Part 1

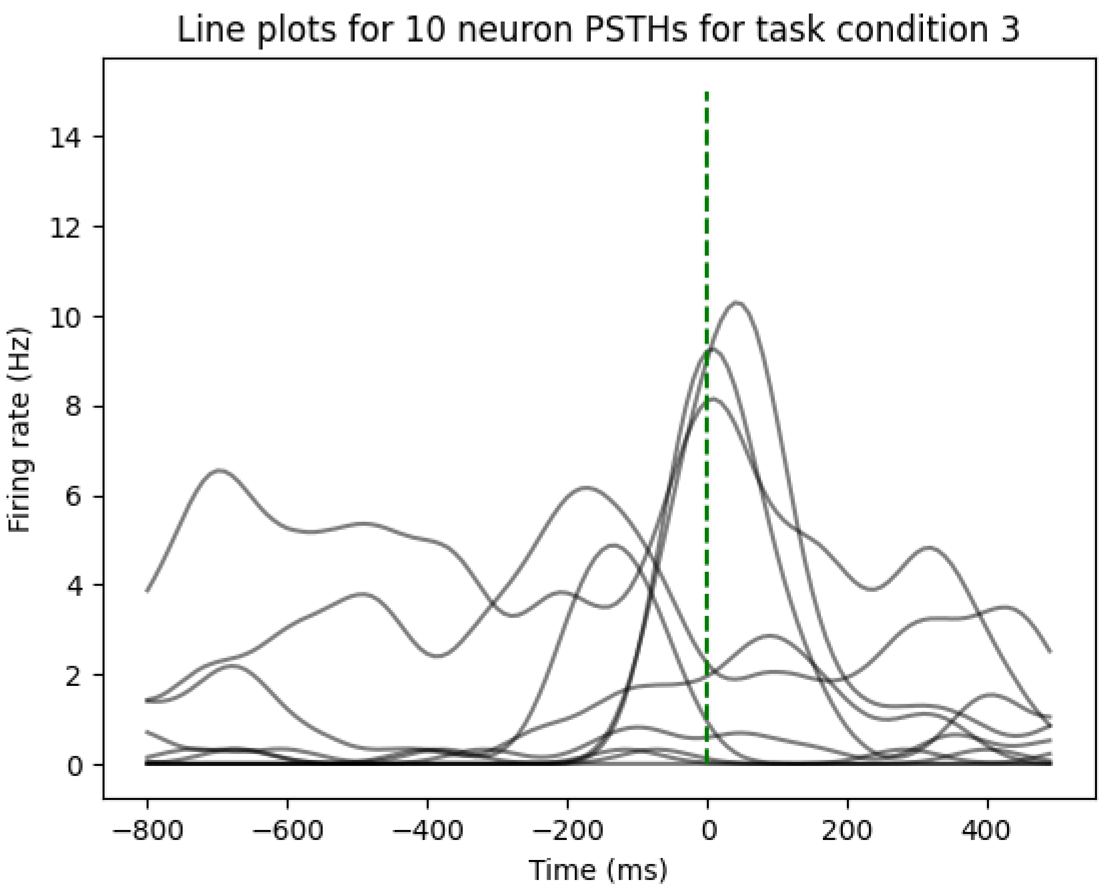
In [648…

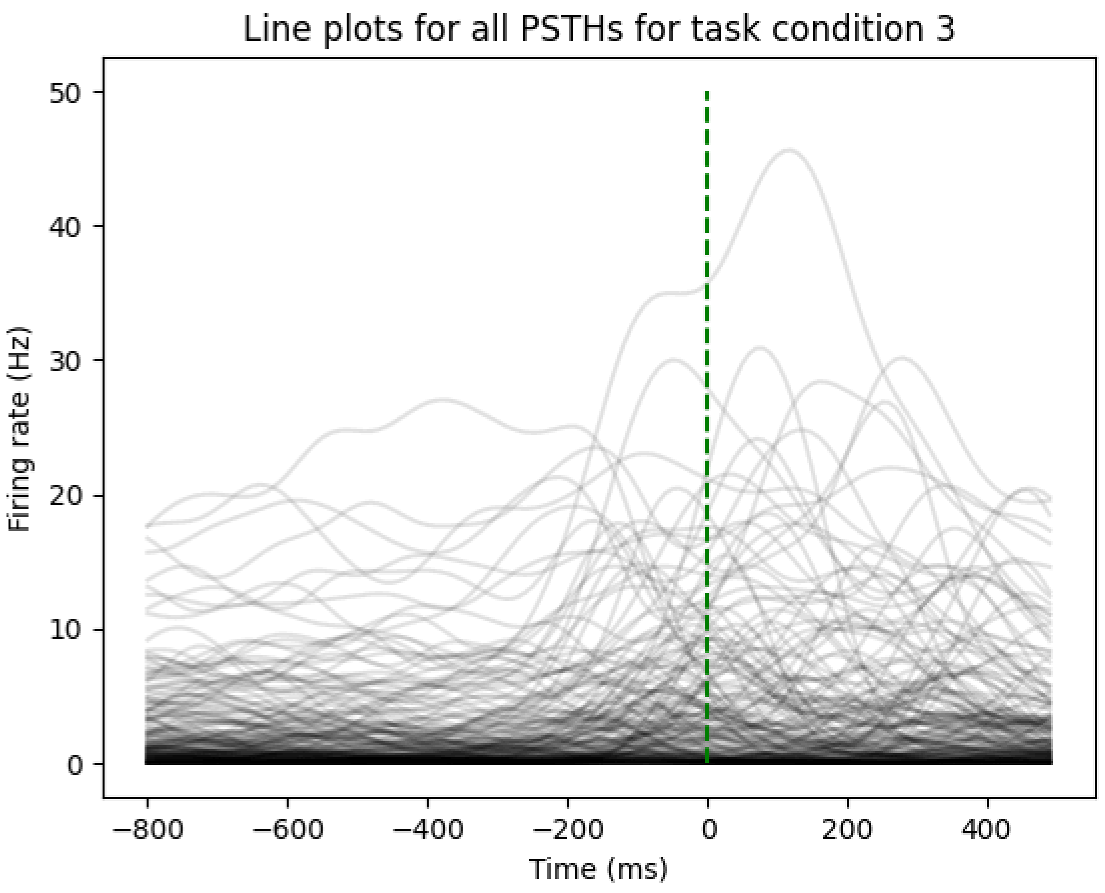
```
import numpy as np
from data import cond_color
from matplotlib import pyplot as plt


psths = np.load('data/psths.npz')
X, times = psths['X'], psths['times']   # X is a 3D array of shape (n_neurons, n_co

print('Shape for X', np.shape(psths['X']))
print('Shape for times', np.shape(psths['times']))
```

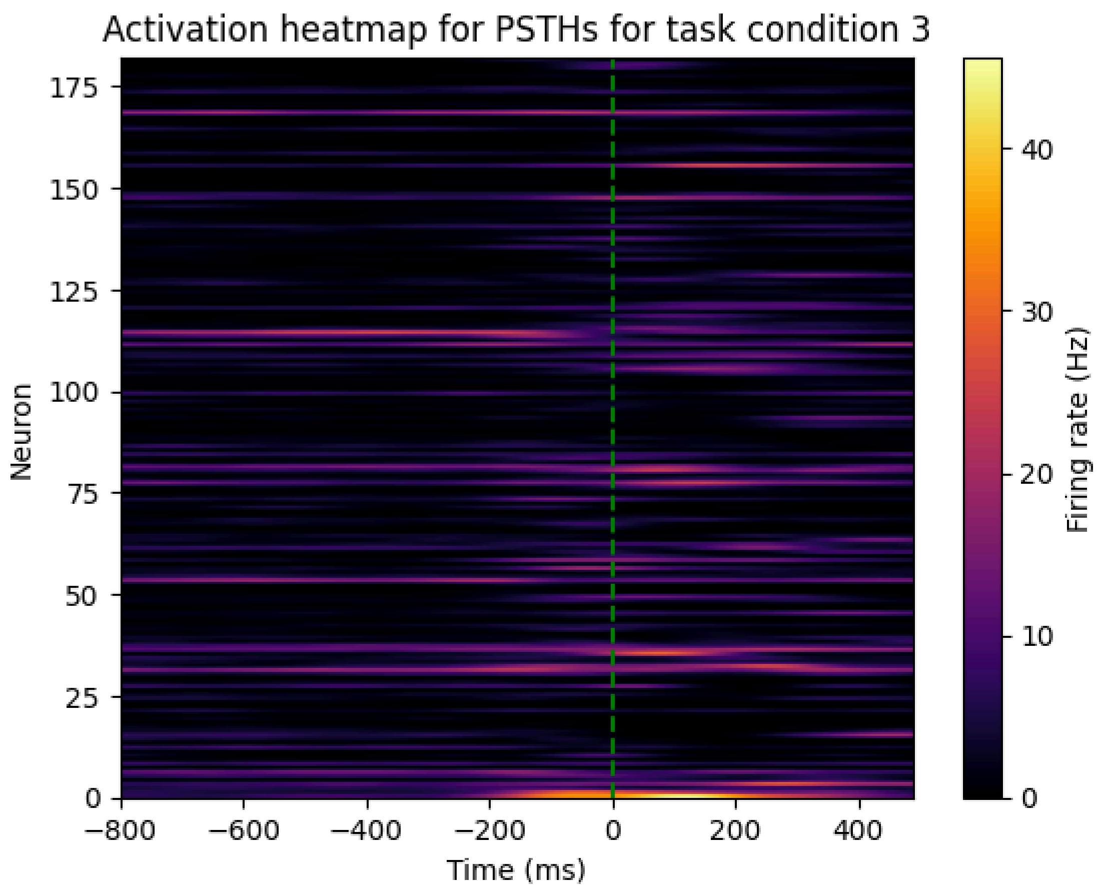
```
Shape for X (182, 108, 130)
Shape for times (130,)
```

In [649…

```
# Some sample plots from X

task_condition = 3
average_slice = np.mean(X, axis=1)
slice_data = X[:, task_condition, :]

plt.plot(times, slice_data.T[:, :10], color='black', alpha=0.5)
plt.title('Line plots for 10 neuron PSTHs for task condition {}'.format(task_condit
plt.vlines(0, 0, 15, color='green', linestyle='--')
plt.xlabel('Time (ms)')
plt.ylabel('Firing rate (Hz)')
plt.show()

plt.plot(times, slice_data.T, color='black', alpha=0.1)
plt.title('Line plots for all PSTHs for task condition {}'.format(task_condition))
plt.vlines(0, 0, 50, color='green', linestyle='--')
plt.xlabel('Time (ms)')
plt.ylabel('Firing rate (Hz)')
plt.show()

# Now plotting with Imshow

plt.imshow(slice_data, aspect='auto', cmap='inferno', extent=[times[0], times[-1],
plt.title('Activation heatmap for PSTHs for task condition {}'.format(task_conditio
plt.vlines(0, 0, slice_data.shape[0], color='green', linestyle='--')
plt.xlabel('Time (ms)')
plt.ylabel('Neuron')
plt.colorbar(label='Firing rate (Hz)')
plt.show()

print(slice_data.shape)
```


Line plots for 10 neuron PSTHs for task condition 3
Firing rate (Hz)
Time (ms)
0
2
4
6
8
10
12
14
−800
−600
−400
−200
0
200
400


Line plots for all PSTHs for task condition 3
Firing rate (Hz)
Time (ms)
0
10
20
30
40
50
−800
−600
−400
−200
0
200
400

