they choose between two different hiraganas (Japanese letters) with a random chance of giving a reward or punishment to the subject. The probability of giving a reward for one hiragana is the compliment of the other meaning if the first hiragana has a chance of p for giving reward feedback and $1-p$ for giving a punishment feedback the other hiragana has the respective chances of $1-p$ and p for reward and punishment. The subject should learn to choose the hiragana with a higher chance of giving reward by trial and error. This process is done until the trials reach a certain number or the subject reaches a threshold for choosing the right hiragana meaning he or she has learned the correct hiragana choice for every group of two hiraganas.

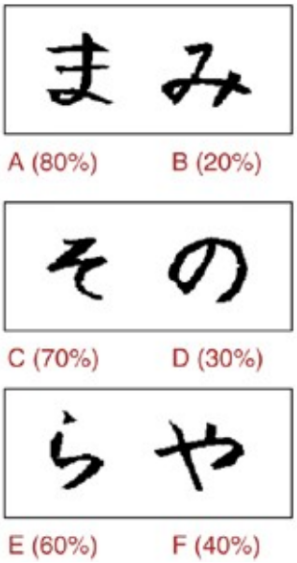


Figure 1: Examples of hiragana groups and their reward probability

Throughout the process the patients head is connected to an Electroencephalography device (EEG) and their brain signals are recorded. The different events happening in the experiment are in the EEG

datasets. for example the moment the pictures are shown or whenever reward and punishment feedback is given to the patient is marked in the dataset.

3 Preprocessing The Data

For preprocessing the data we used Makoto's preprocessing pipeline using EEGLAB in MATLAB. The following sections describe the process in detail. At the end a code will be represented that does the automatic parts of the pipeline(the choice of rejected data sections and ICA components were done by hand and are not incorporated into the code).

3.1 Importing the Data

The dataset used wasn't raw and it was an EEGLAB dataset therefore the importing process is pretty straightforward:

```
[ALLEEG EEG CURRENTSET ALLCOM] = eeglab;
EEG = pop_loadset('filename',name,'filepath',path);
[ALLEEG, EEG, CURRENTSET] = eeg_store( ALLEEG, EEG, 0 );
```

3.2 Bad Period Rejection

Bad data periods like strong movements of patient's head or various other reasons can damage the frequency response and ICA analysis. In this dataset most bad data periods are marked with ' keyboard0 ' and can be easily found. The following code plots the data and presents it to the coder and they can choose data bad periods and give them to `eeg_eegrej` function as a $2 \times n$ matrix with rows representing bad data periods that should be removed.

```
% plot data and reject bad periods
pop_eegplot( EEG, 1, 1, 1);
bads = [3 3;6 57119;59288 59372;61378 61468;] % example periods
EEG = eeg_eegrej( EEG, bads);
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 1,'name'
                        , 'name' ,'comments',[],'gui','off');
```

3.3 Importing Channel Locations and Channel Selection

In the next step we find the channel locations with EEGLAB default head model and then remove channels that were not relevant to our processing (CB1, CB2, HEOG, VEOG, EKG were all removed because we didn't use eye or muscle signals in our preprocessing pipeline).

```
% channel data and rejection
EEG=pop_chanedit(EEG, {'lookup','ch_path\\standard_1005.elc'});
[ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, CURRENTSET);
EEG = pop_select( EEG, 'rmchannel',{'CB1','CB2','HEOG','VEOG','EKG'});
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 2,'setname',
                        'name','comments',[],'gui','off');
```

3.4 Re-referencing the Data

In this dataset the MB1 and MB2 channels were placed behind the ears and we Re-referenced the data to these two channels.

```
% Re-reference data
EEG = pop_reref( EEG, [33 43] );
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 3,
'setname','name','comments',[],'gui','off');
```

3.5 Apply High-pass filter on the Data at 1Hz

The following code applies a high-pass filter on the data at 1Hz for removing the drift seen typically in EEG signals.

```
% hpf at 1Hz
EEG = pop_eegfiltnew(EEG, 'locutoff',1,'plotfreqz',1);
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 4,'setname',
    'name','comments',[],'gui','off');
```

3.6 Filtering Line noise with a notch filter

Analog signals always have line noise due to the usage of amplifiers. using the cleanLine tool from EEGLAB we remove this noise.

```
% notch filtering line nois
EEG = pop_cleanline(EEG, 'bandwidth',2,'chanlist',[1:60] ,
    'computepower',1,'linefreqs',60,'newversion',0,'normSpectrum',
    0,'p',0.01,'pad',2,'plotfigures',0,'scanforlines',0,'sigtype',
    'Channels','taperbandwidth',2,'tau',100,
    'verb',1,'winsize',4,'winstep',1);
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 5,'setname',
    'name','comments',[],'gui','off');
```

3.7 Artifact Subspace Reconstruction(and Re-referencing to AVG)

Artifact Subspace Reconstruction (ASR) is used to reconstruct periods of data that have a high voltage changes (meaning they are not brain signals) and tries to remove those artifacts but retain the brain signal.

we also re-reference the data to average for it to have zero sum both before and after the ASR.

```
% notch filtering line nois
EEG = pop_cleanline(EEG, 'bandwidth',2,'chanlist',[1:60] ,
    'computepower',1,'linefreqs',60,'newversion',0,'normSpectrum',
    0,'p',0.01,'pad',2,'plotfigures',0,'scanforlines',0,'sigtype',
    'Channels','taperbandwidth',2,'tau',100,
    'verb',1,'winsize',4,'winstep',1);
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 5,'setname',
    'name','comments',[],'gui','off');
```

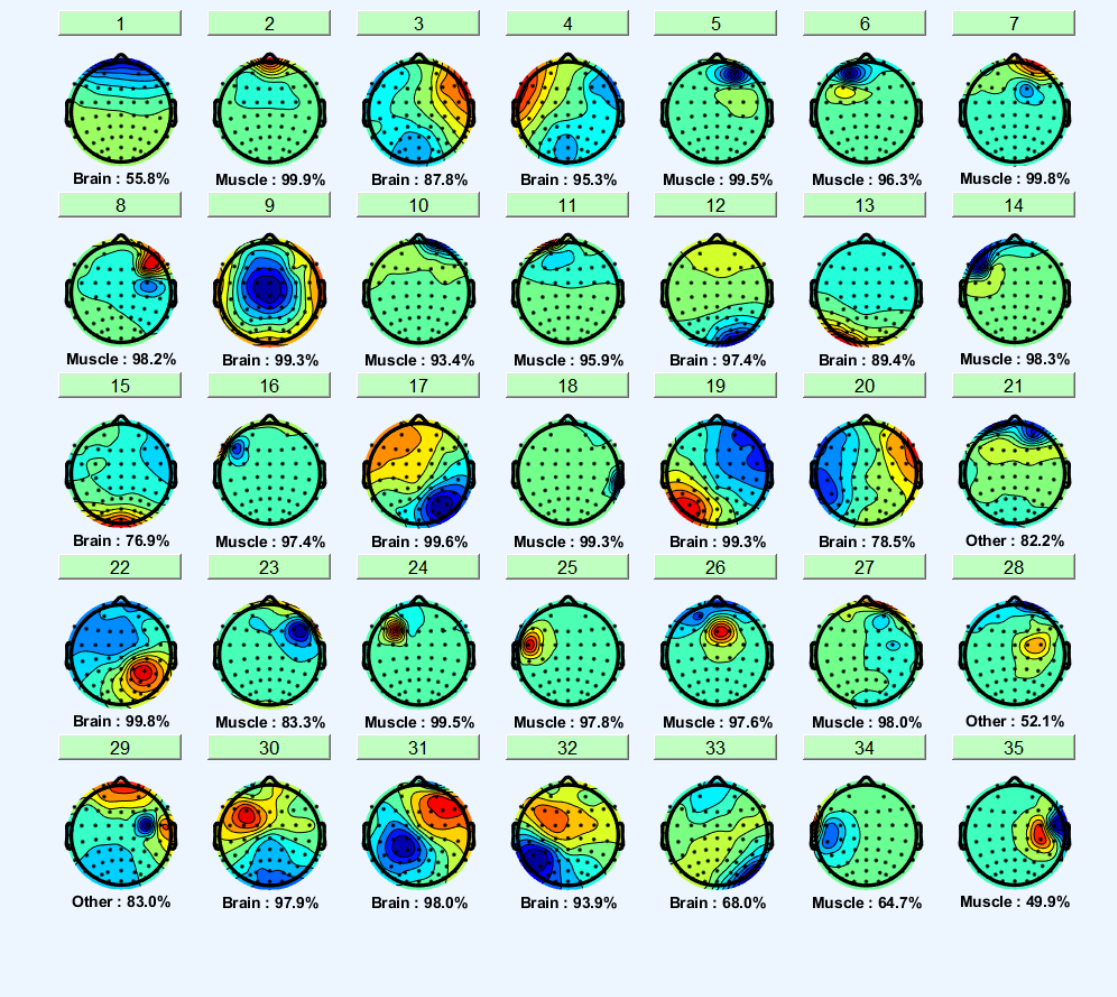
3.8 Independent Component Analysis (ICA)

ICA is a Blind Source Separation method for finding Independent signals from signals that are received through different locations for finding different sources of signals and keep the data that is useful and dispose the noise data. In EEG signals ICA is used for differentiating between brian artifacts and noise

artifacts including muscle signals, eye blinks, channel noise and etc. Here we present the code and after that show the ICA components of one of the patients showing which components we removed and why did we remove them.

```
% Independent Component Analysis
EEG = pop_runica(EEG, 'icatype', 'runica', 'extended',
    1,'rndreset','yes','interrupt','on','pca',59);
[ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, CURRENTSET);
EEG = pop_iclabel(EEG, 'default');
[ALLEEG, EEG, CURRENTSET] = eeg_store(ALLEEG, EEG, CURRENTSET);
pop_eegplot( EEG, 0, 1, 1);
% Example component list:
removed_components = [4   6   26   53   54   55   56   57   58   59]
EEG = pop_subcomp( EEG, removed_components, 0);
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 9,'setname',
    'name','comments',[],'gui','off');
```

The figures below show the IC components for one of the depressed subjects:



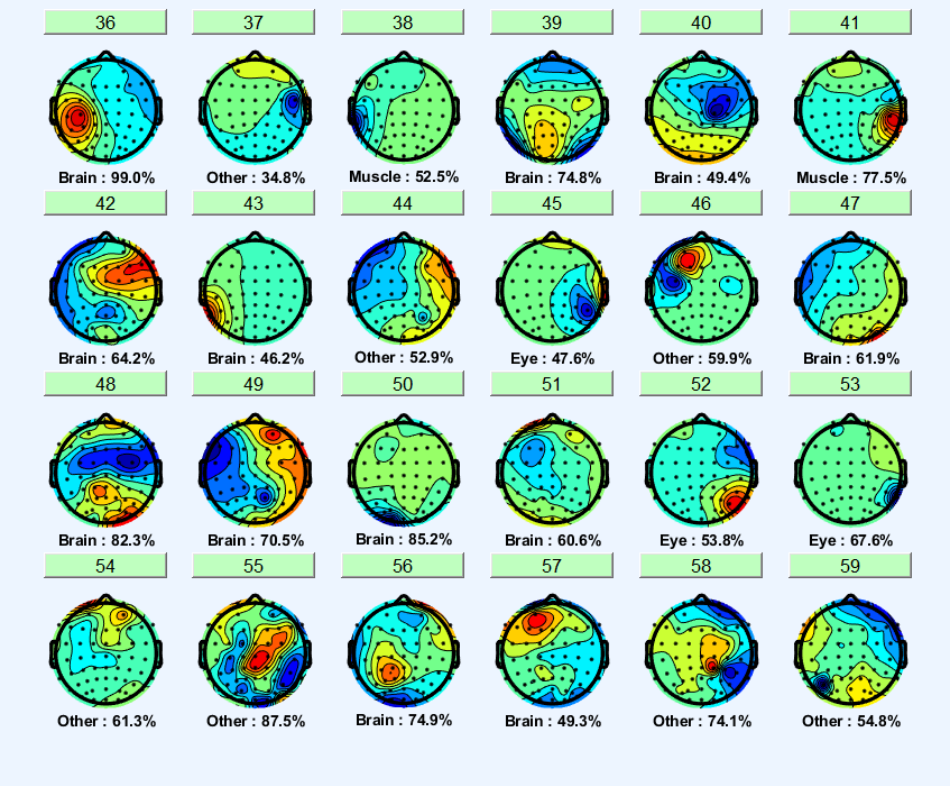


Figure 2: IC components of depressed subject 594

Now we show some components and why we removed them or kept them, below:

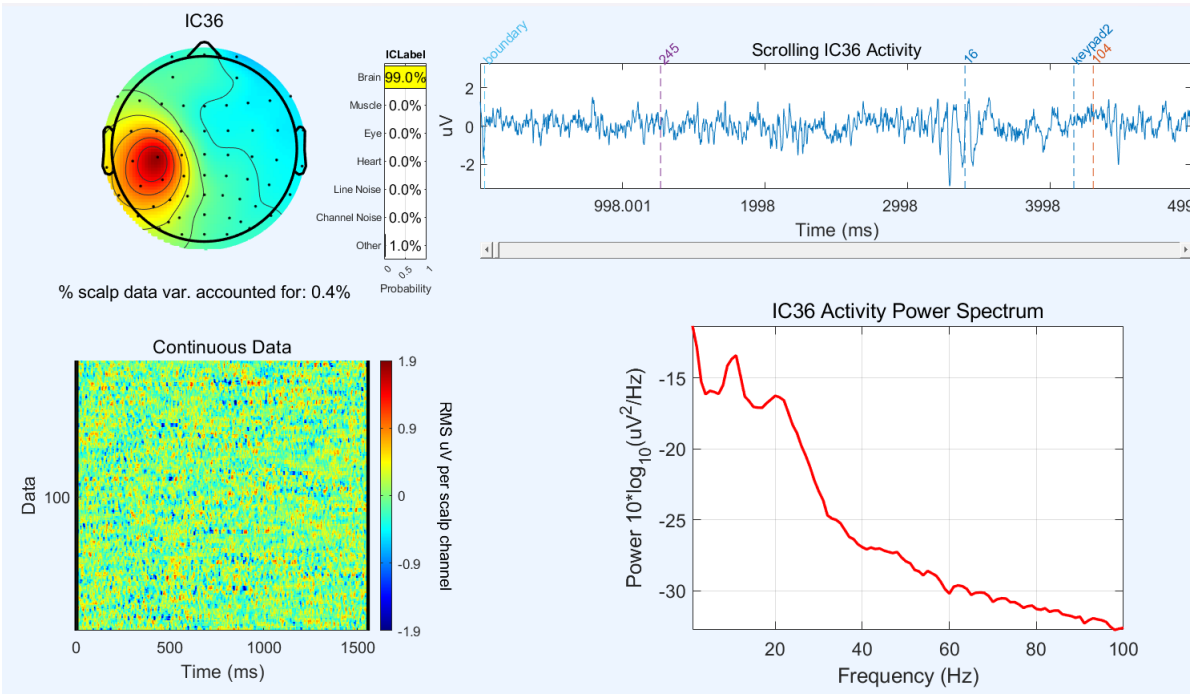


Figure 3: IC 36 was kept because it had a peak at 10H z which brain signals do also its temporal map on the head resembles brain signals

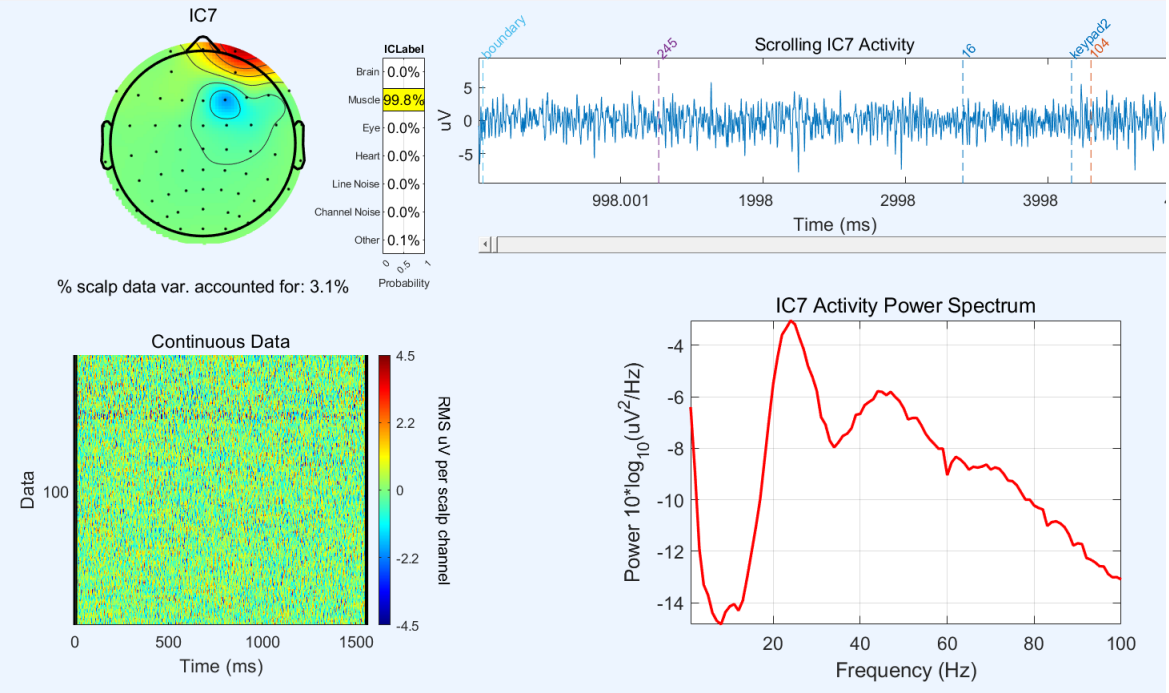


Figure 4: IC 7 was removed because it had high values at high frequencies which is typical for muscle signals also its temporal map on the head model is concentrated at one neighborhood which again is typical for muscle signals

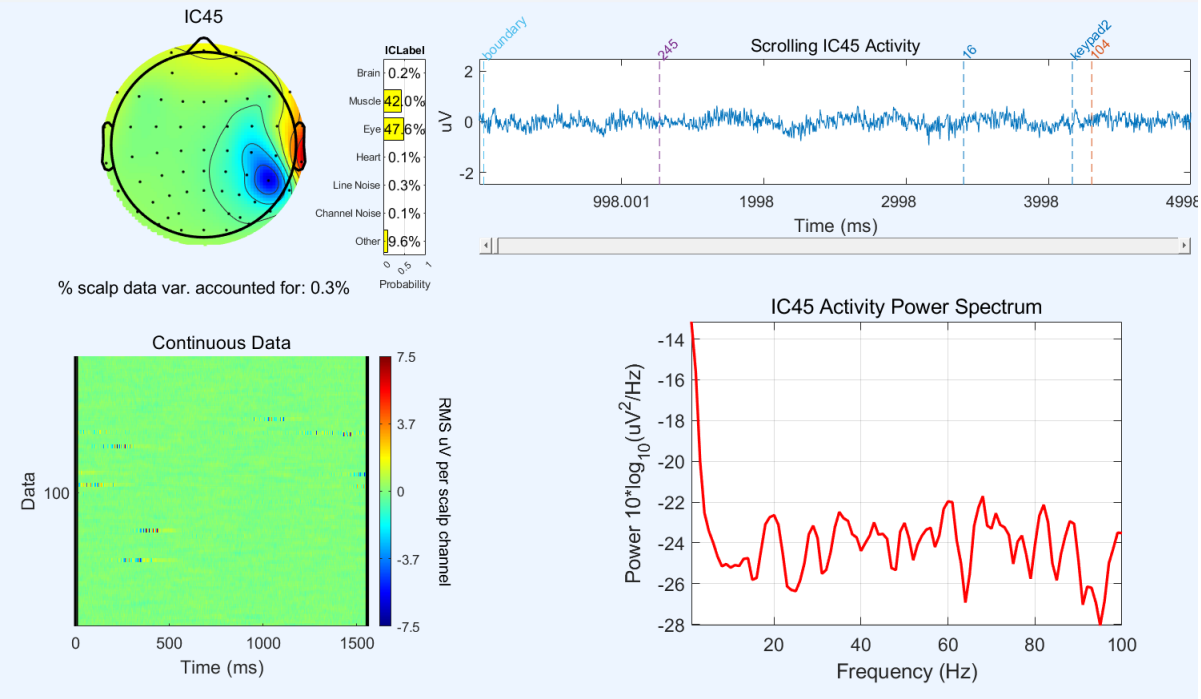


Figure 5: IC 54 doesn't have a 1/f shape like breain signals in frequency domain and has a peak at really low frequencies. It also has a brain map shape with high values near the eye. Both these signs show that it may be an eye signal

3.9 Epoching the data for further analysis

After doing all of Makoto's preprocessing step we then epoch the data based on event marked in the file and save the data for further analysis. the code below shows an example of epoching the data based on some example events:

```
list_of_events = {'10'  '11'  '12'  '13'  '14'  '15'  ... }
time_range = [-0.5 1]
EEG = pop_epoch( EEG, list_of_events, time_range,
    'newname', 's516_stimulus', 'epochinfo', 'yes');
[ALLEEG EEG CURRENTSET] = pop_newset(ALLEEG, EEG, 10,'savenew',
    'adress\\name.set','gui','off');
```

4 Event Related Potentials (ERP)

4.1 Loading the Data

first we present the code used for importing the `EEGLAB` preprocessed data into python using `MNE` library:

```
%%capture
# capture silences the cell
control_index = [510, 516, 568, 576, 577]
depressed_index = [590, 592, 594, 595, 602]

channels = ["FPZ", "F7", "F3", "FZ", "F4", "F8", "T7",
            "C3", "CZ", "C4", "T8", "P7", "P3", "PZ", "P4", "P8", "OZ"]

# Define file types and create dictionaries to hold data
file_types = ["reward", "punishment", "stimulus", "action"]
data_groups = {"control": control_index, "depressed": depressed_index}
all_data = {group: {ftype: {} for ftype in file_types}
        for group in data_groups}

'''
Loop over groups, file types, and subject indices to load data
Every all_data[group][ftype][i] contains one mne epochs object
'''
for group, indices in data_groups.items():
    for ftype in file_types:
        for i in indices:
            addr = f"preprocessed/{group}/subject{i}/s{i}_{ftype}.set"
            print(addr)  # [DEBUG]
            data = mne.read_epochs_eeglab(addr, events=None, event_id=None,
                eog=(), uint16_codec=None, montage_units='auto', verbose=False)
            all_data[group][ftype][i] = data.pick(channels)
```

The code imports all the epochs we have created into the list `all_data`. Any type of event epochs can be found using the subjects group and the event type and the subject number, For example for getting the reward epochs of depressed subject 602 you should type in `all_data["depressed"]["reward"][602]`.