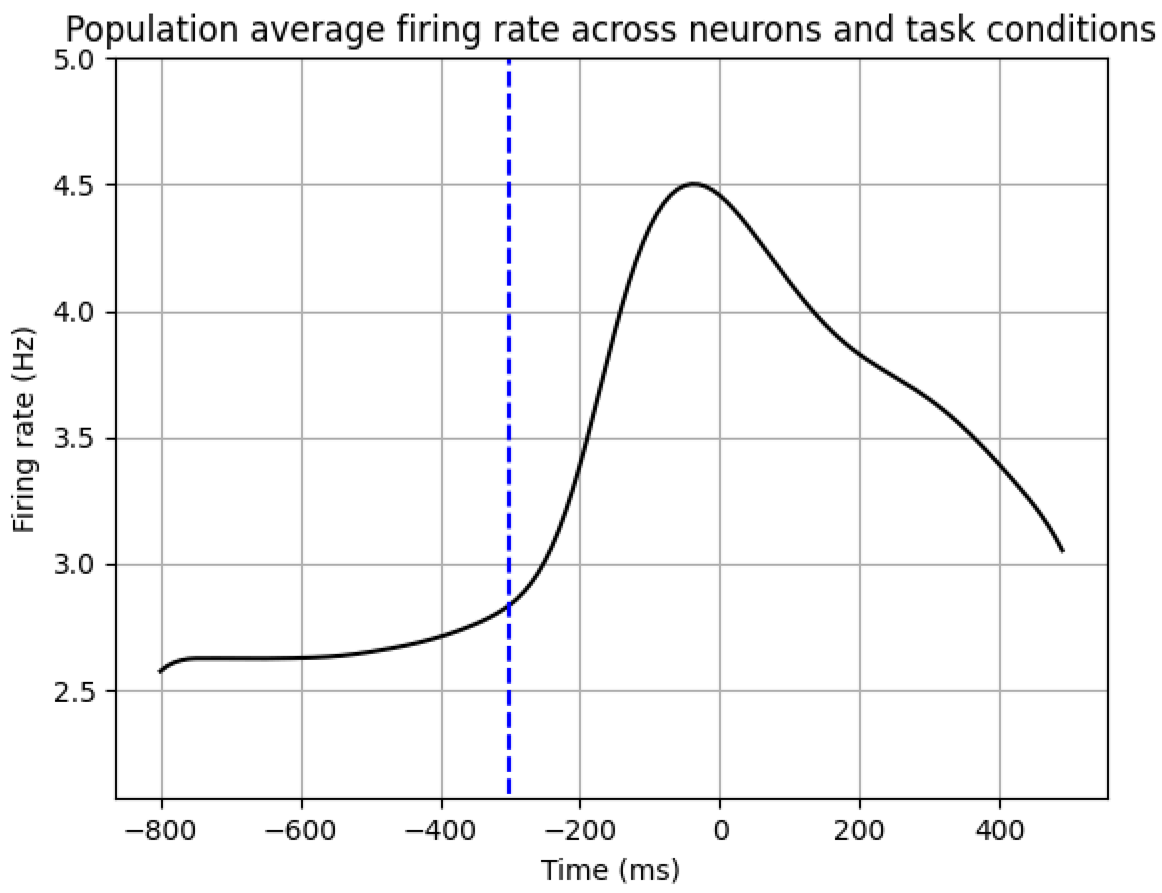


```
(182, 130)
```

In [650…

```
# Plot population average firing rate across neurons and task conditions

mean_firing_rate = np.mean(np.mean(X, axis=0), axis=0)
print('Shape for mean_firing_rate', np.shape(mean_firing_rate))

plt.plot(times, mean_firing_rate, color='black')
plt.xlabel('Time (ms)')
plt.ylabel('Firing rate (Hz)')
plt.ylim(min(mean_firing_rate) - 0.5, max(mean_firing_rate) + 0.5)
plt.title('Population average firing rate across neurons and task conditions')
plt.vlines(-300, 2, 5, color='blue', linestyle='--')
plt.grid()
plt.show()
```

```
Shape for mean_firing_rate (130,)
```



> Q: What qualitative differences do you notice in the behaviour of PSTH’s in the pre-movement period vs. during or just before hand movement? Plot also the population average firing rate as a function of time, obtained by taking the average of the PSTHs across neurons and conditions.

A: From qualititative analysis of the line plots and activation heatmaps - there is a marked difference between the PSTHs after the go cue. Well before the go cue, the neuron activity is stagnant, and somewhat constant. Individual neurons are either in a constant 'on' or 'off' state (corresponding to consistent firing or no firing). About 400 ms before the cue there is a transient stage where neurons change their firing rates. This transience is held until about 200 ms after the go cue after which there is another period of neurons stabilising (presumably as the activity being peformed is constant in that time frame - e.g. moving in a straight line to target).

The plots are made first with reference to a specific task-condition, then the average firing rate is plotted across tasks and across neurons.

> Q: At what time point (roughly), relative to the movement onset, does this mean rate start to rise significantly above its baseline level (i.e., the approximate firing rate values between, say,-800ms and-600ms, well before the movement onset)? Provide a possible explanation.

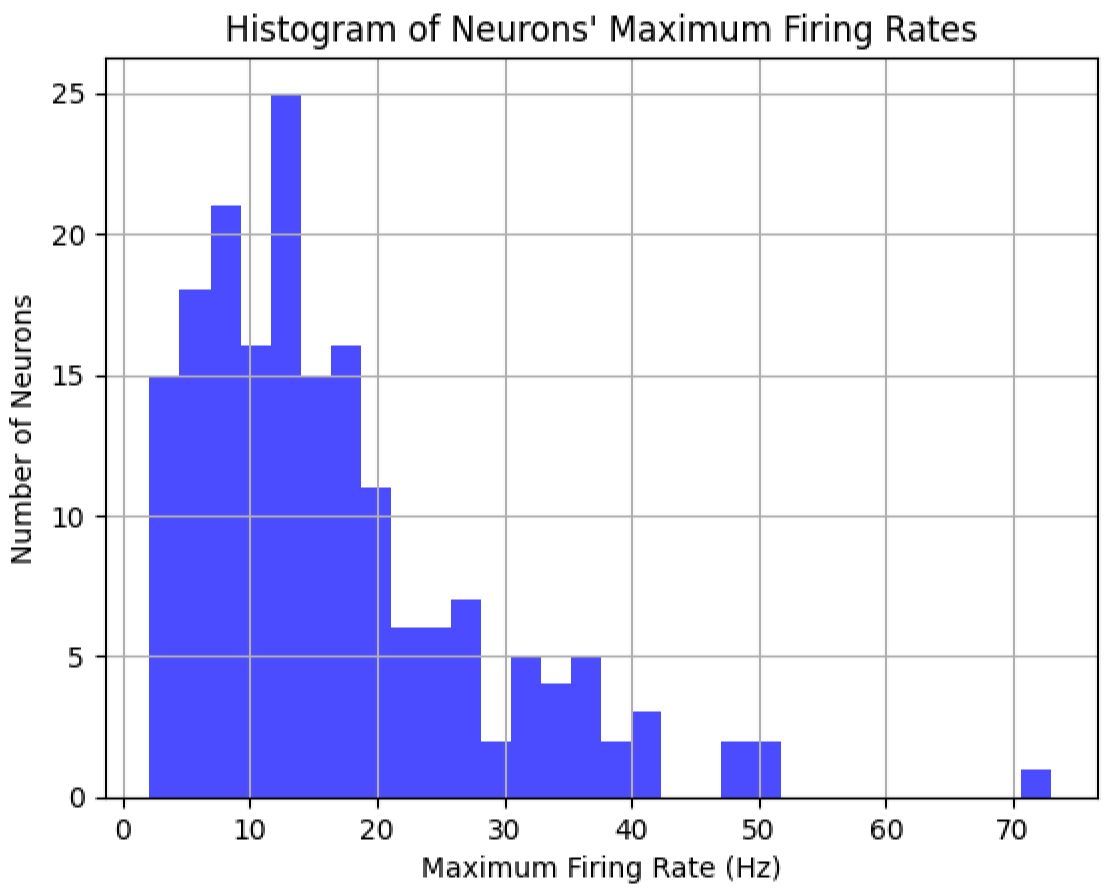
From qualitiative analysis - though this can be asserted by using something like an edge-identifying filter like a LoG - around -300 ms is the time when neuron activity changes significantly. This is interesting as it suggests non-causality between the timing of the go cue and the neuron activity. LDS models are best suited for causal inference. If one is to use the model to describe neuron activity, the non-causality can be attributed to mismatched calibration of the measuring instrument. More likely though, the monkey will be anticipating the go cue - and in expectation of the reward, neurons may begin firing well before the actual command. During training, the monkey could have picked up on cues that would indicate that the go cue is about to be given and the training would have induced this anticipation.

Part 2

In [651…

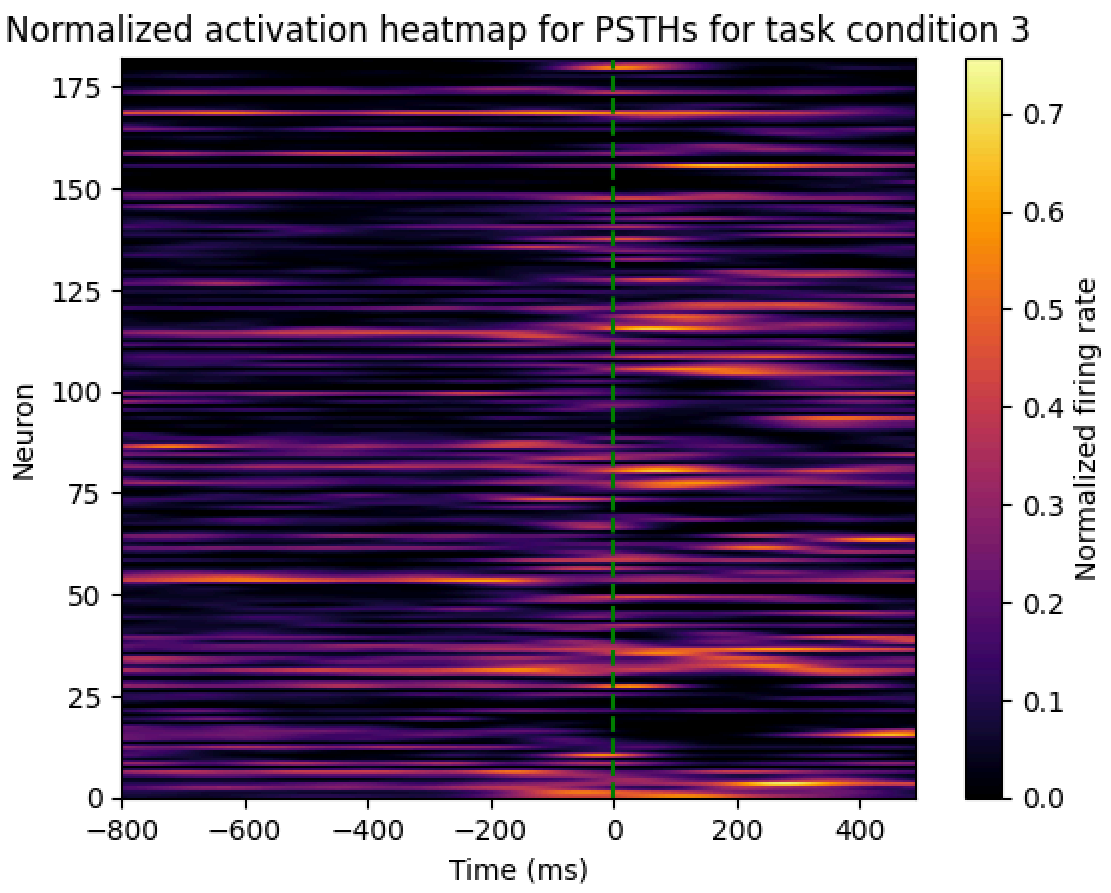
```
# Calculate the maximum firing rate for each neuron across conditions and time
max_firing_rates = np.max(X, axis=(1, 2))
min_firing_rates = np.min(X, axis=(1, 2))

print('Shape of max_firing_rates', np.shape(max_firing_rates))
print('Shape of min_firing_rates', np.shape(min_firing_rates))

# Plot the histogram
plt.hist(max_firing_rates, bins=30, color='blue', alpha=0.7)
plt.title('Histogram of Neurons\' Maximum Firing Rates')
plt.xlabel('Maximum Firing Rate (Hz)')
plt.ylabel('Number of Neurons')
plt.grid()
plt.show()
```

```
Shape of max_firing_rates (182,)
Shape of min_firing_rates (182,)
```



In [652…

```
# Normalizing the spike firing data accoridng to min and max
# (a)

X_normalized = (X - min_firing_rates[:, None, None]) / (max_firing_rates[:, None, N
slice_normalized = X_normalized[:, task_condition, :]

plt.imshow(slice_normalized, aspect='auto', cmap='inferno', extent=[times[0], times
plt.title('Normalized activation heatmap for PSTHs for task condition {}'.format(ta
plt.vlines(0, 0, slice_normalized.shape[0], color='green', linestyle='--')
plt.xlabel('Time (ms)')
plt.ylabel('Neuron')
plt.colorbar(label='Normalized firing rate')
plt.show()
```

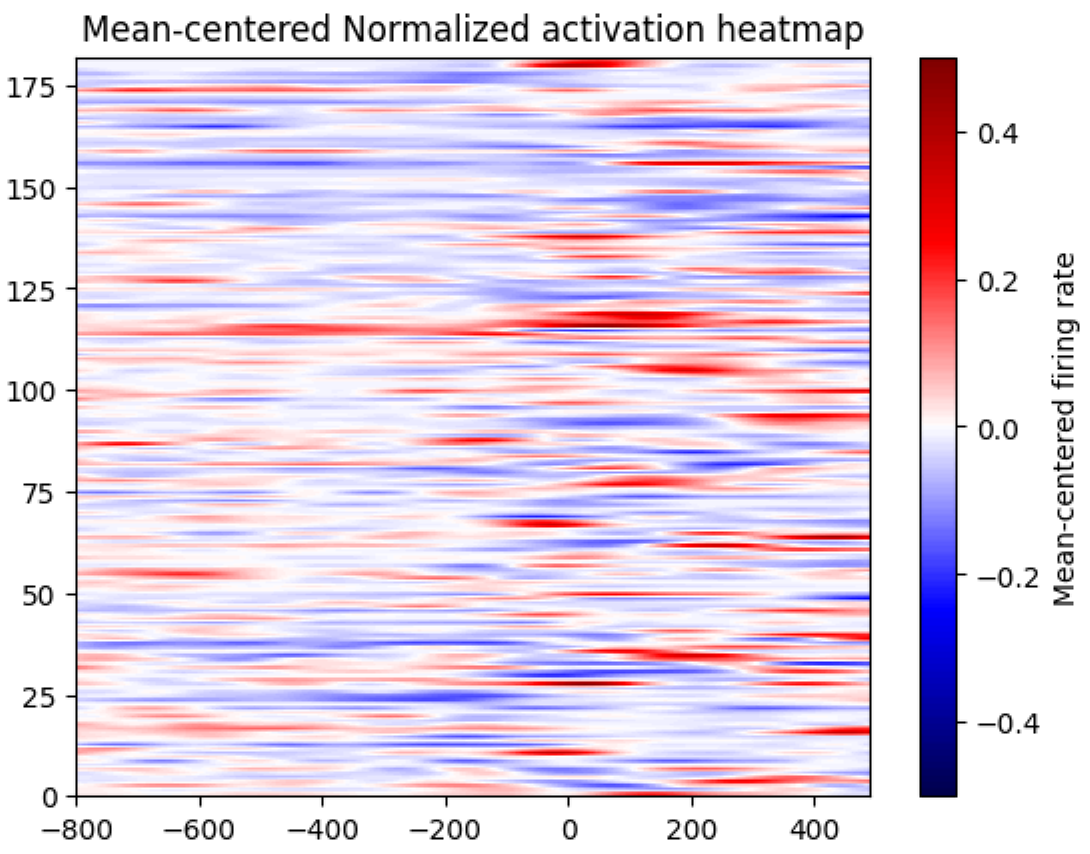


> Q: Why do you think this normalization step will be helpful? (You can come back to this question after going through all exercises.)

A: TBA (NOTE)

In [653…

```
# Mean centering (removing cross condition mean)
# (b)

X_mean_centered = X_normalized - np.mean(X_normalized, axis=1)[:, None, :]
slice_mean_centered = X_mean_centered[:, task_condition, :]
plt.imshow(slice_mean_centered, aspect='auto', cmap='seismic', extent=[times[0], ti
plt.colorbar(label='Mean-centered firing rate')
plt.title('Mean-centered Normalized activation heatmap')
plt.show()
```



In [654…

```
start_index = np.where(times == -150)[0][0]
end_index = np.where(times == 300)[0][0]

X_new = X_mean_centered[:, :, start_index:end_index+1]
times_new = times[start_index:end_index + 1]# Only keep the time points between -15
X_new = np.reshape(X_new, (X_new.shape[0], -1))
print('Shape of X_new', np.shape(X_new))
```

```
Shape of X_new (182, 4968)
```

In [655…

```
# Project the data to the first 12 principal components
#S = np.sum([np.outer(x, x) for x in X_new])

from sklearn.decomposition import PCA
import tqdm


S = np.matmul(X_new, X_new.T)
S = S / X_new.shape[0]
Z_sk = PCA(n_components=12).fit_transform(X_new.T).T
print('Shape of S', np.shape(S))
print('Shape of Z_sk', np.shape(Z_sk))
```

```
Shape of S (182, 182)
Shape of Z_sk (12, 4968)
```

In [656…

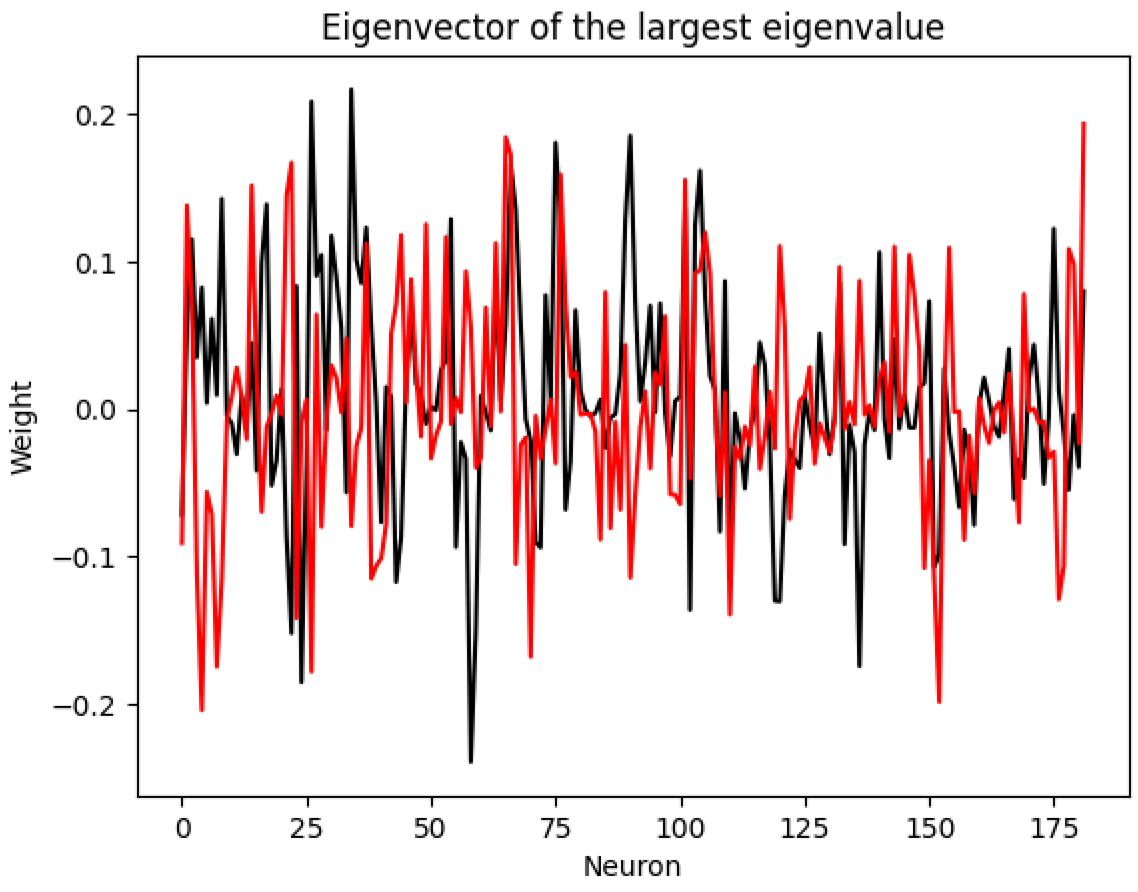
```
# Calculating eigenvalues and eigenvectors
eigvals, eigvecs = np.linalg.eigh(S)
print('Shape of eigvals', np.shape(eigvals))
print('Shape of eigvecs', np.shape(eigvecs))
```

```
plt.plot(eigvecs[:, -1], color='black')
plt.plot(eigvecs[:, -2], color='red')
plt.xlabel('Neuron')
plt.ylabel('Weight')
plt.title('Eigenvector of the largest eigenvalue')
```

```
Shape of eigvals (182,)
Shape of eigvecs (182, 182)
```

Out[656…

```
Text(0.5, 1.0, 'Eigenvector of the largest eigenvalue')
```



In [657…

```
M = 12
print(np.shape(eigvals), np.shape(eigvecs))
mle_var = np.mean(eigvals[:-M])

print('MLE sigma for {} PCA components'.format(M), mle_var)

l_array = [np.sqrt(eig - mle_var) for eig in eigvals[-M:]]
print('Shape of l_array', np.shape(l_array))
print(l_array)


V_m = eigvecs[:, -M:]
print('Shape of V_m', np.shape(V_m))
L = np.diag(l_array)
print('Shape of L', np.shape(L))
C = np.matmul(V_m, L)
```

```
print('Shape of C', np.shape(C))
```

```
(182,) (182, 182)
MLE sigma for 12 PCA components 0.1180841669436586
Shape of l_array (12,)
[0.9620902208545898, 1.1015114601886355, 1.178633044908589, 1.279904849058771, 1.317
503583095343, 1.4252285036445065, 1.5648830863329113, 1.6047398126558317, 1.70858575
78588999, 2.0659460954680156, 2.2697127561147727, 2.7218746892457135]
Shape of V_m (182, 12)
Shape of L (12, 12)
Shape of C (182, 12)
```

In [658…

```
# Projecting into M space
print('Shape of X_new', np.shape(X_new))
Z_pca = np.matmul(V_m.T, X_new)
print('Shape of Z_pca', np.shape(Z_pca))

np.mean(abs(Z_pca), axis=1)
```

```
Shape of X_new (182, 4968)
Shape of Z_pca (12, 4968)
```

Out[658…

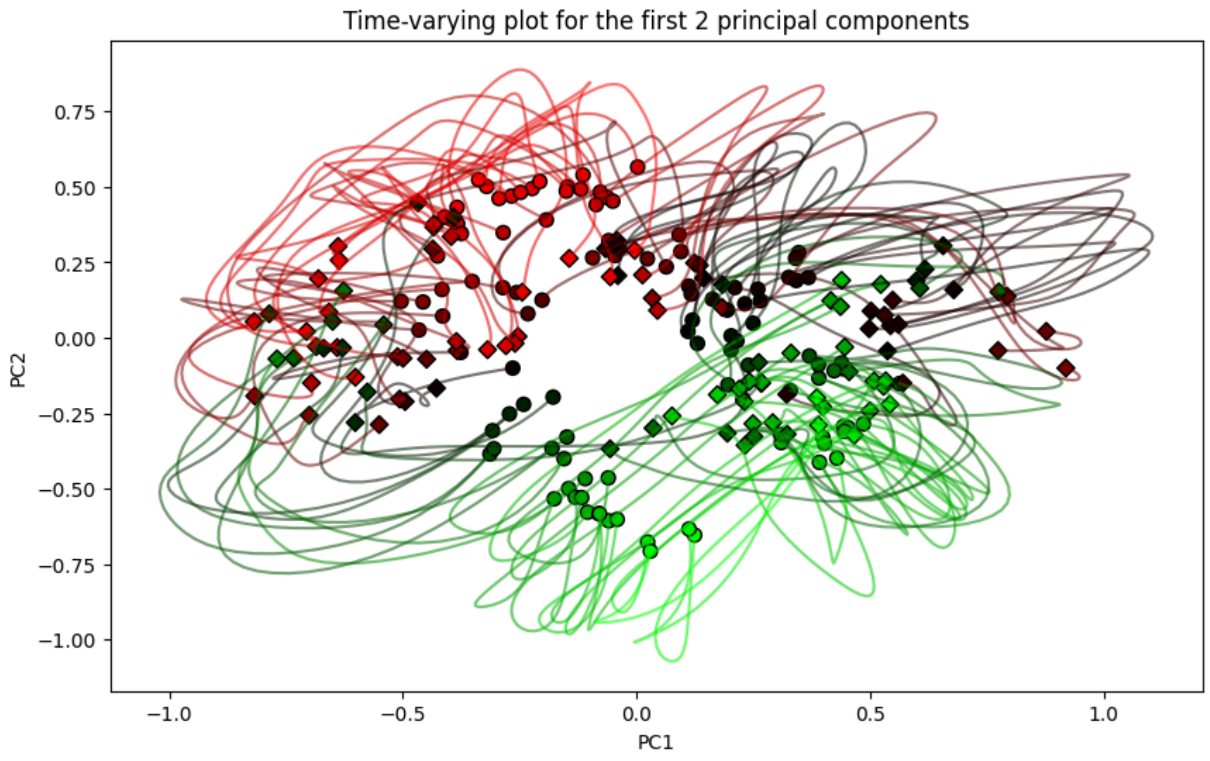
```
array([0.15276846, 0.17298975, 0.18847616, 0.20617637, 0.21813384,
       0.23300791, 0.2491839 , 0.25737574, 0.27382417, 0.31829943,
       0.37512896, 0.45387445])
```

Part 3

In [659…

```
# Making the time-varying plot for the first 2 principal components
from data import cond_color
import numpy as np



pc1 = Z_pca[-1].reshape(-1, len(times_new))
pc2 = Z_pca[-2].reshape(-1, len(times_new))
print('Shape of pc1', np.shape(pc1))

fig, ax = plt.subplots(figsize=(10, 6))

c_colors = cond_color.get_colors(pc1[:, 0], pc2[:, 0])
cond_color.plot_start(pc1[:, 0], pc2[:, 0], c_colors, 200, ax)
cond_color.plot_end(pc1[:, -1], pc2[:, -1], c_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, c_colors):
    ax.plot(xs, ys, color=c, alpha=0.55)

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Time-varying plot for the first 2 principal components')
plt.show()
```

```
Shape of pc1 (108, 46)
```



Part 4

In [660…

```python
# Finding gradient with respect to matrix A

print('Shape of Z_pca', np.shape(Z_pca))

Z_pca_reshape = Z_pca.reshape(12, X.shape[1], -1)
print('Shape of Z_pca_reshape', np.shape(Z_pca_reshape))

delta_Z_reshape = np.diff(Z_pca_reshape, axis=2)
print('Shape of delta_Z_reshape', np.shape(delta_Z_reshape))

Z_pca_reshape = np.delete(Z_pca_reshape, 0, axis=2)
Z_pca = Z_pca_reshape.reshape(12, -1)

delta_Z = delta_Z_reshape.reshape(12, -1)

Z = Z_pca
delta_Z = delta_Z
print('_'*100)
print('Shape of Z', np.shape(Z))
print('Shape of delta_Z', np.shape(delta_Z))
```

```
Shape of Z_pca (12, 4968)
Shape of Z_pca_reshape (12, 108, 46)
Shape of delta_Z_reshape (12, 108, 45)
____________________________________________________________________________________
________________
Shape of Z (12, 4860)
Shape of delta_Z (12, 4860)
```

In [661…

```python
def construct_H(m = 12):
    # Constructing the Beta array

    H = []

    row, col = 0, 1
    for i in range(m*(m-1)//2):

        H_a = np.zeros((m, m))
        H_a[row, col] = 1
        H_a[col, row] = -1

        H.append(list(H_a))

        col += 1
        if col == m:
            row += 1
            col = row + 1

    return H


def reconstruct_A(H, b):
    return np.tensordot(b, H, axes=1)
```