4.2 Z-Score Calculation

Because multiple EEG data can have slightly different voltage ranges and we want to normalize them on every channel to compare (and add them together in ERP calculation), so we compute the data's Z-Score

and use that as the basis of our further calculations. The formula for Z-Score calculation is as follows:

$$x_{zscore}(t) = \frac{x(t) - mean(x)}{std(x)}$$

It can be seen that the data is moved around zero and it's changes are normalize by the amount of standard deviation. The function `zscore_epochs` gets the epochs and replaces data points with their Z-score:

```
def zscore_epochs_internal(epochs, print_indices=None):
"""
calculates Z-score for the given epochs and returns the new Z-socre data
print_indices is for [DEBUG]
"""
data = epochs.get_data()  # Get the raw data as a NumPy array
times = epochs.times  # Required for rescale

# Apply MNE's rescale with mode 'zscore' and baseline=None
data_zscore = mne.baseline.rescale(data, times=times, baseline=(None,None),
     mode='zscore', copy=True)
# [DEBUG] shows the value at those indices before and after z-score calculation
if print_indices:
    print(f"Original value at {print_indices}: {data[print_indices]}")
    print(f"Z-scored value at {print_indices}: {data_zscore[print_indices]}")
    print("-----")

z_epochs = copy.deepcopy(epochs)
# Modify the original Epochs object with transformed data
z_epochs._data = data_zscore  # Replace raw data with Z-scored data
return z_epochs



# Create a new dictionary to store the Z-scored data
all_data_zscore = {group: {ftype: {} for ftype in file_types}
    for group in data_groups}

# Loop over all groups, conditions, and indices
for group, indices in data_groups.items():
    for ftype in file_types:
        for i in indices:
            # Apply the Z-score transformation using MNE's rescale internally
            epochs = all_data[group][ftype][i].copy()
            # Use .copy() to avoid altering original data

            # Specify the indices to print (epoch, channel, timepoint)
            print_indices = (0, 1, 5)  # [DEBUG]
            all_data_zscore[group][ftype][i] = zscore_epochs_internal(epochs,
                 print_indices=print_indices)
```

The follwing code plots some epochs along with their Z-score to show how the data changes with applying Z-Score:

```
def zscore_epochs_internal(epochs, print_indices=None):
"""
calculates Z-score for the given epochs and returns the new Z-socre data
print_indices is for [DEBUG]
"""
```

```
data = epochs.get_data()  # Get the raw data as a NumPy array
times = epochs.times  # Required for rescale

# Apply MNE's rescale with mode 'zscore' and baseline=None
data_zscore = mne.baseline.rescale(data, times=times, baseline=(None,None),
        mode='zscore', copy=True)
# [DEBUG] shows the value at those indices before and after z-score calculation
if print_indices:
    print(f"Original value at {print_indices}: {data[print_indices]}")
    print(f"Z-scored value at {print_indices}: {data_zscore[print_indices]}")
    print("-----")

z_epochs = copy.deepcopy(epochs)
# Modify the original Epochs object with transformed data
z_epochs._data = data_zscore  # Replace raw data with Z-scored data
return z_epochs



# Create a new dictionary to store the Z-scored data
all_data_zscore = {group: {ftype: {} for ftype in file_types}
    for group in data_groups}

# Loop over all groups, conditions, and indices
for group, indices in data_groups.items():
    for ftype in file_types:
        for i in indices:
            # Apply the Z-score transformation using MNE's rescale internally
            epochs = all_data[group][ftype][i].copy()
            # Use .copy() to avoid altering original data

            # Specify the indices to print (epoch, channel, timepoint)
            print_indices = (0, 1, 5)  # [DEBUG]
            all_data_zscore[group][ftype][i] = zscore_epochs_internal(epochs,
                    print_indices=print_indices)
```

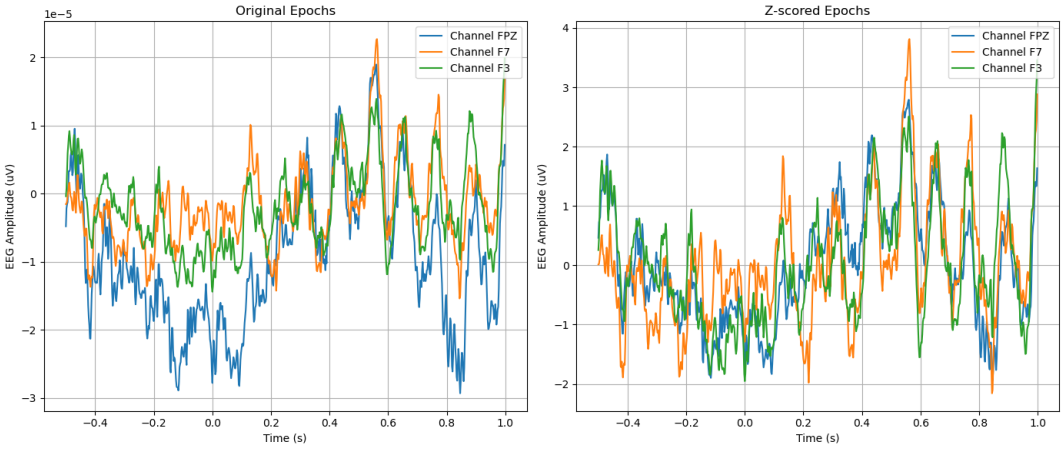


Figure 6: data compared to it's Z-score

4.3 ERP Definition and Implementation

When we record EEG data related to a certain event, due to the randomness in brain signals and high noise in EEG recordings we cannot find the exact response of the brain to the particular event, for finding the meaningful part of the response signal we do the experiment several times and average on all of the trials.

The following code finds the ERP of a list of `MNE` epochs:

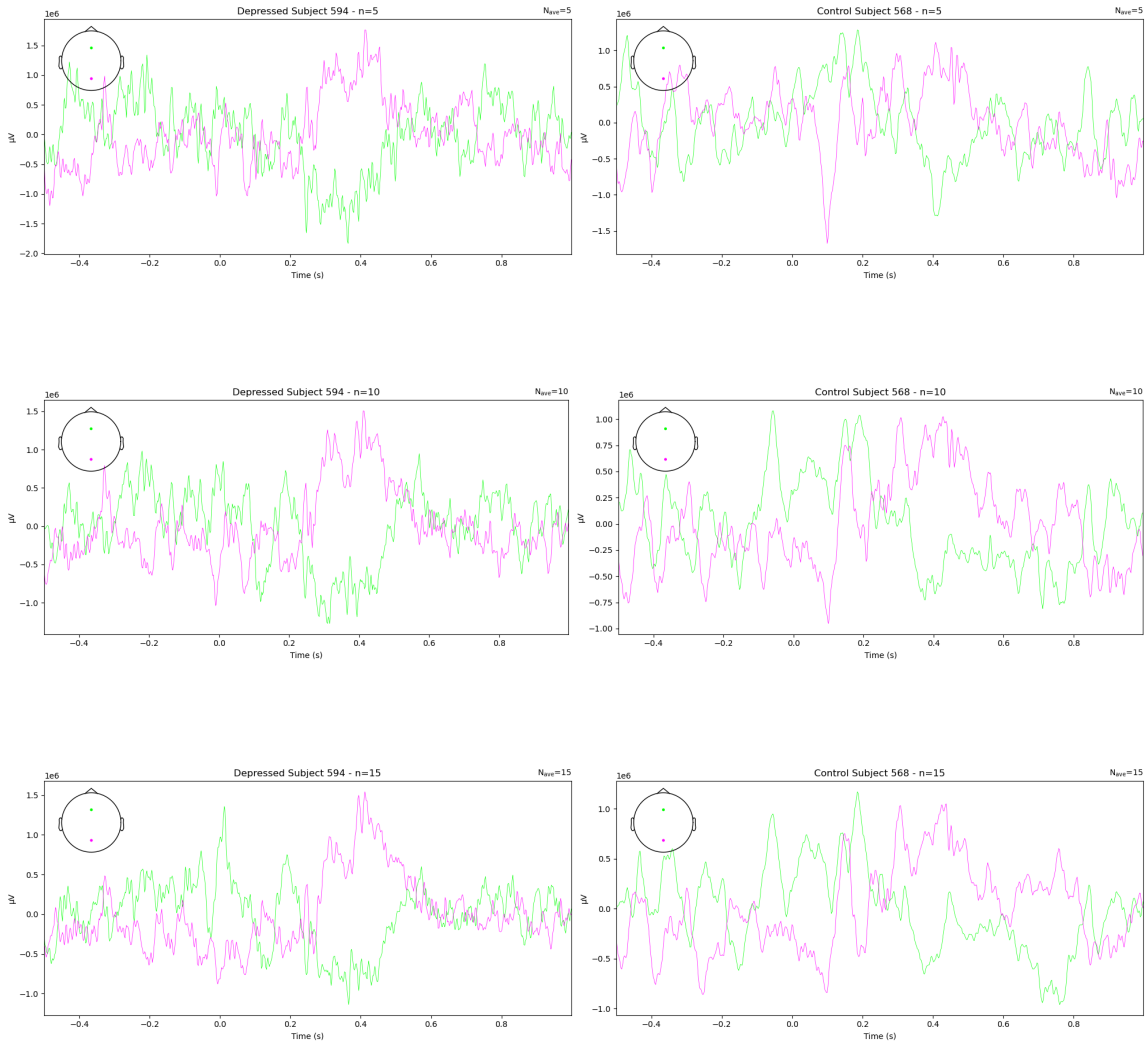
```
def ERP_onlist(epochs, trial_list=None):
"""
Calculates ERP from the selected trials of the given EEG data.

Parameters:
- epochs: The EEG data from which ERP is calculated (MNE Epochs object).
- trial_list: A list of trial indices to include for ERP calculation.
If None, all trials are used.

Returns:
- erp: The ERP object for the selected trials.
"""
if trial_list is None:
    trial_list = range(len(epochs))  # Default to all

# Ensure the trial list is not empty
# and doesn't exceed the number of available epochs
if len(trial_list) == 0:
    raise ValueError("trial_list must not be empty.")
if max(trial_list) >= len(epochs):
    raise ValueError(f"trial_list contains indices out of range.
        Maximum index: {max(trial_list)}")

epochs_subset = epochs[trial_list]
erp = epochs_subset.average()
return erp
```

4.4 Effect of Increasing trials on ERP

Now having the z-score values and ERP function we will show the effect of increasing the number of trials on the ERP signals shape, showing that indeed by this procedure we obtain a better signal shape and will converge to a less noisy signal:

```
# List of n values for which we want to see the effect on ERP
n_values = [5, 10, 15, 20, 100]

# Example subjects and channels (FZ and PZ)
chosen_dep = 594
chosen_ctrl = 568
chosen_ch = ["FZ", "PZ"]

example_dep = all_data_zscore["depressed"]["stimulus"][chosen_dep]
example_ctrl = all_data_zscore["control"]["stimulus"][chosen_ctrl]

for n in n_values:
    erp_dep = ERP_onlist(example_dep, range(n))
    erp_ctrl = ERP_onlist(example_ctrl, range(n))

```

```
    fig, axes = plt.subplots(1, 2, figsize=(20, 5))

    # Plot the ERP for the depressed subject (FZ and PZ)
    erp_dep.plot(picks=chosen_ch, axes=axes[0], show=False)
    axes[0].set_title(f"Depressed Subject {chosen_dep} - n={n}")

    # Plot the ERP for the control subject (FZ and PZ)
    erp_ctrl.plot(picks=chosen_ch, axes=axes[1], show=False)
    axes[1].set_title(f"Control Subject {chosen_ctrl} - n={n}")

    plt.tight_layout()
    plt.show()
```



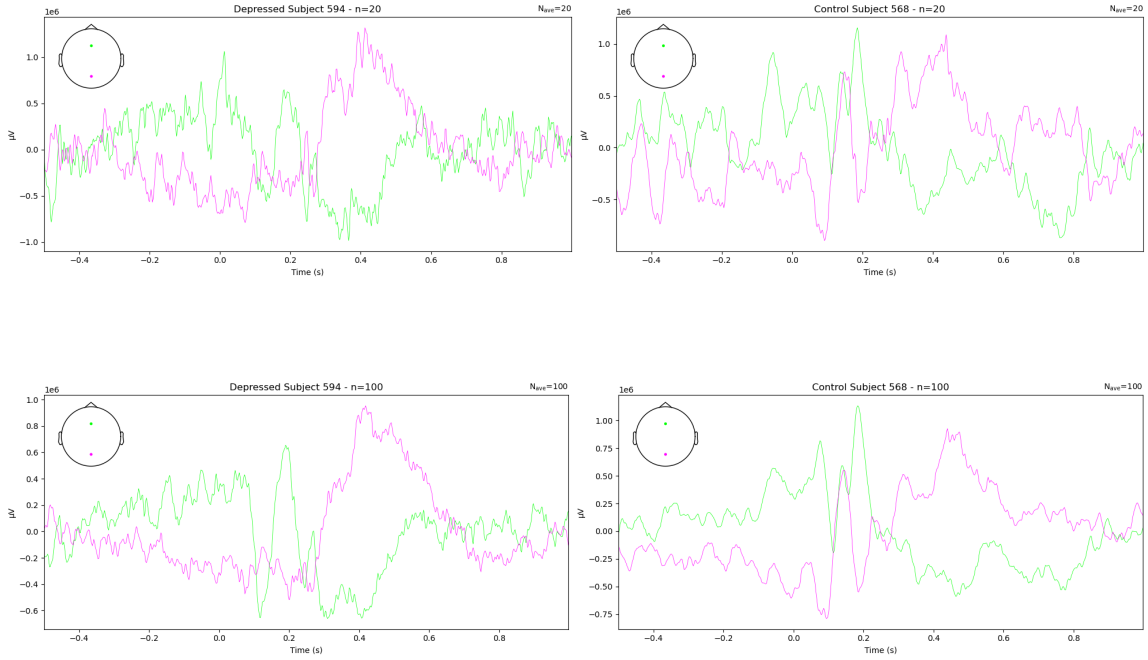


Figure 7: Effect of increasing epochs on ERP shape

4.5 Difference in Punishment and Reward ERP for Depressed and Control subjects

For analysing the differences between depressed and normal subjects better we show the ERP of each subject in response to reward and punishment and try to see if there is anny difference between the response of the subjects to these stimuluses. For doing so we first compute the ERP for all of the reward and punishment events of one subject on FZ and PZ channels (frontal lobe and parietal lobe) and then plot them together. The data isn't still very useful for analysing by eye, but we shall fix that in the next section by grand averaging. below is the code developed for this task:

```
chosen_ch = ["FZ", "PZ"]
fz_reward = []
fz_punish = []
pz_reward = []
pz_punish = []
fz_reward_labels = []
fz_punish_labels = []
pz_reward_labels = []
pz_punish_labels = []
# colors for different subjects
control_colors = ['#d62728', '#ff7f0e', '#e41a1c', '#9e0142', '#f46d43']
# Different warm shades for control group
depressed_colors = ['#1f77b4', '#377eb8', '#66aaff', '#1f78b4', '#a6cee3']
# Different cool shades for depressed group

# Loop through groups and subjects
for group, indices in data_groups.items():
    for n_subject in indices:
        # Get the data for the current subject
        reward = all_data_zscore[group]["reward"][n_subject]
        punish = all_data_zscore[group]["punishment"][n_subject]

        reward_ERP = ERP_onlist(reward)
```

```
        punish_ERP = ERP_onlist(punish)

        # Pick FZ and PZ channels
        fz_reward.append(reward_ERP.copy().pick('FZ'))
        fz_punish.append(punish_ERP.copy().pick('FZ'))
        pz_reward.append(reward_ERP.copy().pick('PZ'))
        pz_punish.append(punish_ERP.copy().pick('PZ'))

        # Store labels for the legend (showing condition and subject number)
        fz_reward_labels.append(f"Reward {group} {n_subject}")
        fz_punish_labels.append(f"Punishment {group} {n_subject}")
        pz_reward_labels.append(f"Reward {group} {n_subject}")
        pz_punish_labels.append(f"Punishment {group} {n_subject}")

# Prepare the evokeds dictionary
evokeds = {
    'FZ - Reward': fz_reward,
    'FZ - Punishment': fz_punish,
    'PZ - Reward': pz_reward,
    'PZ - Punishment': pz_punish,
}

fig, axes = plt.subplots(2, 2, figsize=(30, 20))

# Plot each ERP in the corresponding subplot with unique color
# and line style for each subject
for idx, (evoked_data, labels, ax, title) in
    enumerate(zip([fz_reward, fz_punish, pz_reward, pz_punish],
        [fz_reward_labels, fz_punish_labels, pz_reward_labels, pz_punish_labels],
        axes.flatten(),
        ['FZ - Reward', 'FZ - Punishment', 'PZ - Reward', 'PZ - Punishment'])):
    # Plot each subject with a unique color and line style based on their group
    for i, erp in enumerate(evoked_data):
        # Assign a unique color for each subject (based on group)
        if 'control' in labels[i]:
            color = control_colors[control_index.index(
                int(labels[i].split()[-1]))]
            # Get color for control
        else:
            color = depressed_colors[depressed_index.index(
                int(labels[i].split()[-1]))]
            # Get color for depressed

        ax.plot(erp.times, erp.data.T, color=color, linestyle='-',
            label=labels[i])

    # Set the title and labels
    ax.set_title(title)
    ax.set_xlabel('Time (s)')
    ax.set_ylabel('Amplitude')

    # Add the legend
    ax.legend(loc='best')

plt.tight_layout()
plt.show()
```

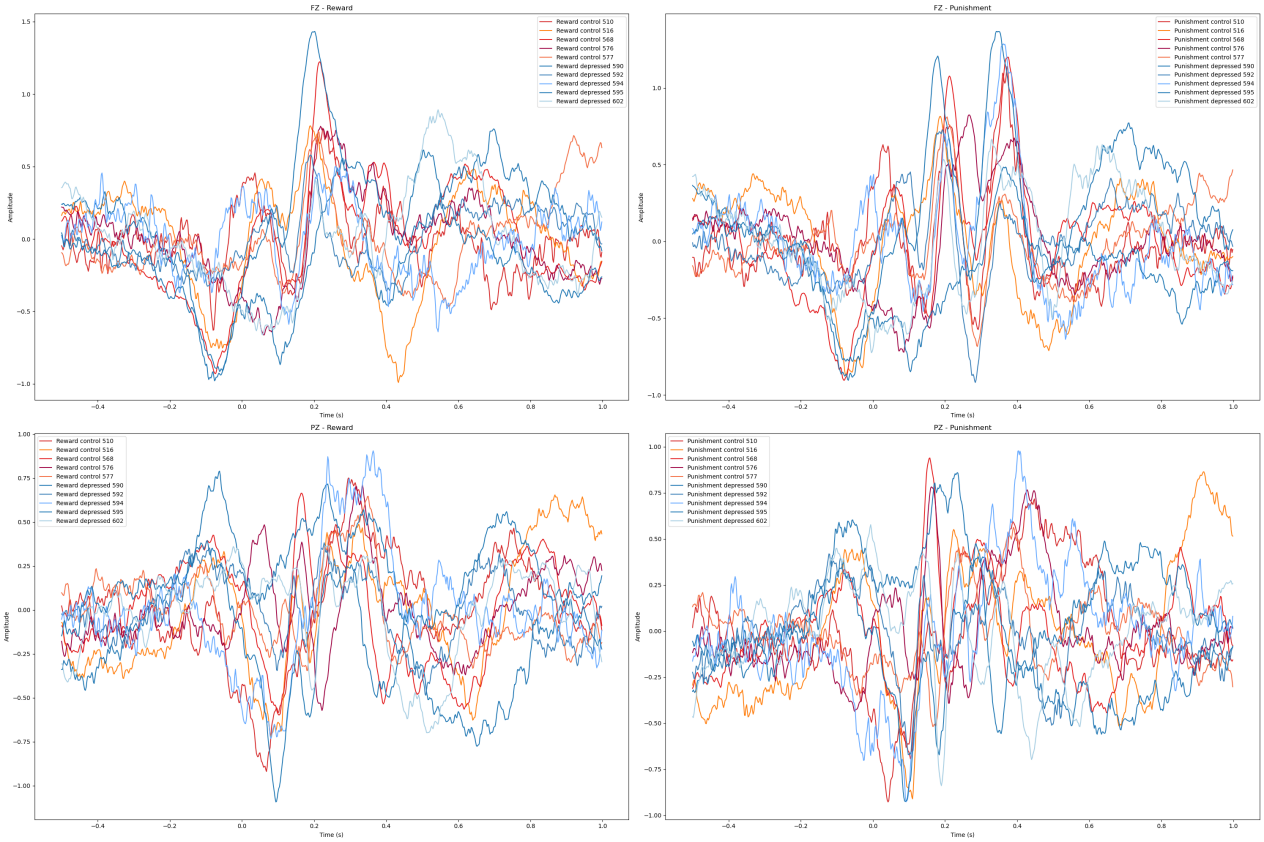


Figure 8: The ERPs for all subjects on Fz and Pz in one plot

4.6 Grand Averaging ERPs

As it can be seen the ERPs of different subjects are similar in shape but have differences with each other that prevents us from understanding the patterns in their brian signals, in order to solve this problem we take the grand average of all ERPs for a type of patient and event on a channel to see understand the changes. For doing so we first group the individual and take the average of ERPs on all individuals. In order to still have the info of how much the average can vary (how much is this average close in between all patiens and how much close are the patients response) we also compute the Standard Error and fill the plot with it ($SE(X) = \frac{std(X)}{\sqrt{n}}$).

The following code does the procedure explained above:

```
# Function to calculate standard error for each time point
def compute_standard_error(data):
    # Stack data from all subjects for each condition
    stacked_data = np.stack(data)
    return np.std(stacked_data, axis=0) / np.sqrt(stacked_data.shape[0])


# Extracting data for each channel, subject type and event
fz_reward_control_data = [fz_reward[i].data for i in
    range(len(fz_reward)) if 'control' in fz_reward_labels[i]]
fz_reward_depressed_data = [fz_reward[i].data for i in
    range(len(fz_reward)) if 'depressed' in fz_reward_labels[i]]

fz_punish_control_data = [fz_punish[i].data for i in
    range(len(fz_punish)) if 'control' in fz_punish_labels[i]]
```

```
fz_punish_depressed_data = [fz_punish[i].data for i in
    range(len(fz_punish)) if 'depressed' in fz_punish_labels[i]]

pz_reward_control_data = [pz_reward[i].data for i in
    range(len(pz_reward)) if 'control' in pz_reward_labels[i]]
pz_reward_depressed_data = [pz_reward[i].data for i in
    range(len(pz_reward)) if 'depressed' in pz_reward_labels[i]]

pz_punish_control_data = [pz_punish[i].data for i in
    range(len(pz_punish)) if 'control' in pz_punish_labels[i]]
pz_punish_depressed_data = [pz_punish[i].data for i in
    range(len(pz_punish)) if 'depressed' in pz_punish_labels[i]]


mean_fz_reward_control = mne.grand_average([fz_reward[i] for i in
    range(len(fz_reward)) if 'control' in fz_reward_labels[i]])
mean_fz_reward_depressed = mne.grand_average([fz_reward[i] for i in
    range(len(fz_reward)) if 'depressed' in fz_reward_labels[i]])
mean_fz_punish_control = mne.grand_average([fz_punish[i] for i in
    range(len(fz_punish)) if 'control' in fz_punish_labels[i]])
mean_fz_punish_depressed = mne.grand_average([fz_punish[i] for i in
    range(len(fz_punish)) if 'depressed' in fz_punish_labels[i]])

mean_pz_reward_control = mne.grand_average([pz_reward[i] for i in
    range(len(pz_reward)) if 'control' in pz_reward_labels[i]])
mean_pz_reward_depressed = mne.grand_average([pz_reward[i] for i in
    range(len(pz_reward)) if 'depressed' in pz_reward_labels[i]])
mean_pz_punish_control = mne.grand_average([pz_punish[i] for i in
    range(len(pz_punish)) if 'control' in pz_punish_labels[i]])
mean_pz_punish_depressed = mne.grand_average([pz_punish[i] for i in
    range(len(pz_punish)) if 'depressed' in pz_punish_labels[i]])


# Compute SE
fz_reward_control_se = compute_standard_error(fz_reward_control_data)
fz_reward_depressed_se = compute_standard_error(fz_reward_depressed_data)

fz_punish_control_se = compute_standard_error(fz_punish_control_data)
fz_punish_depressed_se = compute_standard_error(fz_punish_depressed_data)

pz_reward_control_se = compute_standard_error(pz_reward_control_data)
pz_reward_depressed_se = compute_standard_error(pz_reward_depressed_data)

pz_punish_control_se = compute_standard_error(pz_punish_control_data)
pz_punish_depressed_se = compute_standard_error(pz_punish_depressed_data)

time = fz_reward[0].times  # Time vector is consistent across ERPs

fig, axes_compare = plt.subplots(2, 2, figsize=(15, 10))  # 2x2 layout

# Function to plot with SE
def plot_with_se(ax, time, mean_data, se_data, color, label, linestyle='-'):
    ax.plot(time, mean_data, color=color, label=label,
        linestyle=linestyle, linewidth=2)
    ax.fill_between(time, mean_data - se_data, mean_data + se_data,
        color=color, alpha=0.2)