In [662…

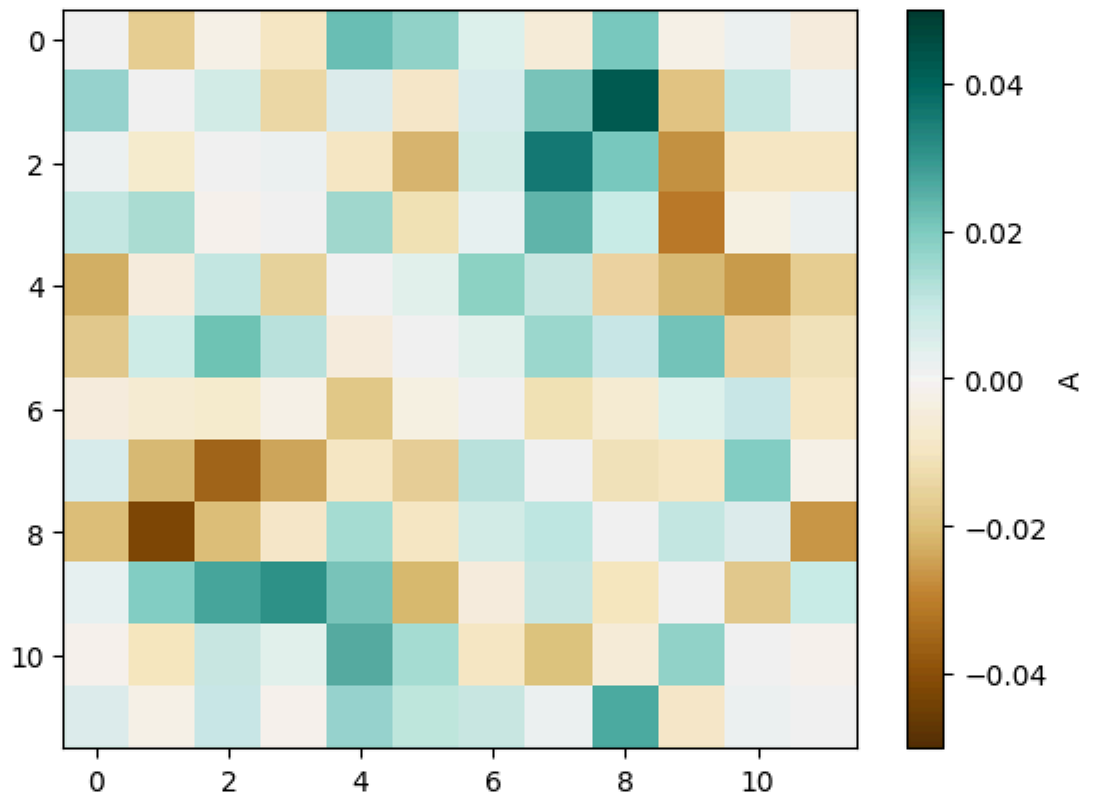
```python
# Taking derivative of the loss function with respect to A
# An important note here is that W is constructed with Z (with last shape T) which
def gen_W(H:np.ndarray, Z):
    return np.tensordot(H, Z, axes = 1)


def gen_bQ(W, delta_Z):
    print('Shape of W', np.shape(W))
    print('Shape of Z', np.shape(delta_Z))
    b_vec = np.tensordot(W, delta_Z, axes=([2, 1], [1, 0]))
    print('Shape of b_vec', np.shape(b_vec))
    Q = np.tensordot(W, W, axes = ([2, 1], [2, 1]))
    print('Shape of Q', np.shape(Q))

    return b_vec, Q

H = construct_H(12)
W = gen_W(H, Z)
b_vec, Q = gen_bQ(W, delta_Z)

beta_pred = np.linalg.inv(Q).dot(b_vec)
print('Shape of beta_pred', np.shape(beta_pred))

A_pred = reconstruct_A(H, beta_pred)
```

```
Shape of W (66, 12, 4860)
Shape of Z (12, 4860)
Shape of b_vec (66,)
Shape of Q (66, 66)
Shape of beta_pred (66,)
```

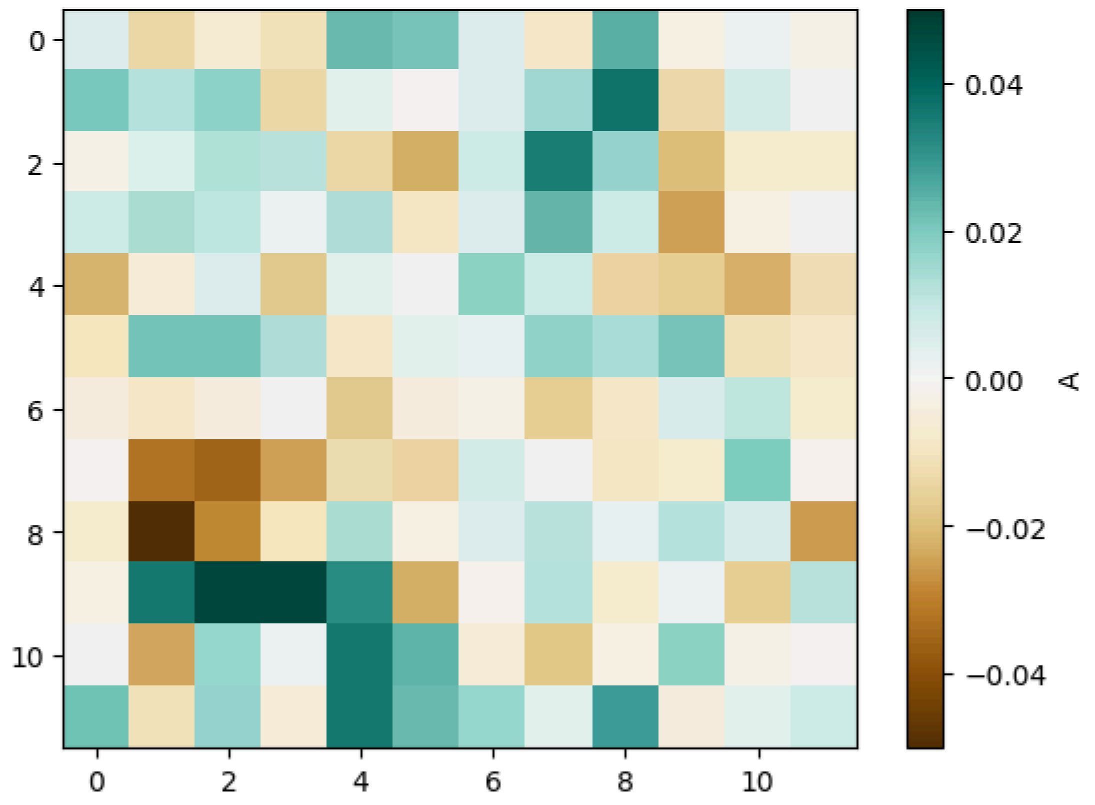
In [670…

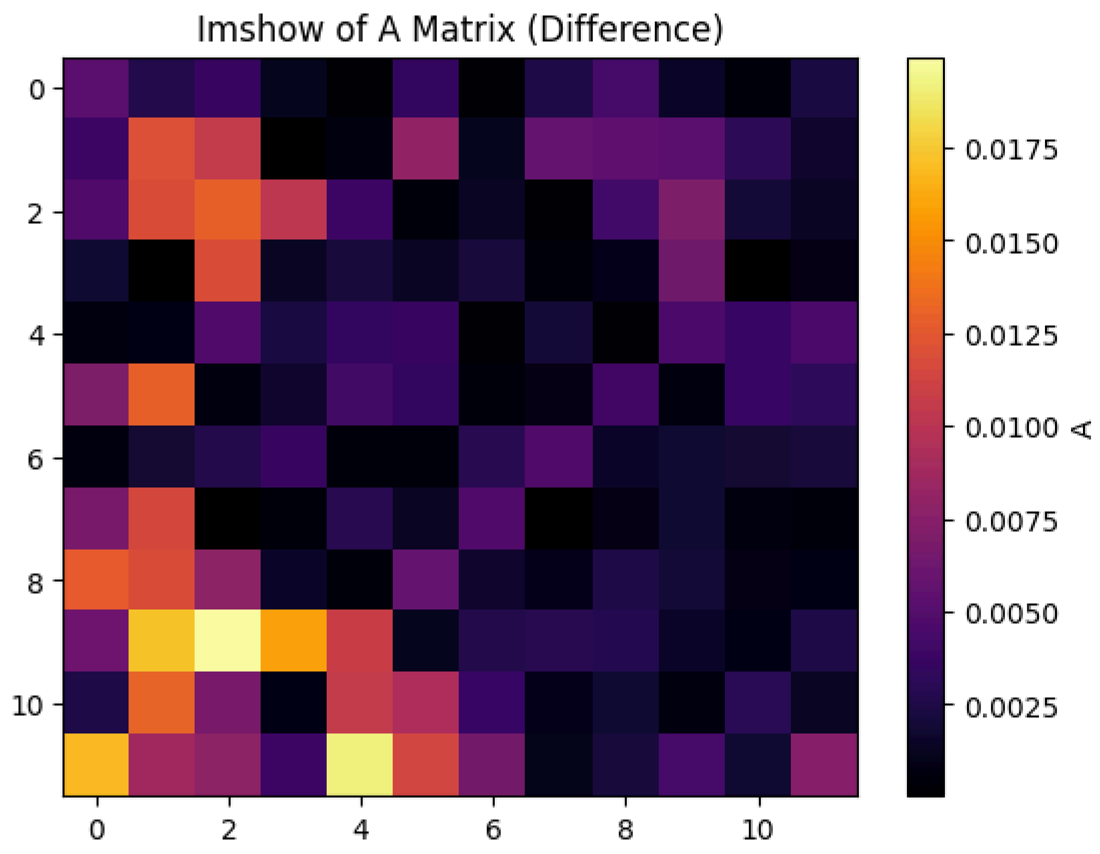
```
# Imshow of A

plt.imshow(A_pred, aspect='auto', cmap='BrBG', interpolation='nearest', vmin=-0.05,
plt.colorbar(label='A')
plt.title('Imshow of A Matrix')
plt.show()

A_rotational  = A_pred
A_mle  = np.matmul(np.matmul(delta_Z, Z.T), np.linalg.inv(np.matmul(Z, Z.T)))

plt.imshow(A_mle, aspect='auto', cmap='BrBG', interpolation='nearest', vmin=-0.05,
plt.colorbar(label='A')
plt.title('Imshow of A Matrix (MLE)')
plt.show()

A_diff  = np.abs(A_rotational - A_mle)
plt.imshow(A_diff, aspect='auto', cmap='inferno', interpolation='nearest')
plt.colorbar(label='A')
plt.title('Imshow of A Matrix (Difference)')
plt.show()

LL_mle  = 0
LL_rotational  = 0

for i in range(Z.shape[1]):
    if i%500 == 0:
        print('Processing for ', i)
    LL_mle += -0.5 * np.linalg.norm(delta_Z[:, i] - np.matmul(A_mle, Z[:, i]))**2
    LL_rotational += -0.5 * np.linalg.norm(delta_Z[:, i] - np.matmul(A_rotational,

print('LL for MLE', LL_mle)
print('LL for Rotational', LL_rotational)
```

Imshow of A Matrix



Imshow of A Matrix (MLE)





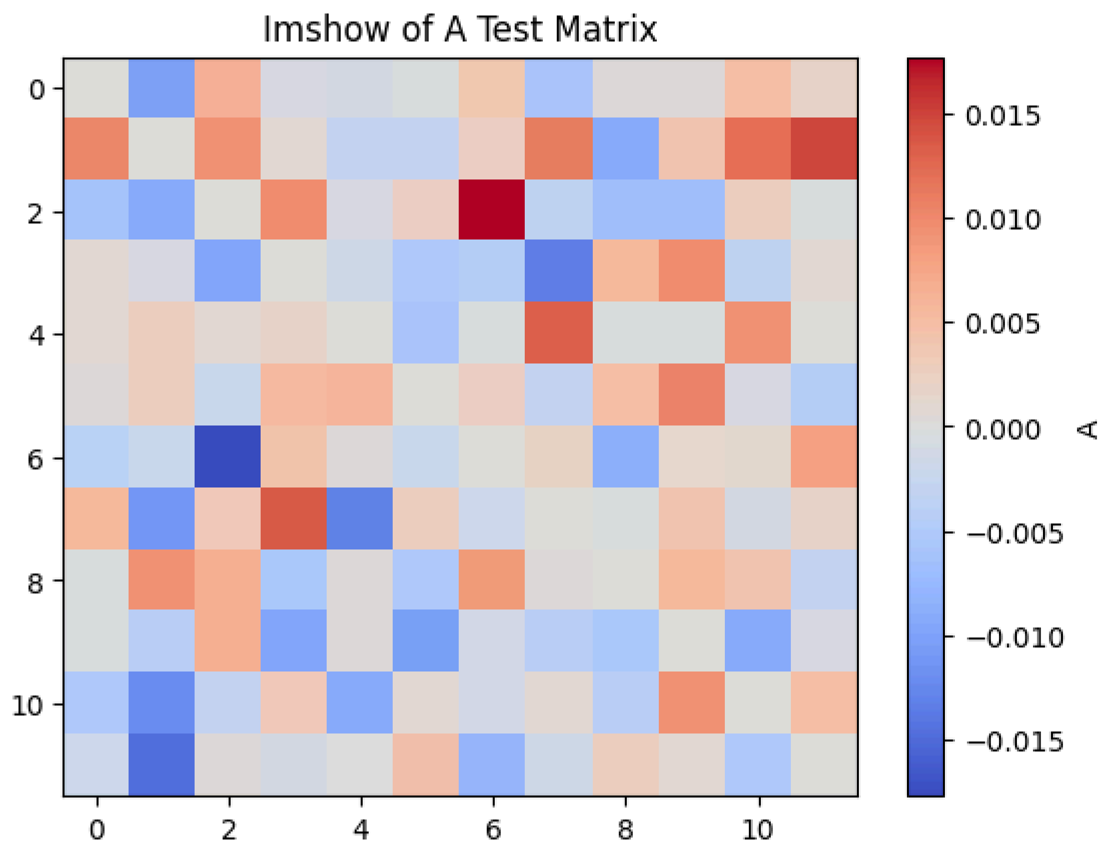
```
Processing for  0
Processing for  500
Processing for  1000
Processing for  1500
Processing for  2000
Processing for  2500
Processing for  3000
Processing for  3500
Processing for  4000
Processing for  4500
LL for MLE -6.484043435044501
LL for Rotational -7.327843832956184
```

In [596…

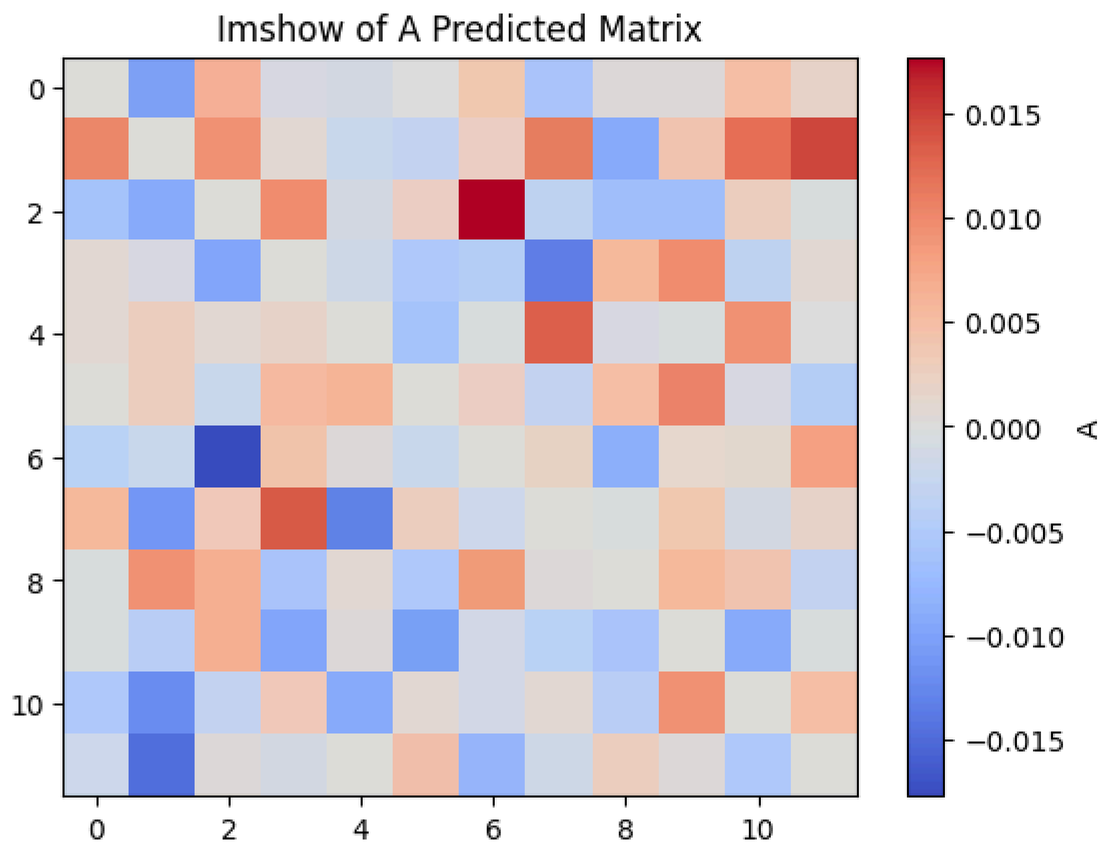
```
test_data = np.load('data/test.npz')
Z_test, A_test = test_data['Z_test'], test_data['A_test']

print('Shape of Z_test', np.shape(Z_test))
print('Shape of A_test', np.shape(A_test))


def solve(Z, pca_dim = 12, is_proj = True, needs_reshape = False, second_dim = X.sh
    """
    Function to solve for A matrix
    """

    print('-'*100)
    print('Shape of Z_pca', np.shape(Z))

    delta_Z_reshape = np.diff(Z, axis=2)
    delta_Z = delta_Z_reshape.reshape(pca_dim, -1)
```

```python
    Z = np.delete(Z, 0, axis=2)
    Z = Z.reshape(pca_dim, -1)
    print('Shape of Z', np.shape(Z))
    print('Shape of delta_Z', np.shape(delta_Z))

    print('-'*100)
    H = construct_H(pca_dim)
    print('Shape of H', np.shape(H))

    print('-'*100)
    W = gen_W(H, Z)
    print('Shape of W', np.shape(W))

    print('-'*100)
    b_vec, Q = gen_bQ(W, delta_Z)
    print('Shape of b_vec', np.shape(b_vec))
    print('Shape of Q', np.shape(Q))

    print('-'*100)
    beta_pred = np.linalg.inv(Q).dot(b_vec)
    print('Shape of beta_pred', np.shape(beta_pred))
    print('-'*100)

    pred = reconstruct_A(H, beta_pred)
    print('Shape of A_pred', np.shape(pred))
    print('-'*100)

    return pred



plt.imshow(A_test, aspect='auto', cmap='coolwarm', interpolation='nearest')
plt.title('Imshow of A Test Matrix')
plt.colorbar(label='A')

plt.show()

pred = solve(Z_test)
plt.imshow(pred, aspect='auto', cmap='coolwarm', interpolation='nearest')
plt.title('Imshow of A Predicted Matrix')
plt.colorbar(label='A')
plt.show()

# Calculating the mean squared error
mre = np.mean(np.abs(A_test - pred))
print('Mean Absolute Error for A matrix', mre)
```

```
Shape of Z_test (12, 108, 46)
Shape of A_test (12, 12)
```



```
--------------------------------------------------------------------------------
----------------
Shape of Z_pca (12, 108, 46)
Shape of Z (12, 4860)
Shape of delta_Z (12, 4860)
--------------------------------------------------------------------------------
----------------
Shape of H (66, 12, 12)
--------------------------------------------------------------------------------
----------------
Shape of W (66, 12, 4860)
--------------------------------------------------------------------------------
----------------
Shape of W (66, 12, 4860)
Shape of Z (12, 4860)
Shape of b_vec (66,)
Shape of Q (66, 66)
Shape of b_vec (66,)
Shape of Q (66, 66)
--------------------------------------------------------------------------------
----------------
Shape of beta_pred (66,)
--------------------------------------------------------------------------------
----------------
Shape of A_pred (12, 12)
--------------------------------------------------------------------------------
----------------
```

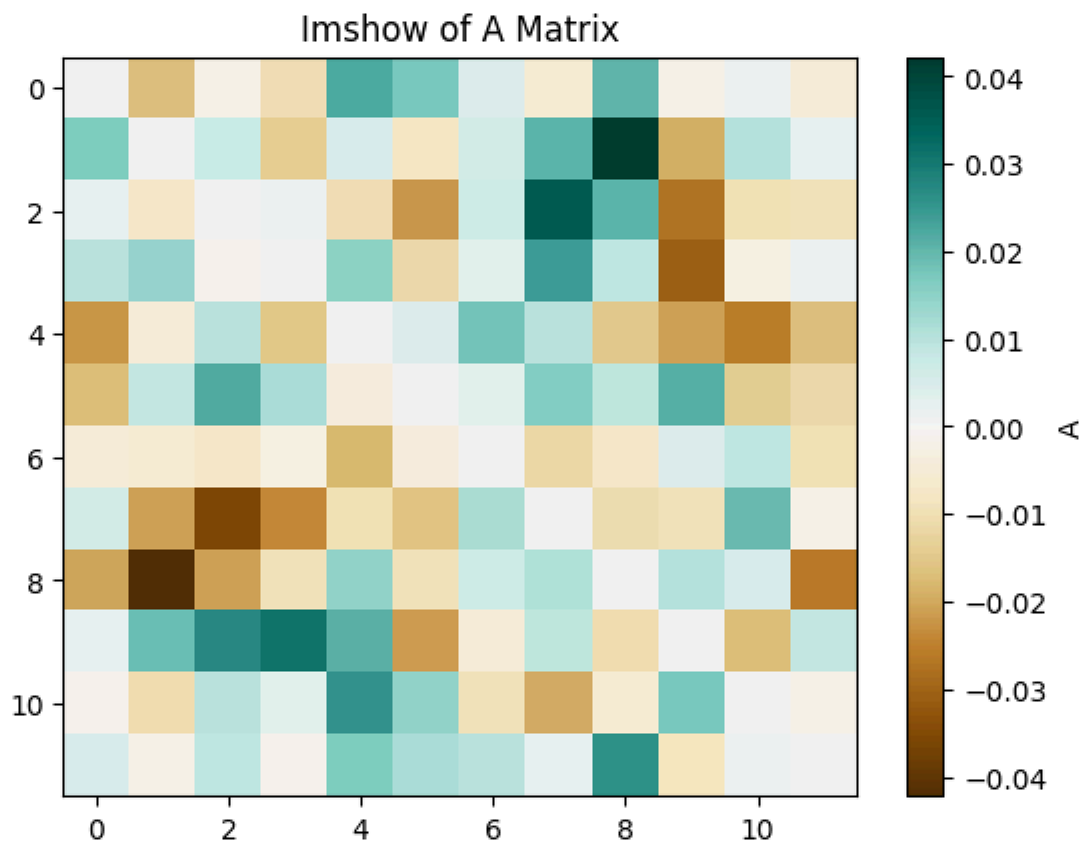


```
Mean Absolute Error for A matrix 2.029546462186608e-05
```

Part 5

In [620…

```
# Refresh the Data
plt.imshow(A_pred, aspect='auto', cmap='BrBG', interpolation='nearest')
plt.title('Imshow of A Matrix')
plt.colorbar(label='A')
plt.show()


eigvals, eigvecs = np.linalg.eig(A_pred)
print('Shape of eig', np.shape(eigvecs))

print('All eigenvalues for A_test')
print(eigvals)
```



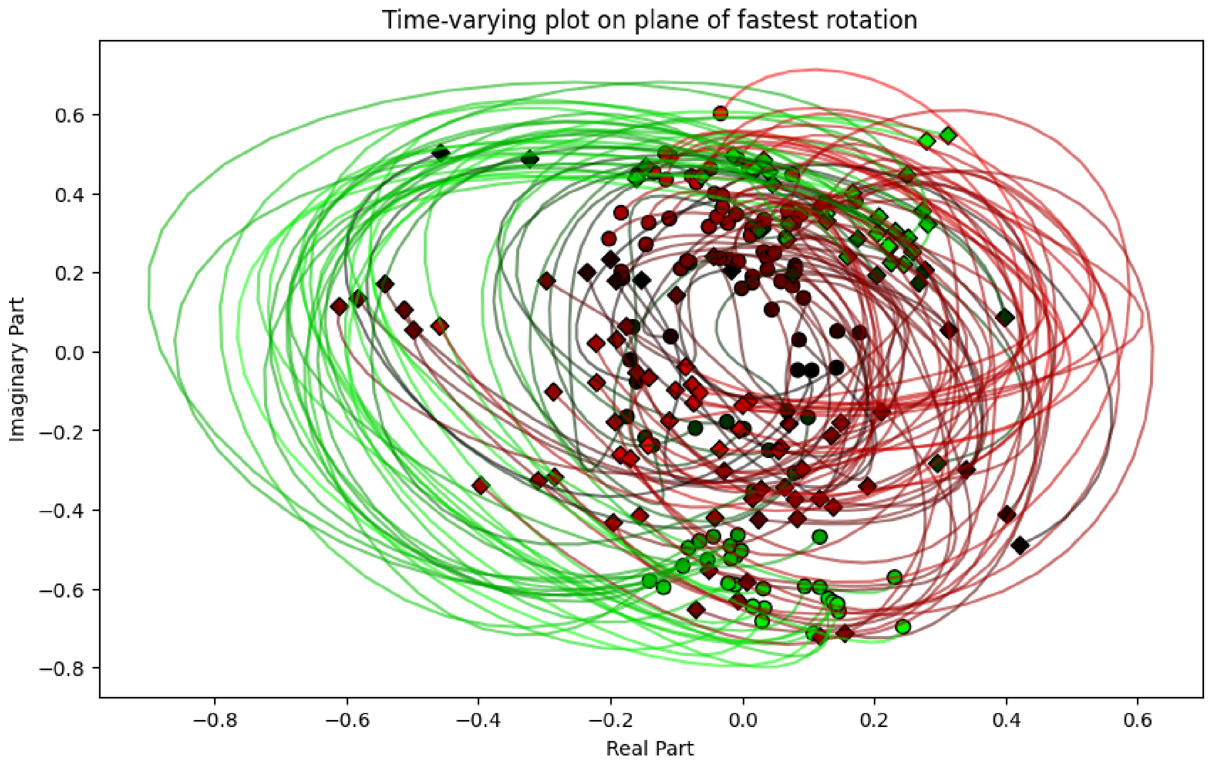
```
Shape of eig (12, 12)
All eigenvalues for A_test
[ 1.05709712e-18+0.09214251j  1.05709712e-18-0.09214251j
 -6.93889390e-18+0.06661613j -6.93889390e-18-0.06661613j
 -2.16840434e-18+0.04475907j -2.16840434e-18-0.04475907j
  1.73472348e-18+0.01605887j  1.73472348e-18-0.01605887j
  5.96311195e-19+0.00284482j  5.96311195e-19-0.00284482j
 -1.73472348e-18+0.01092628j -1.73472348e-18-0.01092628j]
```