```

```python
# Define titles and plot for each subplot
titles = ["FZ - Reward", "FZ - Punishment", "PZ - Reward", "PZ - Punishment"]
control_averages = [
    mean_fz_reward_control.data[0], mean_fz_punish_control.data[0],
    mean_pz_reward_control.data[0], mean_pz_punish_control.data[0]
]
depressed_averages = [
    mean_fz_reward_depressed.data[0], mean_fz_punish_depressed.data[0],
    mean_pz_reward_depressed.data[0], mean_pz_punish_depressed.data[0]
]
control_ses = [fz_reward_control_se[0], fz_punish_control_se[0],
    pz_reward_control_se[0], pz_punish_control_se[0]]
depressed_ses = [fz_reward_depressed_se[0], fz_punish_depressed_se[0],
    pz_reward_depressed_se[0], pz_punish_depressed_se[0]]

colors = {"Control": "orange", "Depressed": "#87CEEB"}

for i, ax in enumerate(axes_compare.flatten()):
    plot_with_se(ax, time, control_averages[i], control_ses[i],
        colors["Control"], "Control")
    plot_with_se(ax, time, depressed_averages[i], depressed_ses[i],
        colors["Depressed"], "Depressed", linestyle="-")
    ax.set_title(titles[i])
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Amplitude")
    ax.legend()

plt.tight_layout()
plt.show()
```

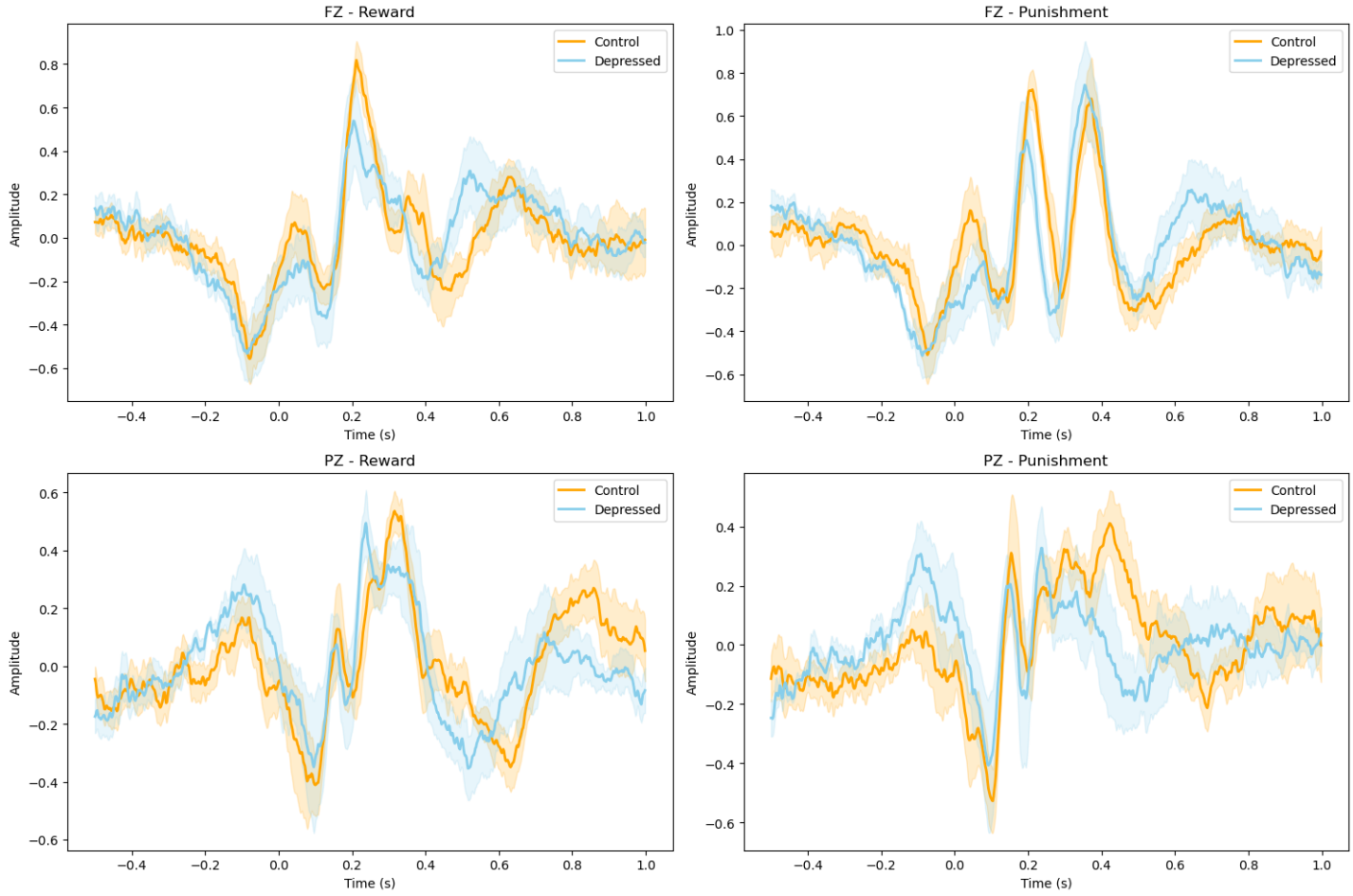


Figure 9: The Grand Avg of ERPs for Depressed and Control subjects for Reward and Punishment

Looking at the Image above we can get some sense of the difference between the depressed responses and control responses:

For Reward ERPs it can be seen that the waveform amplitude for the Depressed group remains flatter overall, indicating a reduced response to reward stimuli.

For Punishment ERPs it can be seen that The Control group maintains a stronger response to punishment stimuli, with higher positive peaks and the Depressed group shows reduced and delayed responses similar to the Reward condition.

In conclusion we can see that firstly Across both Reward and Punishment conditions, the ERP amplitudes for the Depressed group are generally lower than those of the Control group, reflecting decreased neural activity or sensitivity to both positive and negative stimuli. Secondly The differences are more pronounced in the early time windows which are typically associated with rapid, automatic processing of stimuli. Also The Control group exhibits clearer and sharper peaks, whereas the Depressed group's ERP responses appear more blunted, potentially indicative of diminished motivational and affective processing.

5 Frequency Analysis

In the brain usually different types of tasks are done in different frequency bands thus one important way for brain data analysis is the frequency analysis. In this section we will complete our picture of depressed subjects with this type of analysis.

5.1 Brains frequency Bands Overview

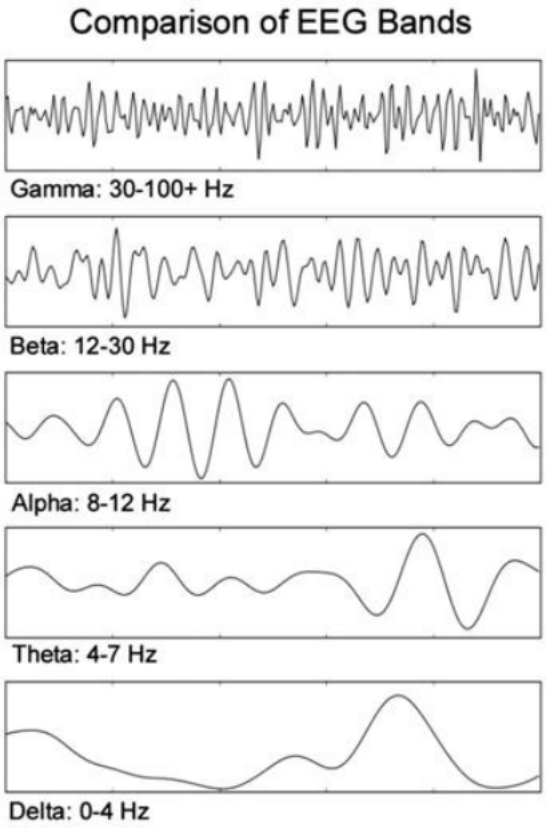


Figure 10: Different Frequency bands of human brain

The human brain operates within distinct frequency bands, each associated with specific functions and states. These frequency bands, derived from electroencephalography (EEG) analysis, provide insights into various brain processes ranging from cognitive tasks to emotional states. Below, we describe the primary brain frequency bands, their functions, and applications in neuroscience research.

Delta Band (0.5–4 Hz): The delta band is the lowest frequency range in the brain and is primarily associated with deep sleep and restorative processes. It is commonly observed during non-REM sleep stages and is critical for cellular regeneration and healing. In research applications, delta activity is often studied in sleep disorders, as well as in pathological conditions like brain injuries or coma. Abnormal delta activity in wakeful states may indicate neurological damage or cognitive impairments.

Theta Band (4–8 Hz): Theta waves are linked to drowsiness, light sleep, and meditative states. They play a key role in memory encoding and retrieval, particularly in tasks involving spatial navigation and working memory. Increased theta activity has been observed during states of creativity and daydreaming. Researchers often analyze theta waves in the context of learning, memory processes, and certain psychiatric conditions like attention-deficit/hyperactivity disorder (ADHD).

Alpha Band (8–13 Hz): Alpha waves are prominent when the brain is in a relaxed and calm state, ypically with closed eyes but not asleep. They originate primarily from the occipital lobe and are associated

with resting and disengagement from active tasks. Alpha rhythms are frequently studied in relaxation techniques, meditation research, and sensory processing. Abnormalities in alpha activity can indicate stress, anxiety, or neurological disorders such as epilepsy.

Beta Band (13–30 Hz): Beta waves represent active cognitive states and are associated with attention, problem-solving, and decision-making. These waves increase during tasks that require focus and alertness and are typically generated in the frontal lobe. Research applications of beta activity include studies on cognitive performance, arousal, and emotional regulation. Excessive beta activity has been linked to anxiety, while reduced beta may correlate with depression or motor impairments.

Gamma Band (30–100 Hz): Gamma waves are the highest frequency range and are involved in higher-order cognitive functions such as perception, attention, and consciousness. They are crucial for integrating sensory information and forming coherent mental representations. Gamma activity is frequently studied in neuroscience to understand neural synchronization and its role in complex processes like learning, memory consolidation, and problem-solving. Altered gamma activity is implicated in disorders like schizophrenia and autism.

5.2 Computing STFT and Frequency Band

Seeing the frequency behaviour of the EEG data as a whole can indeed be helpful, but it loses the time frame when we see the whole thing together, instead It is better to see how frequency behaviour changes with time and specially when event happen. For doing so we use Short Time Fourier Transform (STFT) which is the same as fourier transform just computed on small time windows of the time signal instead of the whole signal. This way by summing over the frequency bands we get the behaviour of that frequency band in time (for example for alpha frequency we get the sum (or mean) on frequency range of 8-13Hz).