In [574…

```
largest_eigvec = eigvecs[:,0]

le_real = np.real(largest_eigvec)
le_imag = np.imag(largest_eigvec)

le_real = le_real / np.linalg.norm(le_real)
le_imag = le_imag / np.linalg.norm(le_imag)

P = []
P.append(le_real)
P.append(le_imag)

max_bins = 35

Z_trunc = np.reshape(Z, (12, X.shape[1], -1))[:, :, :max_bins]
Z_trunc = np.reshape(Z_trunc, (12, -1))

print('Shape of P', np.shape(P))
print('Shape of Z_trunc', np.shape(Z_trunc))

trajectory = np.matmul(P, Z_trunc)
```

```
pc1 = trajectory[0].reshape(-1, max_bins)
pc2 = trajectory[1].reshape(-1, max_bins)

fig, ax = plt.subplots(figsize=(10, 6))

c_colors = cond_color.get_colors(pc1[:, 0], pc2[:, 0])
cond_color.plot_start(pc1[:, 0], pc2[:, 0], c_colors, 200, ax)
cond_color.plot_end(pc1[:, -1], pc2[:, -1], c_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, c_colors):
    ax.plot(xs, ys, color=c, alpha=0.55)

ax.set_xlabel('Real Part')
ax.set_ylabel('Imaginary Part')
ax.set_title('Time-varying plot on plane of fastest rotation')
plt.show()
```

```
Shape of P (2, 12)
Shape of Z_trunc (12, 3780)
```



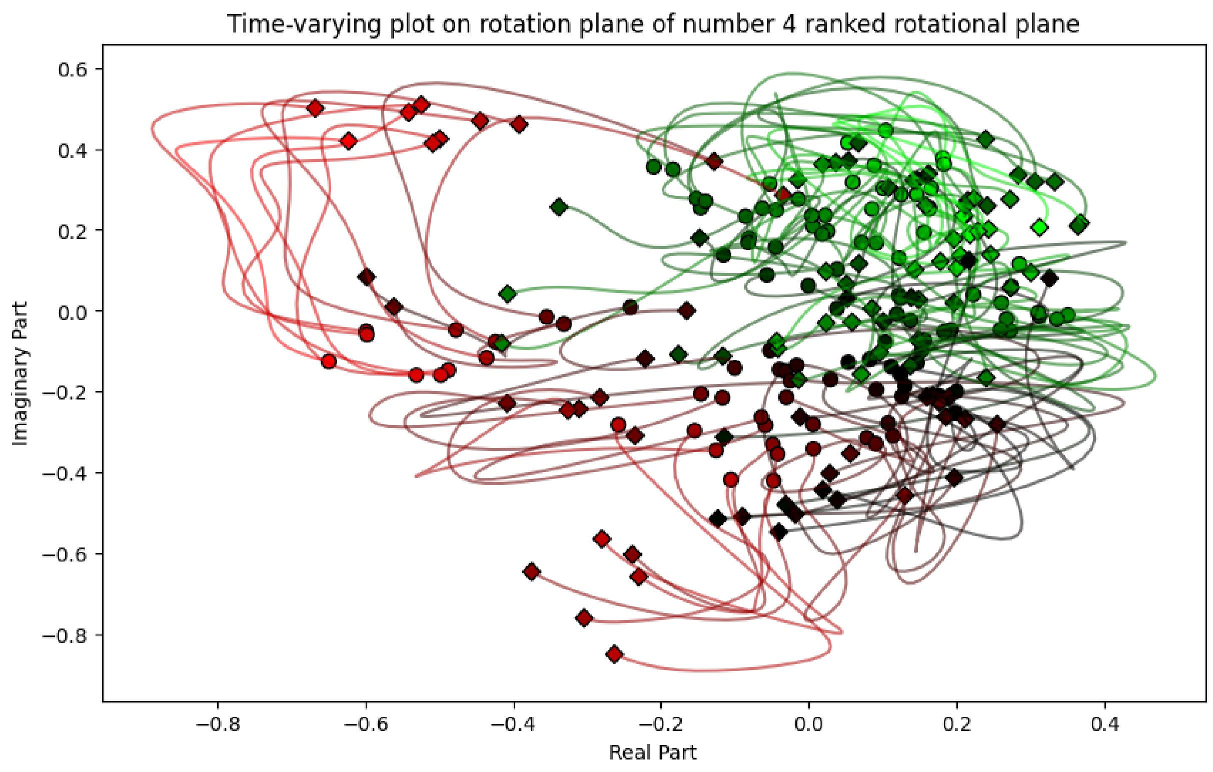
> Q: How do these trajectories differ (qualitatively) from those in exercise 3?

A: We see better path separation, and more 'extreme' trajectories. This makes sense given the projection onto the high-angular speed plane, the trajectories are more skewed and longer lasting (even as the time interval has been cropped to [-150, 200]). There is more 'rotational movement' in this plane - which is expected as its the plane of fastest angular rotation. Naturally, the biggest difference is the overarching curved shape of all the trajectories - this homegeneity in task paths is not so apparent in the last trajectory plot. Green paths for example, seem to follow a segment of an ellipse and can quite clearly be discerned from the red and black paths. In one way - one can expect this to be a good representation of the monkey's motor dynamics, this is how the hand is actually moving. In a semantic sense, the trajectories may well indeed line up to the curvature of the hand path as the monkey responds to different 'go' cues.

In [579…

```
# Doing the same with the 2nd and 3rd largest eigenvalues

print(np.sort(eigvals))
n = 4

index = (2*n - 2)
second_largest_eigvec = eigvecs[:, index]

sl_real = np.real(second_largest_eigvec)
sl_imag = np.imag(second_largest_eigvec)

sl_real = sl_real / np.linalg.norm(sl_real)
sl_imag = sl_imag / np.linalg.norm(sl_imag)

P = []
P.append(sl_real)
P.append(sl_imag)

Z_trunc = np.reshape(Z, (12, X.shape[1], -1))[:, :, :max_bins]
Z_trunc = np.reshape(Z_trunc, (12, -1))

trajectory = np.matmul(P, Z_trunc)

pc1 = trajectory[0].reshape(-1, max_bins)
pc2 = trajectory[1].reshape(-1, max_bins)

fig, ax = plt.subplots(figsize=(10, 6))

c_colors = cond_color.get_colors(pc1[:, 0], pc2[:, 0])
cond_color.plot_start(pc1[:, 0], pc2[:, 0], c_colors, 200, ax)
cond_color.plot_end(pc1[:, -1], pc2[:, -1], c_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, c_colors):
    ax.plot(xs, ys, color=c, alpha=0.55)

ax.set_xlabel('Real Part')
ax.set_ylabel('Imaginary Part')
ax.set_title('Time-varying plot on rotation plane of number {} ranked rotational pl
plt.show()
```

```
[-6.93889390e-18-0.06661613j -6.93889390e-18+0.06661613j
 -2.16840434e-18-0.04475907j -2.16840434e-18+0.04475907j
 -1.73472348e-18-0.01092628j -1.73472348e-18+0.01092628j
  5.96311195e-19-0.00284482j  5.96311195e-19+0.00284482j
  1.05709712e-18-0.09214251j  1.05709712e-18+0.09214251j
  1.73472348e-18-0.01605887j  1.73472348e-18+0.01605887j]
```



> Q: How do these trajectories differ from those projected in the plane corresponding to the fastest 2D rotation? Speculate why.

A: As we filter down in the eigenvalues - note that we have to skip to index (2n - 1) because of the conjugate symmetry in the eigenvalues - the structure gradually deteriorates, becoming less ellipsoidal and more stochastic. Task paths are not as discernable and do not follow a smooth rotational trajectory, and there are changes in the heading with respect to time - this is as we explore the planes for slower rotations. There could be a few explanations for the same, the faster eigen-values dominate the matrix, the linearity of the operation means that on the faster plane they are smooth and ellipsoidal. This holds for the second and third fastest eigen-values too. For slower eigen-values, the effects of the fast vectors compound and corrupt movement in the corresponding plane, possibly leading to the non-ellipsoidal and 'stochastic' nature of the trajectories in these slower spaces. Those hyper-planes are not the driving force behind the neural activity, and so, in a state-space sense, they are less accountable for the final movement.

What is also key to note here is that similar colors have similar trajectories and different colors have seperable trajectories - this is good evidence that this projection has captured some underlying neural data that corresponds to the monkeys motor movements. The trajectories are colored by the distribution of start points - and therefore - they roughly correspond to different tasks. So seeing the separation in this plane - taskwise - is a good indicator that the rotation dynamics have modelled the neural data well.

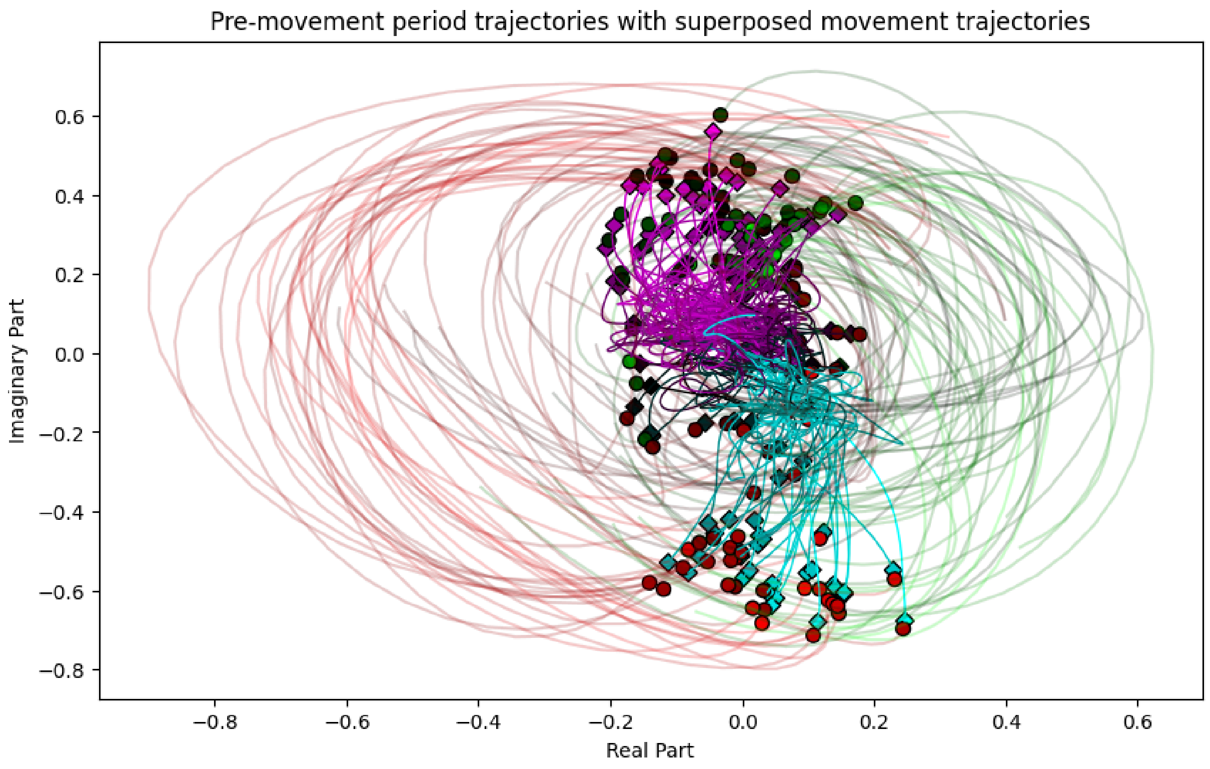
Part 6

In [514…
```
print('Shape of V_m', np.shape(V_m))
print('Shape of P_fr', np.shape(P))

pfrvmt = np.matmul(P, V_m.T)

print('Shape of pfrvmt', np.shape(pfrvmt))
print('Shape of X', np.shape(X))

X_6 = X_mean_centered

s_6_ind = times.tolist().index(-800)
e_6_ind = times.tolist().index(-150)

print('Start index', s_6_ind)
print('End index', e_6_ind)

X_6 = X_6[:, :, s_6_ind:e_6_ind+1]
print('Shape of X_6', np.shape(X_6))
```

```
Shape of V_m (182, 12)
Shape of P_fr (2, 12)
Shape of pfrvmt (2, 182)
Shape of X (182, 108, 130)
Start index 0
End index 65
Shape of X_6 (182, 108, 66)
```

In [515…
```
X_6 = X_6.reshape(X_6.shape[0], -1)
trajectory_6 = np.matmul(pfrvmt, X_6)

print('Shape of trajectory_6', np.shape(trajectory_6))

pc1 = trajectory_6[0].reshape(-1, e_6_ind - s_6_ind + 1)
pc2 = trajectory_6[1].reshape(-1, e_6_ind - s_6_ind + 1)

fig, ax = plt.subplots(figsize=(10, 6))

alt_colors = cond_color.get_colors(pc1[:, -1], pc2[:, -1], True)
cond_color.plot_end(pc1[:, -1], pc2[:, -1], alt_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, alt_colors):
    ax.plot(xs, ys, color=c, alpha=1, linewidth=1)

pc1 = trajectory[0].reshape(-1, max_bins)
pc2 = trajectory[1].reshape(-1, max_bins)

c_colors = cond_color.get_colors(pc1[:, -1], pc2[:, -1], False)
cond_color.plot_start(pc1[:, 0], pc2[:, 0], c_colors, 200, ax)
#cond_color.plot_end(pc1[:, -1], pc2[:, -1], c_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, c_colors):
    ax.plot(xs, ys, color=c, alpha=0.2)

ax.set_xlabel('Real Part')
ax.set_ylabel('Imaginary Part')
ax.set_title('Pre-movement period trajectories with superposed movement trajectorie
plt.show()
```

```
Shape of trajectory_6 (2, 7128)
```



> Q: How do these trajectories differ from the ones after movement-related activity has begun?