The following code computes each band power in time and averages over subjects. it also computes the standard error and fill the plot with it. `analyze_and_plot` plots all of the data of a subjects type and event type together while `analyze_and_plot_by_band` plots all of the subjects bands into one plot for each band. We only present the output of the second one here because it is better for comparing the depressed and control subjects together.

```
# Frequency bands
frequency_bands = {
    "Delta": (0.5, 4),
    "Theta": (4, 8),
    "Alpha": (8, 13),
    "Beta": (13, 30),
    "Gamma": (30, 100),
}

f_windows = {
    "Delta": 0.5,
    "Theta": 0.5,
    "Alpha": 0.5,
    "Beta": 0.5,
    "Gamma": 0.5,
}

# STFT function with step parameter
def compute_band_power(epochs, band_limits,
    sample_rate=500, window_size=0.5, step=0.1):
    """
    Compute band power for a given frequency band and epochs using
    a sliding window.
```

```
    Parameters:
    - epochs: Epoch data (n_epochs x n_channels x n_times)
    - band_limits: Tuple of (low, high) frequency band
    - sample_rate: Sampling frequency of data
    - window_size: STFT window size in seconds
    - step: Step size in seconds for sliding the window

    Returns:
    - band_powers: Mean power for the specified band over time
    (n_epochs x n_time_windows)
    - times: Array of time points corresponding to the center of each window
    """
    band_powers = []
    window_samples = int(window_size * sample_rate)
    step_samples = int(step * sample_rate)

    for epoch in epochs:
        # Compute STFT for each channel in the epoch and store band power
        channel_band_powers = []
        for channel_data in epoch:
            f, t, Sxx = spectrogram(
                channel_data,
                fs = sample_rate,
                nperseg = window_samples,
                noverlap = window_samples - step_samples,
                mode='psd'
            )
            # Select frequency band
            band_idx = (f >= band_limits[0]) & (f <= band_limits[1])
            band_power = Sxx[band_idx].mean(axis=0)
            # Mean power over band frequencies
            band_power = (band_power - np.mean(band_power))/np.std(band_power)

            channel_band_powers.append(band_power)

        # Average across channels
        mean_band_power = np.mean(channel_band_powers, axis=0)
        band_powers.append(mean_band_power)

    band_powers = np.array(band_powers)  # Shape: (n_epochs, n_time_windows)
    times = t  # Time points are consistent across epochs

    return band_powers, times

# Extract and process condition/group data
def extract_epochs(data, group, condition):
    """
    Extract epochs for a specific group and condition.

    Parameters:
    - all_data: Dictionary containing all data
    - group: Group name ('control' or 'depressed')
    - condition: Condition name ('reward' or 'punishment')

    Returns:
    - epochs_list: List of epochs (n_epochs x n_channels x n_times)
```

```
    """
    epochs_list = []
    for subject_id, epochs in data[group][condition].items():
        epochs_data = epochs.get_data()
        epochs_list.extend(epochs_data)  # Add all epochs for this subject
    return np.array(epochs_list)

# Process and analyze data for each condition and group
def process_condition_group(epochs, sample_rate=500, step=0.1):
    """
    Compute mean and SE of power for all frequency bands for a condition
    and group.

    Parameters:
    - epochs: List of epochs (n_epochs x n_channels x n_times)
    - sample_rate: Sampling frequency
    - window_size: STFT window size in seconds
    - step: Step size in seconds for sliding the window

    Returns:
    - mean_powers: Dictionary of mean power for each band
    - se_powers: Dictionary of standard error for each band
    - times: Array of time points corresponding to the center of each window
    """
    mean_powers = {}
    se_powers = {}
    times = None  # Initialize times

    for (band, limits), (_, win_siz) in zip(frequency_bands.items(),
            f_windows.items()):
        band_powers, t = compute_band_power(epochs, limits, sample_rate,
            win_siz, step)
        mean_powers[band] = band_powers.mean(axis=0)  # Mean across epochs
        se_powers[band] =
            band_powers.std(axis=0) / np.sqrt(band_powers.shape[0])
            # SE across epochs
        if times is None:
            times = t  # Set times once

    return mean_powers, se_powers, times

# Plot function
def plot_band_changes(time, mean_powers, se_powers, title):
    """
    Plot mean power and standard error for each band.

    Parameters:
    - time: Time points
    - mean_powers: Mean power dictionary
    - se_powers: SE dictionary
    - title: Title of the plot
    """
    plt.figure(figsize=(12, 6))
    for band in frequency_bands.keys():
        plt.plot(time, mean_powers[band], label=f'{band} (mean)', linewidth=2)
        plt.fill_between(
            time,
```

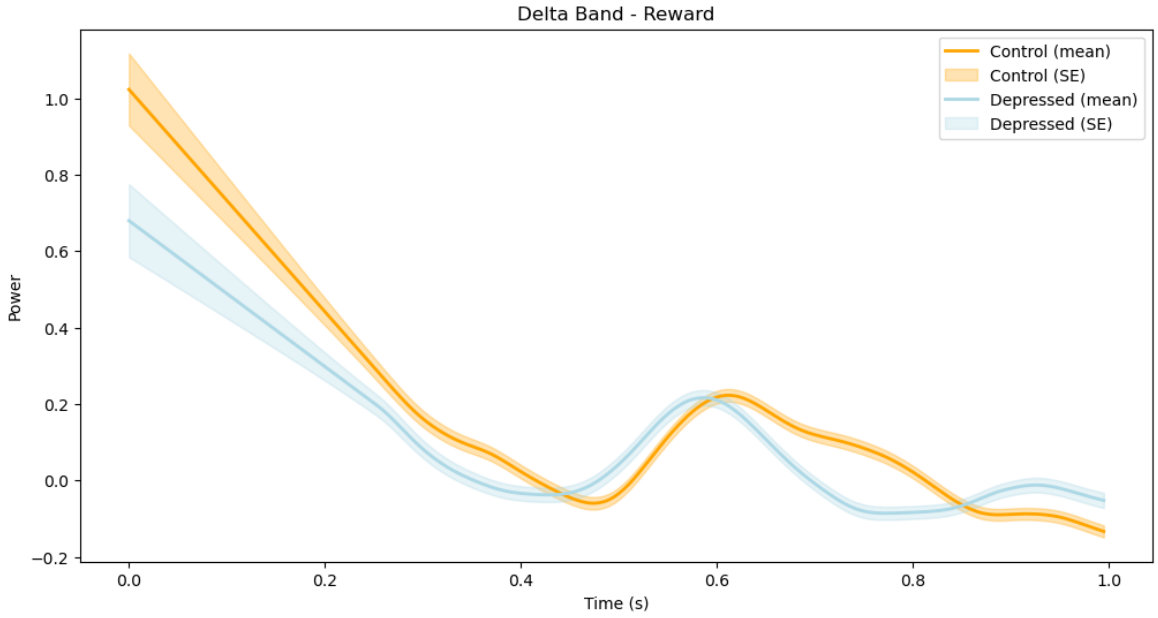
```
            mean_powers[band] - se_powers[band],
            mean_powers[band] + se_powers[band],
            alpha=0.3,
            label=f'{band} (SE)'
        )

    plt.title(title)
    plt.xlabel('Time (s)')
    plt.ylabel('Power')
    plt.legend()
    plt.show()

# Define a common time grid
def get_common_time_grid(sample_rate, epoch_length, step=0.005):
    """
    Create a common time grid for all frequency bands based on the step size.

    Parameters:
    - sample_rate: Sampling frequency
    - epoch_length: Length of the epoch in seconds
    - step: Desired step size in seconds

    Returns:
    - common_times: Array of time points for the common grid
    """
    return np.arange(0, epoch_length, step)

# Resample band power onto a common time grid
def resample_band_power(band_power, original_times, common_times):
    """
    Resample band power data to a common time grid using interpolation.

    Parameters:
    - band_power: Array of band powers (n_epochs x n_time_windows)
    - original_times: Original time points corresponding to the band power
    - common_times: Common time grid to resample onto

    Returns:
    - resampled_power: Resampled band power (n_epochs x len(common_times))
    """
    resampled_power = []
    for epoch_power in band_power:
        f_interp = interp1d(original_times, epoch_power, kind='linear',
            fill_value="extrapolate")
        resampled_power.append(f_interp(common_times))
    return np.array(resampled_power)

# Update the compute_band_power function to handle resampling
def compute_band_power2(epochs, band_limits, sample_rate, window_size,
    step, common_times):
    """
    Compute band power and resample onto a common time grid.
    """
    band_powers, original_times = compute_band_power(epochs,
        band_limits, sample_rate, window_size, step)
    resampled_powers = resample_band_power(band_powers,
        original_times, common_times)
```

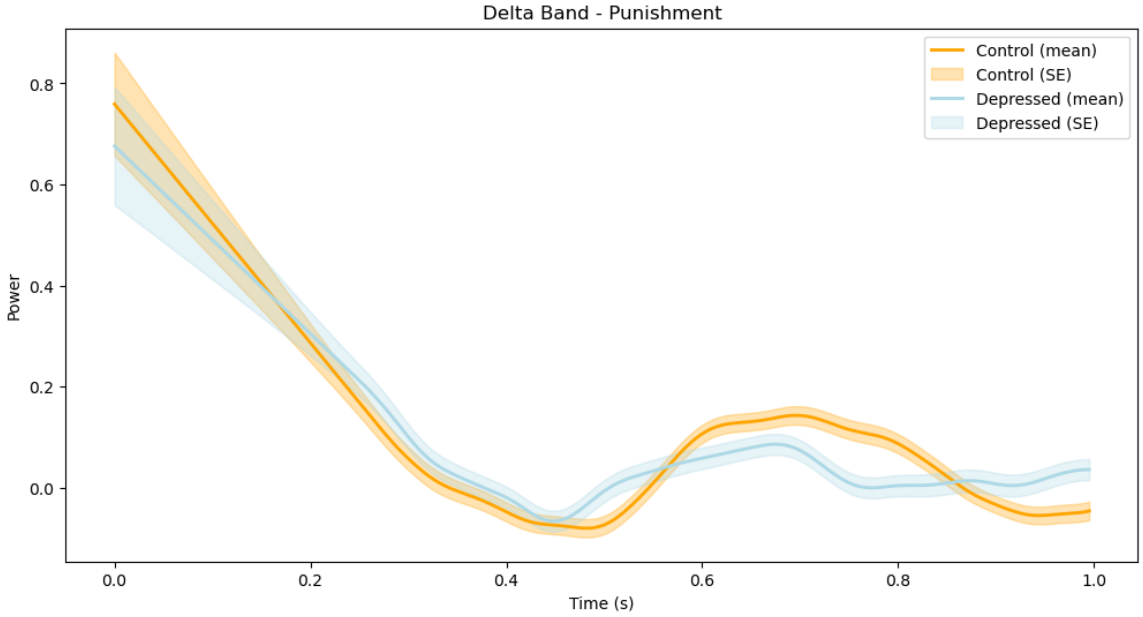
```
    return resampled_powers, common_times

# Modify process_condition_group to use a common time grid
def process_condition_group(epochs, sample_rate, step, common_times):
    """
    Compute mean and SE of power for all frequency bands,
    resampled onto a common grid.
    """
    mean_powers = {}
    se_powers = {}

    for (band, limits), (_, win_siz) in zip(frequency_bands.items(),
        f_windows.items()):
        band_powers, common_times = compute_band_power2(epochs, limits,
            sample_rate, win_siz, step, common_times)
        mean_powers[band] = band_powers.mean(axis=0)  # Mean across epochs
        se_powers[band] =
        band_powers.std(axis=0) / np.sqrt(band_powers.shape[0])
        # SE across epochs

    return mean_powers, se_powers, common_times

# Update analyze_and_plot to define a common grid
def analyze_and_plot(data, sample_rate=500, step=0.005):
    """
    Analyze and plot power changes for each group and
    condition on a common time grid.
    """
    epoch_length =
    data["control"]["reward"][list(data["control"]["reward"].keys())[0]].tmax
    common_times = get_common_time_grid(sample_rate, epoch_length, step)

    for condition in ["reward", "punishment"]:
        for group in ["control", "depressed"]:
            print(f"Processing {condition} - {group}...")
            epochs = extract_epochs(data, group, condition)
            mean_powers, se_powers, times = process_condition_group(
                epochs, sample_rate, step, common_times
            )
            title = f'{condition.capitalize()} - {group.capitalize()}'
            plot_band_changes(times, mean_powers, se_powers, title)

analyze_and_plot(all_data_zscore)

def plot_separate_band_condition(common_times, mean_powers_group,
    se_powers_group):
    """
    Plot mean power and SE for each frequency band
    and condition separately with custom colors.

    Parameters:
    - common_times: Common time grid for all bands
    - mean_powers_group: Dictionary of mean powers for each group and condition
    - se_powers_group: Dictionary of SEs for each group and condition
    """
    colors = {"control": "orange", "depressed": "lightblue"}