> Q: Assuming the rotational dynamics observed during the period of movement-related activity is indeed autonomous, how can you interpret the projected trajectories during the preceding (pre-movement) interval? Think about the epoch in the task: what does the monkey know in this epoch (specifically after the target onset, but before the go-cue or movement onset)?

A: We see very little actual coherence in the movement - the magnitude of movement is a lot smaller too. Quite Clearly the start points of the movement period and the end points the pre-movement period align very well. This is expected as the trajectories are expected to be smooth for the entire duration, but what is to note is how the neurons somehow anticipate a start position in this hyper-plane and gradually the activity moves towards it.

If we assume autonomous rotational dynamics, for which this plot shows good evidence (given the faster planes of rotation) - one can infer the pre-movement interval as the anticipation interval. The monkey will be trained to recognise certain facts that precede the go 'cue'. Whilst it may not known exactly when the cue is going to be given, it knows the rough interval (one can infer that the monkey has built up an internal model of roughly how long it must wait before the go cue). The anticipation of acting could be presented as the neural activity projected on this plane gravitating towards the required task-specific framework to then execute the action. In humans, and from empirical evidence, the anticipation of executing an action or internalising it can make one better trained to execute it better.

An interesting line of study would be to test the same monkey but periodically (or with some random probability) never give it the go cue - and record the neural activity past this pre-movement period. Whether the projected readings would remain in the same space (or osciallate about the same region) or gradually return to the source region (roughly around 0,0 in the plane of fastest rotation).

> Reward prediction error (RPE) signals are crucial for reinforcement learning and decision-making as they quantify the mis match between predicted and obtained rewards. RPE signals are encoded in the neural activity of multiple brain areas, such as midbrain dopaminergic neurons, prefrontal cortex, and striatum. However, it remains unclear how these signals are expressed through anatomically and functionally distinct subregions of the striatum. In the current study, we examined to which extent RPE signals are represented across different striatal regions. To do so, we recorded local field potentials (LFPs) in sensorimotor, associative, and limbic striatal territories of two male rhesus monkeys performing a free-choice probabilistic learning task. The trial-by-trial evolution of RPE during task performance was estimated using a reinforcement learning model fitted on monkeys' choice behavior. Overall, we found that changes in beta band oscillations (15–35Hz), after the out come of the animal's choice, are consistent with RPE encoding. Moreover, we provide evidence that the signals related to RPE are more strongly represented in the ventral (limbic) than dorsal (sensorimotor and associative) part of the striatum. To conclude, our results suggest a relationship between striatal beta oscillations and the evaluation of outcomes based on RPE signals and highlight a major contribution of the ventral striatum to the updating of learning processes.

> https://www.jneurosci.org/content/jneuro/43/18/3339.full.pdf

In essence, the monkey has realised that on target-onset it is required to execute a specific action. This is not yet a probabilsitic task - so its neural data attempts to anticipate activating the necessary commands to guide the hand. While the direction may be ambiguous, the general action stays the same.

> https://www.jneurosci.org/content/jneuro/27/16/4334.full.pdf

> Researchers found that monkeys generally relied on a sense of elapsed time and prior probabilities on deciding when to initiate an anticipatory [neural] response. When the timing between on-set and the go-cue is increased, as

> predicted, there is higher uncertainty and variability in the monkey's response

Part 7

In [637…

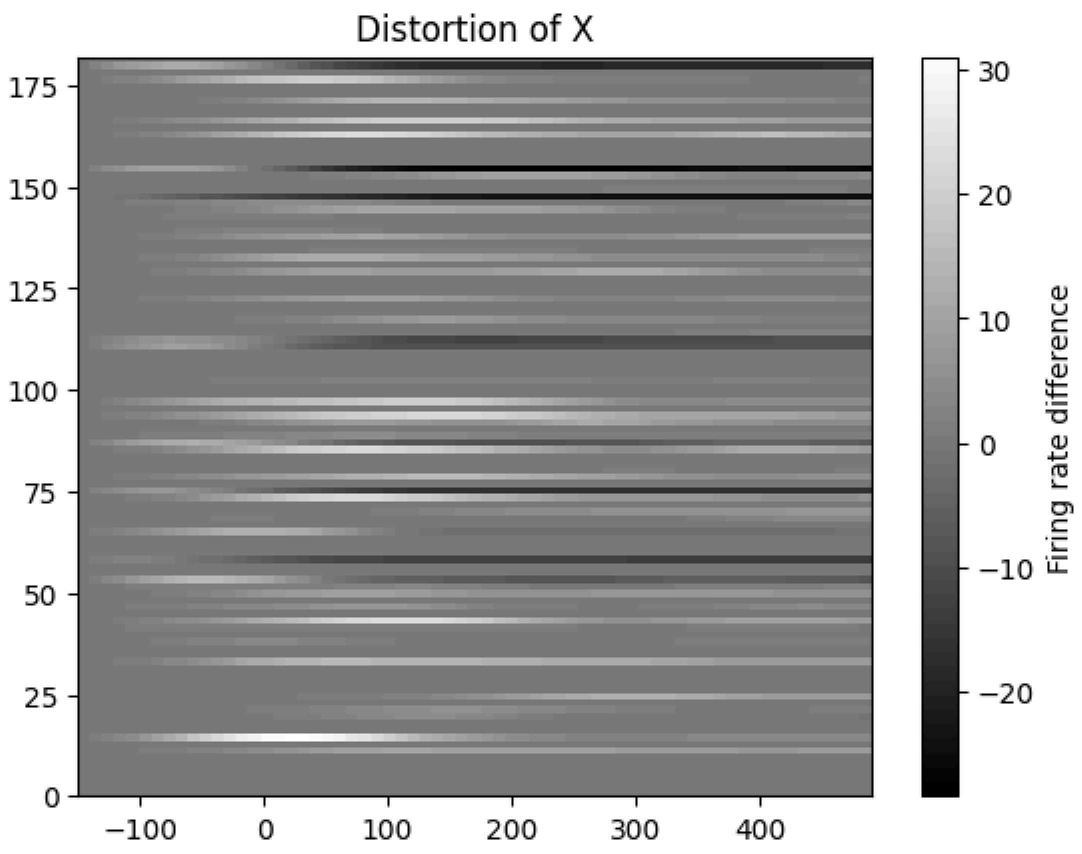
```
# Rerunning with distortion

psths = np.load('data/psths.npz')
X, times = psths['X'], psths['times']   # X is a 3D array of shape (n_neurons, n_co

print('Shape for X', np.shape(X))
C = np.shape(X)[1]

start_time = np.where(times == -150)[0][0]
end_time = np.where(times == 490)[0][0]

X_mp = X[:, :, start_time:end_time+1]
X_mp = np.array(X_mp)
times_mp = times[start_time:end_time+1]

print('Shape of X_mp', np.shape(X_mp))
```

```
Shape for X (182, 108, 130)
Shape of X_mp (182, 108, 65)
```

In [638…

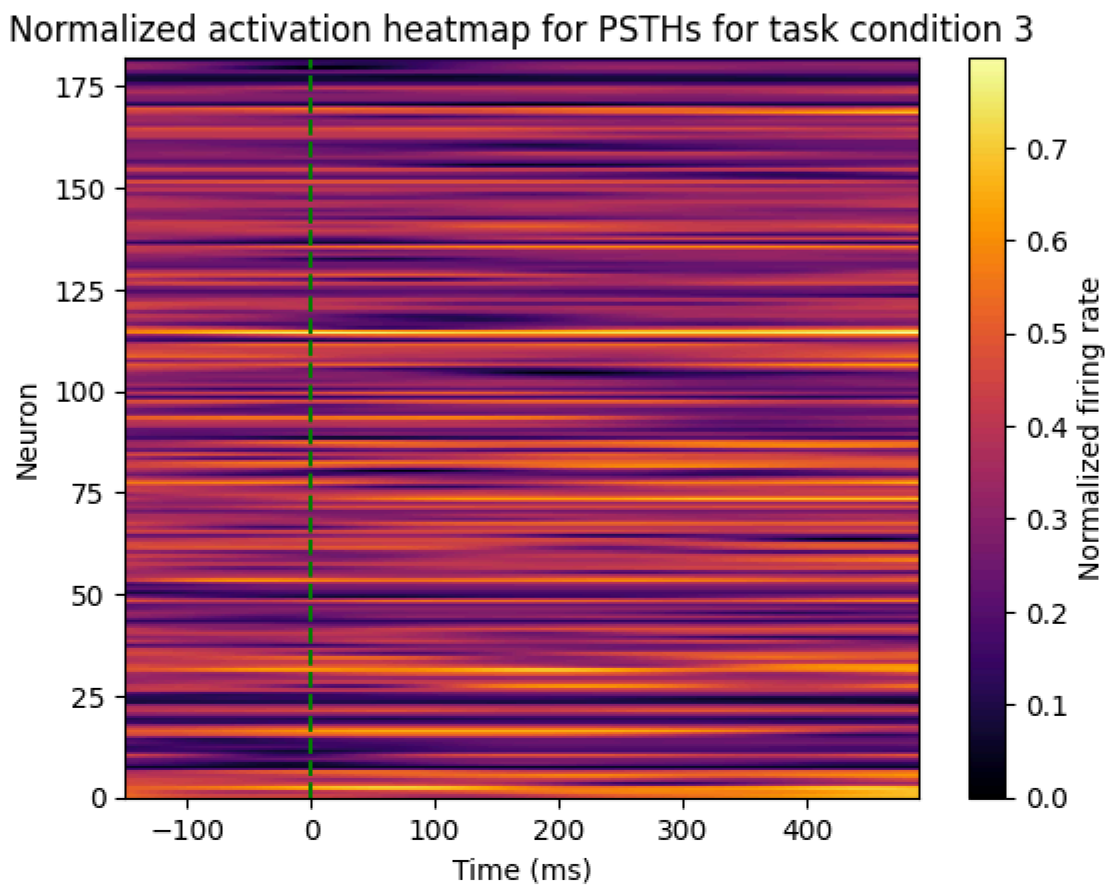
```
# Distortion for X_mp
C_rands = np.random.choice(C, C//2, replace=False)
X_mp_distort = np.zeros_like(X_mp)

print('Shape of C_rands', np.shape(C_rands))

for i, n_slice in enumerate(X_mp):
    distort_slice = n_slice.copy()
    distort_slice[C_rands] = 2 * np.vstack([n_slice[C_rands, 0]]*np.shape(X_mp)[2])
    X_mp_distort[i] = distort_slice


sample_index = 1
plt.imshow(X_mp[sample_index, :, :] - X_mp_distort[sample_index, :, :], aspect='aut
plt.colorbar(label='Firing rate difference')
plt.title('Distortion of X')
plt.show()
```

```
Shape of C_rands (54,)
```

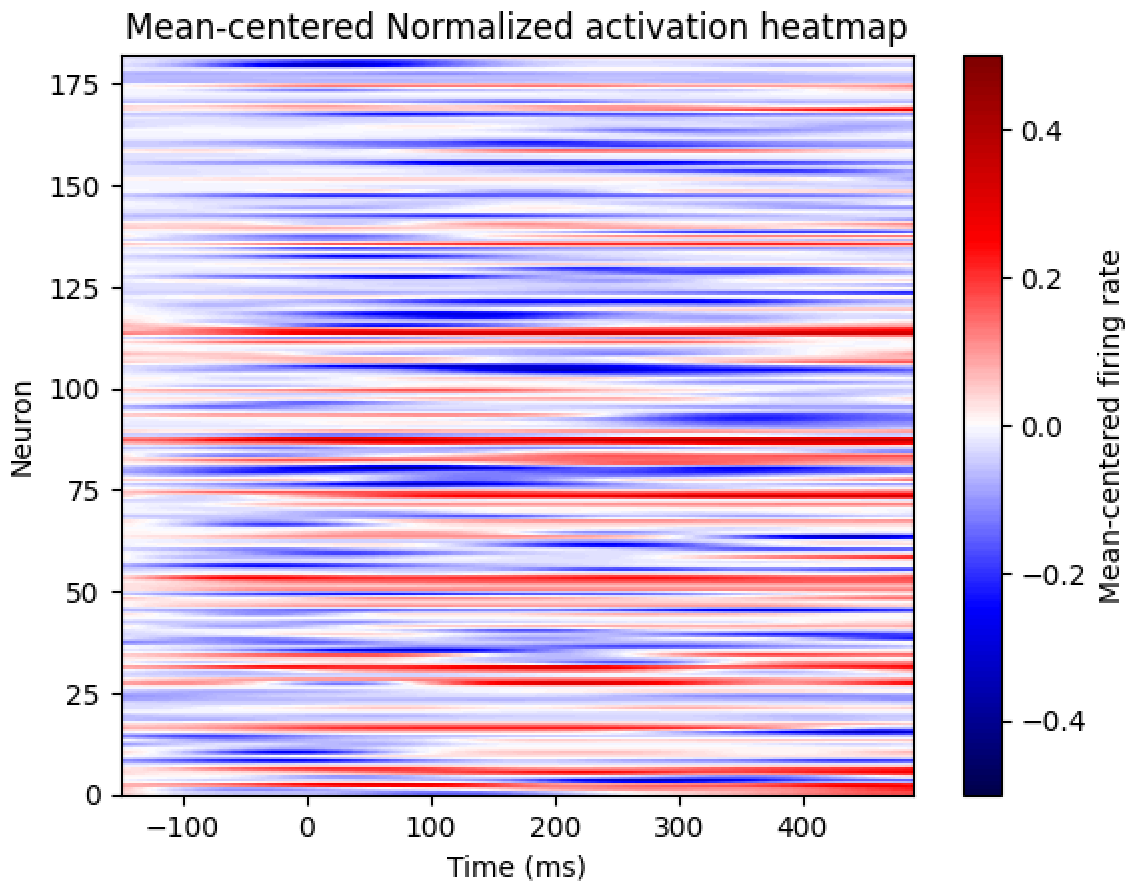


In [639…

```
# Now we repeat everything on X_mp_distort

max_firing_rates = np.max(X_mp_distort, axis=(1, 2))
min_firing_rates = np.min(X_mp_distort, axis=(1, 2))

X_normalized = (X_mp_distort - min_firing_rates[:, None, None]) / (max_firing_rates
slice_normalized = X_normalized[:, task_condition, :]
plt.imshow(slice_normalized, aspect='auto', cmap='inferno', extent=[times_mp[0], ti
plt.title('Normalized activation heatmap for PSTHs for task condition {}'.format(ta
plt.vlines(0, 0, slice_normalized.shape[0], color='green', linestyle='--')
plt.xlabel('Time (ms)')
plt.ylabel('Neuron')
plt.colorbar(label='Normalized firing rate')
plt.show()

X_mean_centered = X_normalized - np.mean(X_normalized, axis=1)[:, None, :]
slice_mean_centered = X_mean_centered[:, task_condition, :]
plt.imshow(slice_mean_centered, aspect='auto', cmap='seismic', extent=[times_mp[0],
plt.colorbar(label='Mean-centered firing rate')
plt.title('Mean-centered Normalized activation heatmap')
plt.xlabel('Time (ms)')
plt.ylabel('Neuron')
plt.show()
```



Normalized activation heatmap for PSTHs for task condition 3
Neuron
Time (ms)
Normalized firing rate



Mean-centered Normalized activation heatmap
Neuron
Time (ms)
Mean-centered firing rate

In [640…

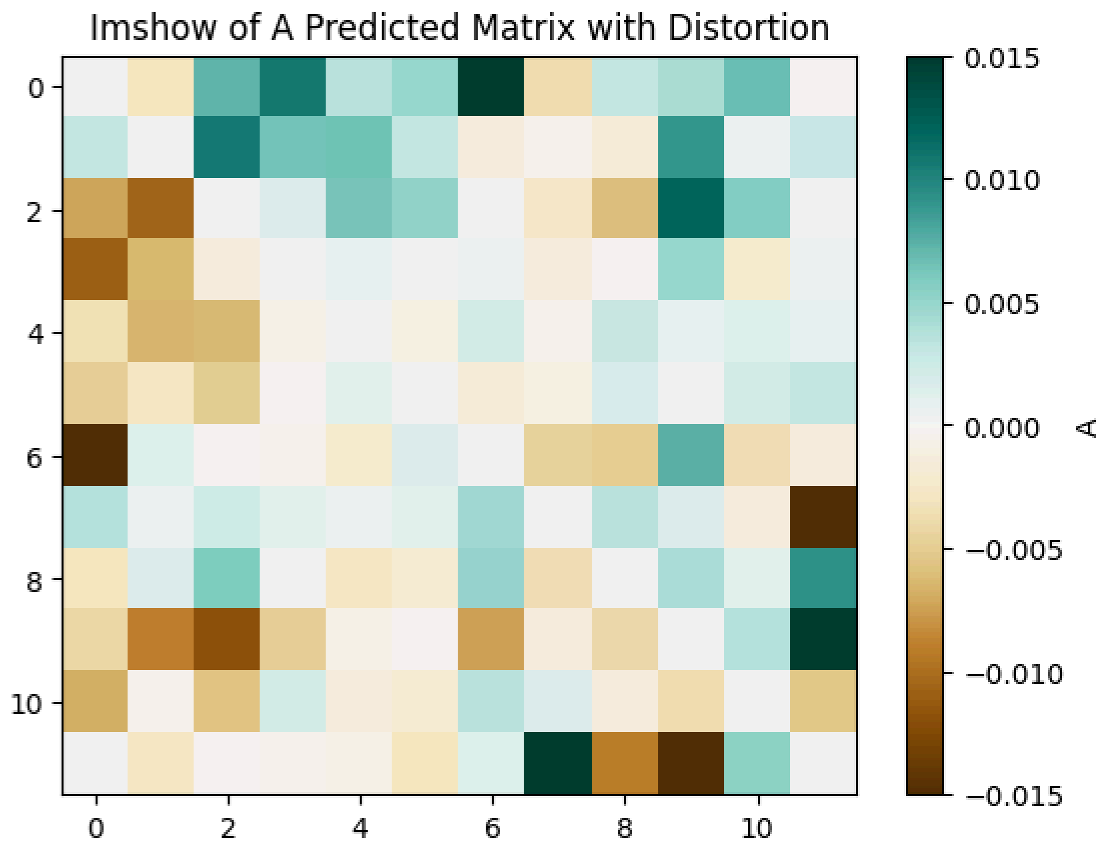
```
X_new = X_mean_centered.reshape(X_mean_centered.shape[0], -1)
print('Shape of X_new', np.shape(X_new))

S = np.matmul(X_new, X_new.T)
S = S / X_new.shape[0]

print('Shape of S', np.shape(S))

eigvals, eigvecs = np.linalg.eigh(S)
print('Shape of eigvals', np.shape(eigvals))

M = 12

V_m = eigvecs[:, -M:]
print('Shape of V_m', np.shape(V_m))
```

```
Shape of X_new (182, 7020)
Shape of S (182, 182)
Shape of eigvals (182,)
Shape of V_m (182, 12)
```

In [641…

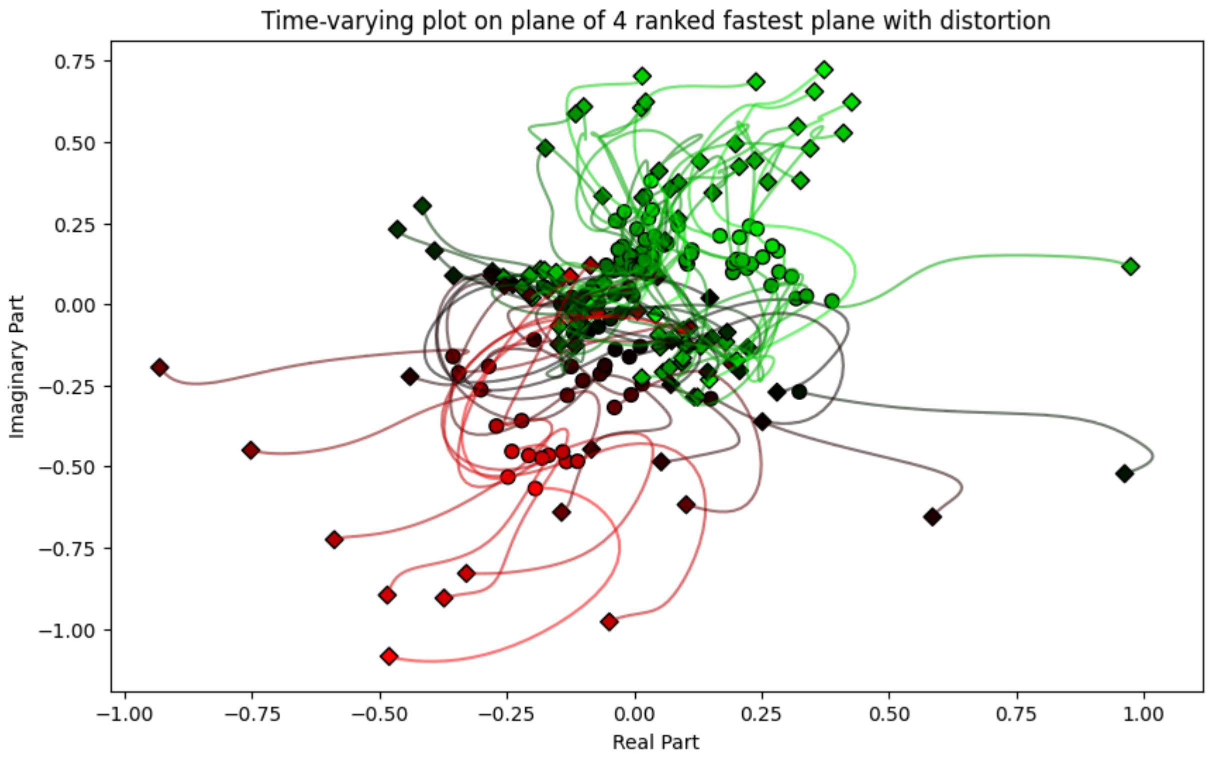
```
Z_7 = np.matmul(V_m.T, X_new)
Z_7 = Z_7.reshape(12, C, -1)

print('Shape of Z_7', np.shape(Z_7))

pred_7 = solve(Z_7, pca_dim=12, second_dim=X_new.shape[1],)

plt.imshow(pred_7, aspect='auto', cmap='BrBG', interpolation='nearest', vmin=-0.015
plt.colorbar(label='A')
plt.title('Imshow of A Predicted Matrix with Distortion')
plt.show()
```

```
Shape of Z_7 (12, 108, 65)
----------------------------------------------------------------------------------
----------------
Shape of Z_pca (12, 108, 65)
Shape of Z (12, 6912)
Shape of delta_Z (12, 6912)
----------------------------------------------------------------------------------
----------------
Shape of H (66, 12, 12)
----------------------------------------------------------------------------------
----------------
Shape of W (66, 12, 6912)
----------------------------------------------------------------------------------
----------------
Shape of W (66, 12, 6912)
Shape of Z (12, 6912)
Shape of b_vec (66,)
Shape of Q (66, 66)
Shape of b_vec (66,)
Shape of Q (66, 66)
----------------------------------------------------------------------------------
----------------
Shape of beta_pred (66,)
----------------------------------------------------------------------------------
----------------
Shape of A_pred (12, 12)
----------------------------------------------------------------------------------
----------------
```



```
In [647…
eigvals, eigvecs = np.linalg.eig(pred_7)
print('Shape of eig', np.shape(eigvecs))
print(eigvals)

index = 6
this_eigvec = eigvecs[:,index]

le_real = np.real(this_eigvec)
le_imag = np.imag(this_eigvec)

le_real = le_real / np.linalg.norm(le_real)
le_imag = le_imag / np.linalg.norm(le_imag)

P = []
P.append(le_real)
P.append(le_imag)

max_bins = 35

Z_trunc = np.reshape(Z_7, (12, C, -1))[:, :, :max_bins]
Z_trunc = np.reshape(Z_trunc, (12, -1))

print('Shape of P', np.shape(P))
print('Shape of Z_trunc', np.shape(Z_trunc))

trajectory_7 = np.matmul(P, Z_trunc)

pc1 = trajectory_7[0].reshape(-1, max_bins)
pc2 = trajectory_7[1].reshape(-1, max_bins)

fig, ax = plt.subplots(figsize=(10, 6))

c_colors = cond_color.get_colors(pc1[:, 0], pc2[:, 0])
cond_color.plot_start(pc1[:, 0], pc2[:, 0], c_colors, 200, ax)
cond_color.plot_end(pc1[:, -1], pc2[:, -1], c_colors, 40, ax)

for xs, ys, c in zip(pc1, pc2, c_colors):
    ax.plot(xs, ys, color=c, alpha=0.55)

ax.set_xlabel('Real Part')
ax.set_ylabel('Imaginary Part')
ax.set_title('Time-varying plot on plane of {} ranked fastest plane with distortion

plt.show()
```

```
Shape of eig (12, 12)
[-1.24683250e-18+0.03544485j -1.24683250e-18-0.03544485j
  0.00000000e+00+0.02356709j  0.00000000e+00-0.02356709j
  1.40946282e-18+0.01621037j  1.40946282e-18-0.01621037j
  0.00000000e+00+0.00624469j  0.00000000e+00-0.00624469j
 -2.71050543e-19+0.00277963j -2.71050543e-19-0.00277963j
 -1.74488787e-19+0.00072833j -1.74488787e-19-0.00072833j]
Shape of P (2, 12)
Shape of Z_trunc (12, 3780)
```


Time-varying plot on plane of 4 ranked fastest plane with distortion
Imaginary Part
Real Part
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

```
In [ ]:
```