```

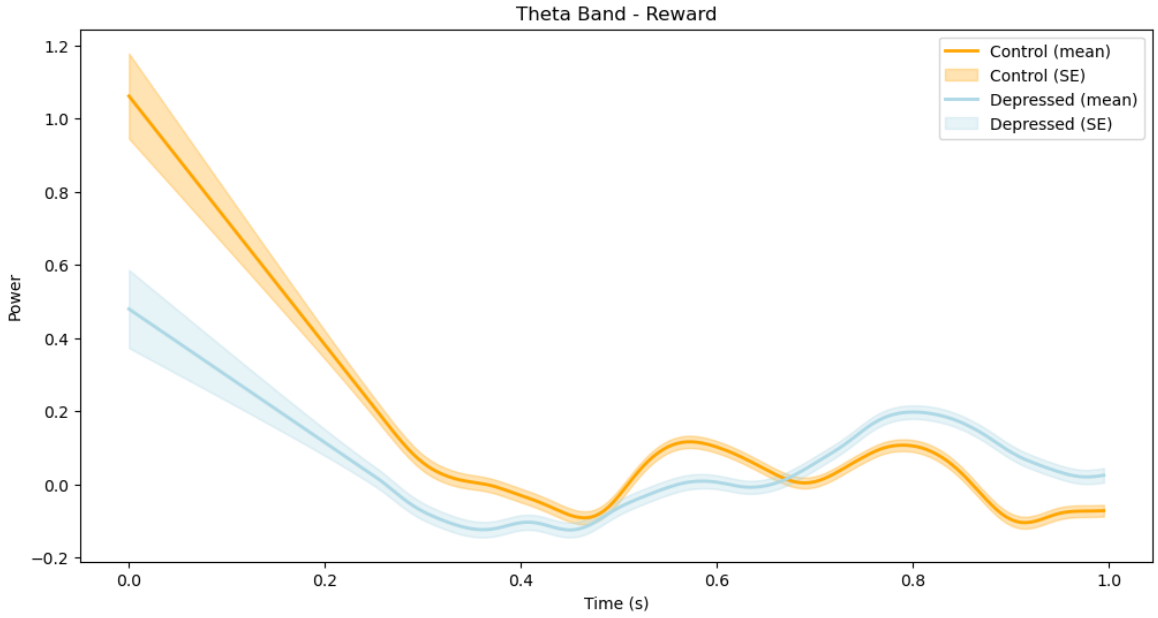
```
    for band in frequency_bands.keys():
        for condition in ["reward", "punishment"]:
            plt.figure(figsize=(12, 6))
            for group in ["control", "depressed"]:
                label = f'{group.capitalize()}'
                mean_power = mean_powers_group[condition][group][band]
                se_power = se_powers_group[condition][group][band]

                plt.plot(
                    common_times,
                    mean_power,
                    label=f'{label} (mean)',
                    linewidth=2,
                    color=colors[group]
                )
                plt.fill_between(
                    common_times,
                    mean_power - se_power,
                    mean_power + se_power,
                    alpha=0.3,
                    color=colors[group],
                    label=f'{label} (SE)'
                )

            plt.title(f'{band} Band - {condition.capitalize()}')
            plt.xlabel('Time (s)')
            plt.ylabel('Power')
            plt.legend()
            plt.show()

# Main analysis function
def analyze_and_plot_by_band(data, sample_rate=500, step=0.005):
    epoch_length =
    data["control"]["reward"][list(data["control"]["reward"].keys())[0]].tmax
    common_times = get_common_time_grid(sample_rate, epoch_length, step)

    # Dictionaries to hold mean and SE powers for all groups and conditions
    mean_powers_group = {condition: {group: {} for group in
        ["control", "depressed"]} for condition in ["reward", "punishment"]}
    se_powers_group = {condition: {group: {} for group in
        ["control", "depressed"]} for condition in ["reward", "punishment"]}

    for condition in ["reward", "punishment"]:
        for group in ["control", "depressed"]:
            print(f"Processing {condition} - {group}...")
            epochs = extract_epochs(data, group, condition)
            mean_powers, se_powers, common_times = process_condition_group(
                epochs, sample_rate, step, common_times
            )
            for band in frequency_bands.keys():
                mean_powers_group[condition][group][band] = mean_powers[band]
                se_powers_group[condition][group][band] = se_powers[band]

    # Generate plots for each band-condition combination
    plot_separate_band_condition(common_times, mean_powers_group,
        se_powers_group)

```

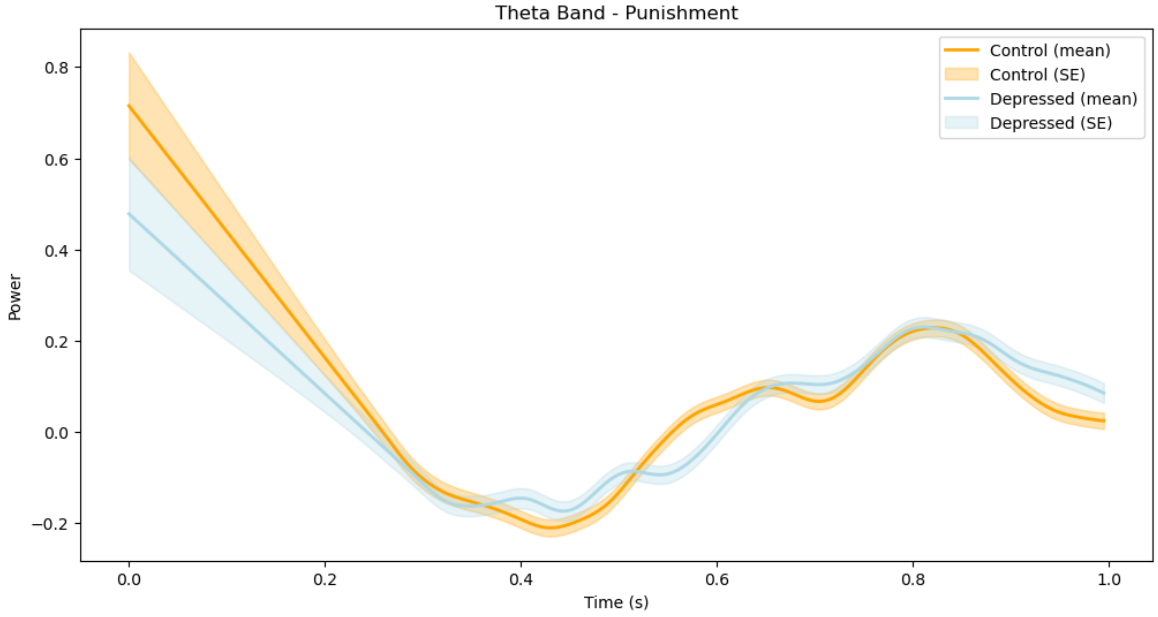
```
309 analyze_and_plot_by_band(all_data_zscore)
```

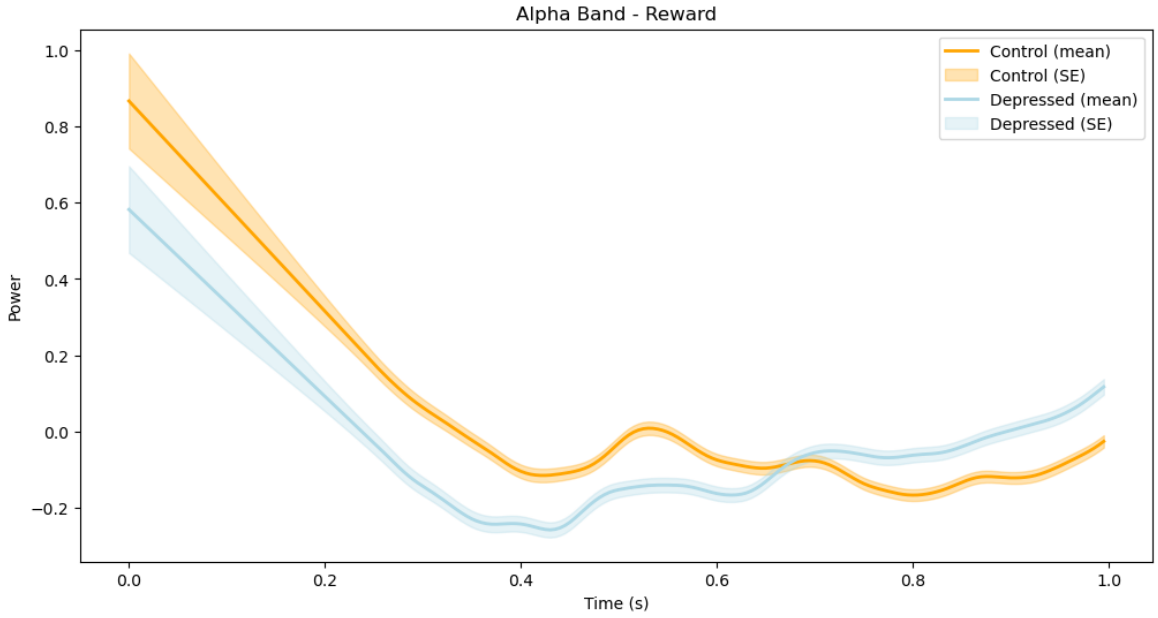
The code produces the following plots:

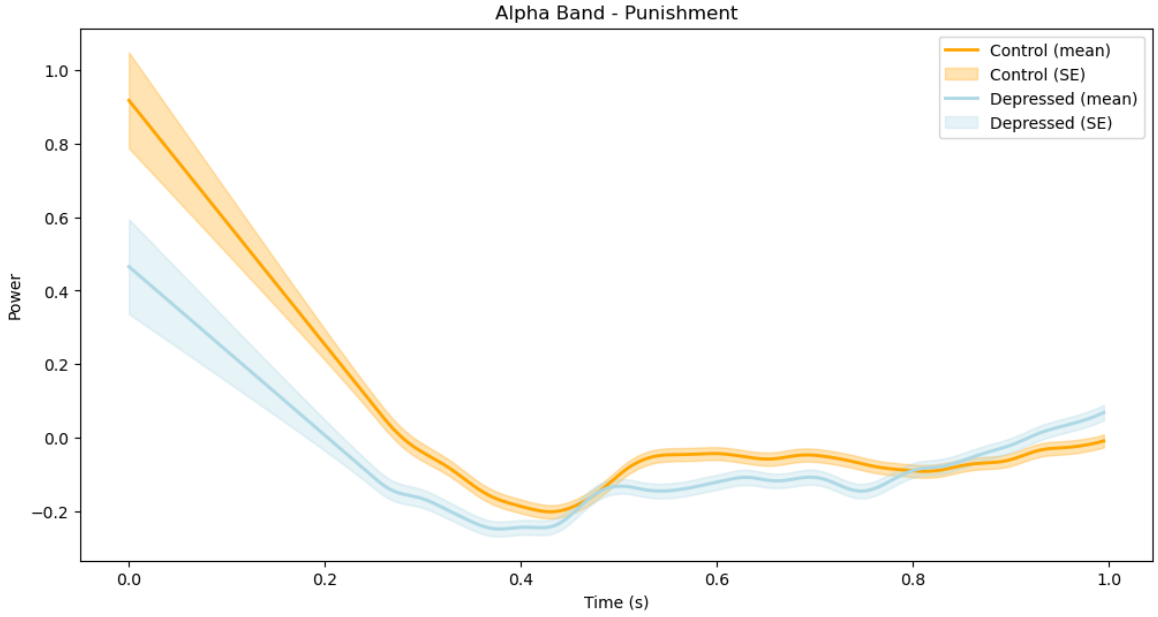


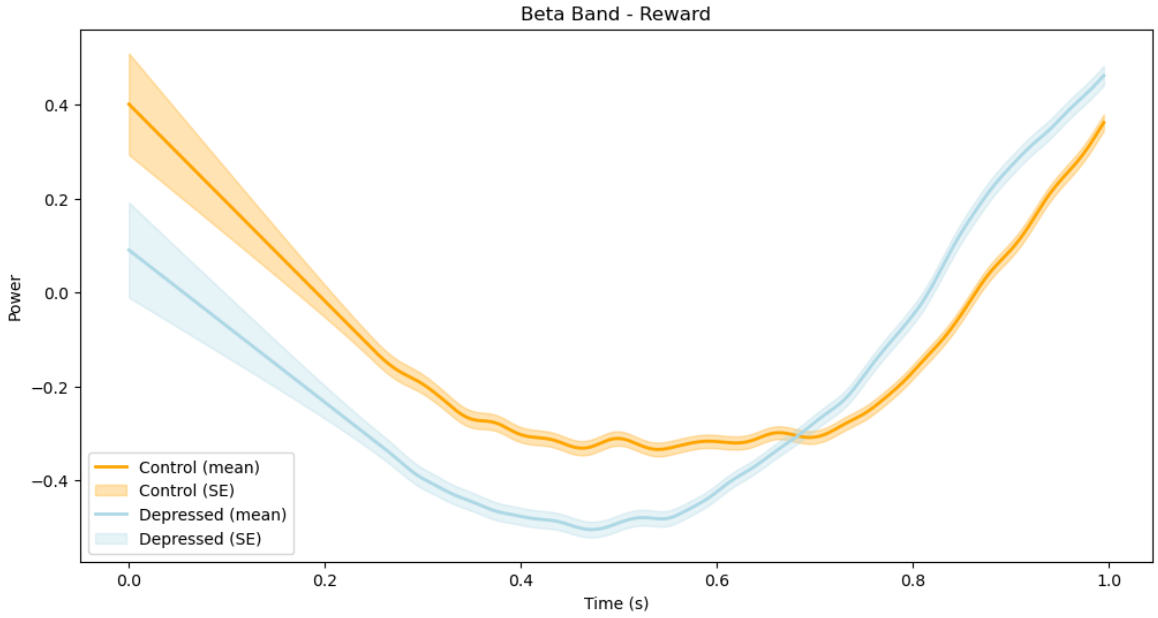


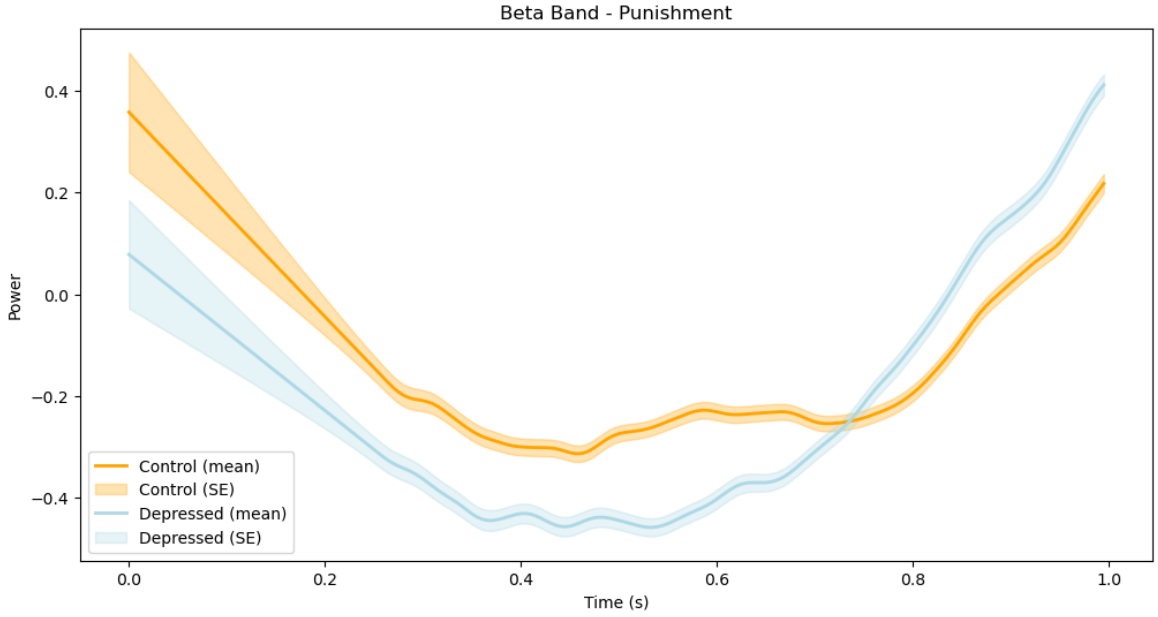

Theta Band - Reward
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)
Power
Time (s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

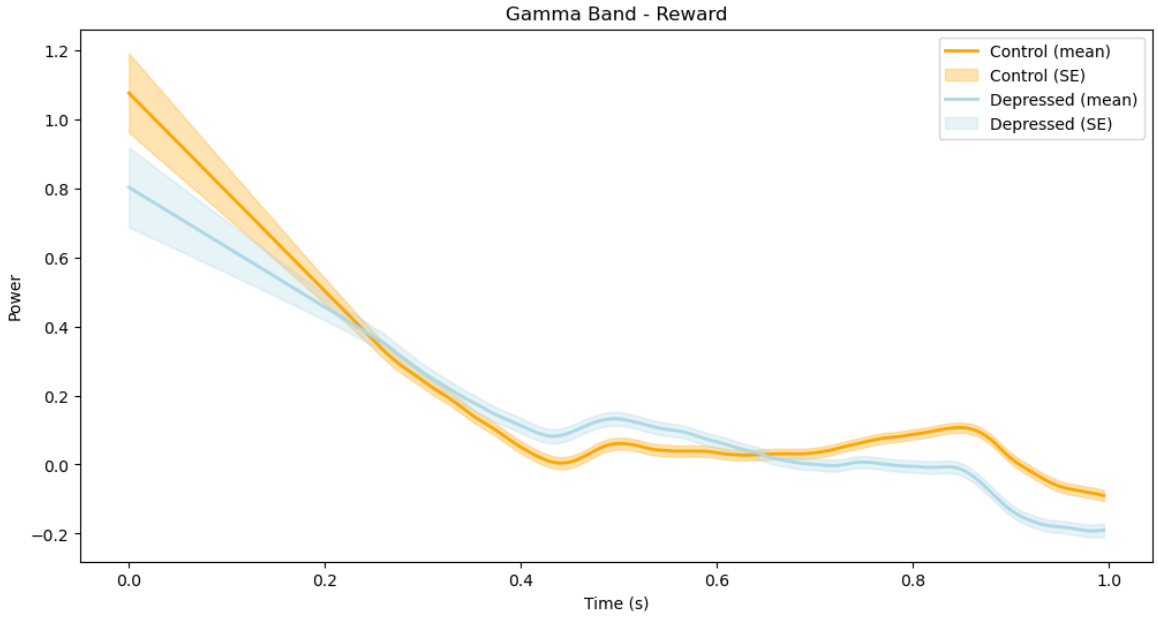

Theta Band - Punishment
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)
Power
Time (s)
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

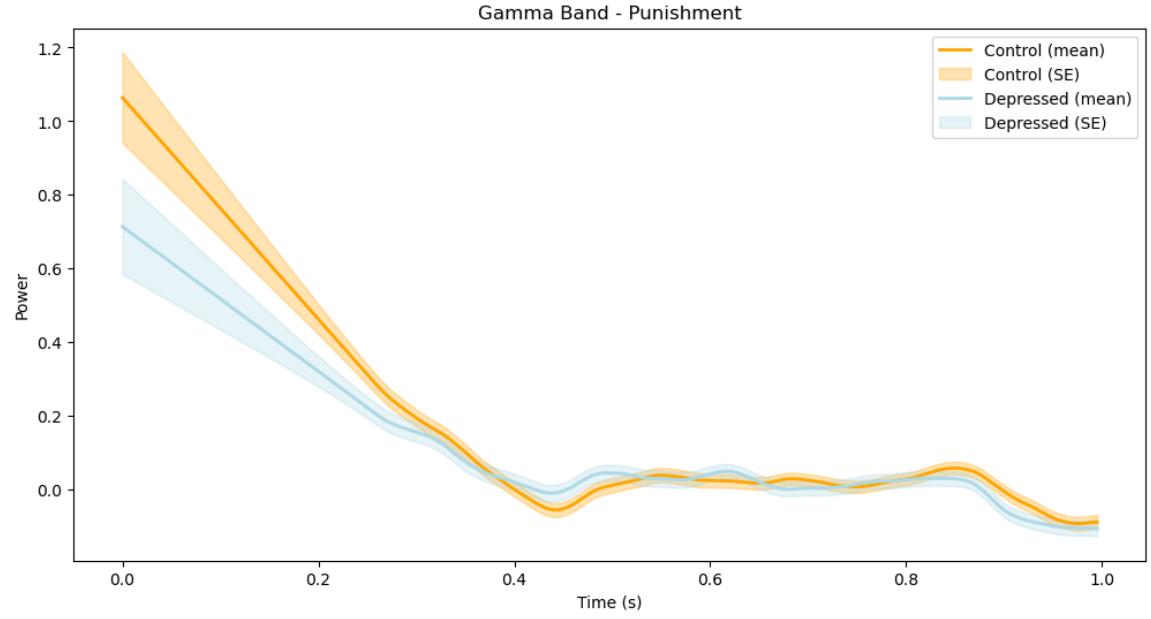

Alpha Band - Reward
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)
Power
Time (s)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Alpha Band - Punishment
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)
Power
Time (s)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


Beta Band - Reward
Power
Time (s)
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)


Beta Band - Punishment
Power
Time (s)
Control (mean)
Control (SE)
Depressed (mean)
Depressed (SE)





As it can be seen the Beta band has the biggest difference in both reward and punishment events. Gamma band looks roughly the same in all of them and alpha band changes more during reward than during punishment. These are consistent with the fact that Beta waves represent active cognition. Again like the ERP analysis we see a delay in the rise of beta band but this time we actully see higher deviation for depressed subjects.

Conclusion

The analysis presented in this report sheds light on how neural responses, as measured through EEG, differ between depressed and control groups across reward and punishment conditions. By employing techniques

like ERP analysis and Short-Time Fourier Transform (STFT), we gained valuable insights into both the temporal dynamics and frequency-specific characteristics of these responses.

Event-Related Potential (ERP) Analysis

- ERPs revealed that the depressed group consistently exhibited lower amplitudes and delayed responses compared to the control group. This pattern was observed in both reward and punishment conditions, indicating a general reduction in neural sensitivity to stimuli.
- The early time windows, typically linked to automatic and rapid processing of stimuli, showed more pronounced differences, with the control group demonstrating sharper and more distinct peaks. This suggests diminished motivational and affective processing in the depressed group.

Frequency Band Analysis with STFT

- The Beta band demonstrated the most significant differences between groups in both reward and punishment conditions. This aligns with its association with active cognition and suggests reduced engagement in the depressed group.
- The Gamma band showed minimal variation across conditions, indicating its limited involvement in these tasks.
- The Alpha band exhibited more substantial changes during reward than punishment, particularly in the control group, highlighting its role in attentional and emotional processing.
- Notably, the depressed group showed not only delayed activation but also higher variability in Beta power, providing additional evidence of altered neural dynamics.

Combined Insights

- Across both analyses, the depressed group consistently showed blunted responses, flatter waveforms, and higher variability, reflecting diminished neural activity and sensitivity to both positive and negative stimuli.
- The control group exhibited clear distinctions between reward and punishment responses, with stronger, more distinct patterns of activity, especially in the Beta and Alpha bands.