**CUED - Engineering Tripos Part IIB 2024-2025** **Module Coursework**

| Module | 4G10 | Title of report | Brain Machine Interfaces: Assignment 1 |
|---|---|---|---|
| Date submitted: 21/11/24 | | Assessment for this module is ☒ 100% / ☐ 25% coursework of which this assignment forms 50 % | |

| UNDERGRADUATE and POST GRADUATE STUDENTS | | |
|---|---|---|
| Candidate number: | 5587B | ☒ Undergraduate<br>☐ Post graduate |

| Feedback to the student<br>☐ **See also comments in the text** | | Very good | **Good** | Needs improvmt |
|---|---|---|---|---|
| CONTENT | **Completeness, quantity of content:**<br>Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly? | | | |
| | **Correctness, quality of content**<br>Is the data correct? Is the analysis of the data correct? Are the conclusions correct? | | | |
| | **Depth of understanding, quality of discussion**<br>Does the report show a good technical understanding? Have all the relevant conclusions been drawn? | | | |
| | Comments: | | | |
| PRESENTATION | **Attention to detail, typesetting and typographical errors**<br>Is the report free of typographical errors? Are the figures/tables/references presented professionally? | | | |
| | Comments: | | | |

Marker: Date:

# 1 Part 1

The following series of exercises concern the application of LDS models to monkey neural data recorded while it was set to perform a task requiring motor control. More specifically, we constrain the model to be one that is rotational in state-space. The motivation for this is the periodicity that is often observed in real life activities for animals. With the given task for the monkey also exhibiting the repetitiveness that may have influenced evolved behaviors - an interesting hypothesis is to explore the validity of this rotational model in being the underlying driving force for the neural data.

To begin with, some basic plots of the neural data can be made and some qualitative assessments of the nature of the data can be made. The Peristimulus Time Histograms (PSTH) data is decomposed into a 3D array (containing data of all neural data across time and different task conditions) and a linear time array.

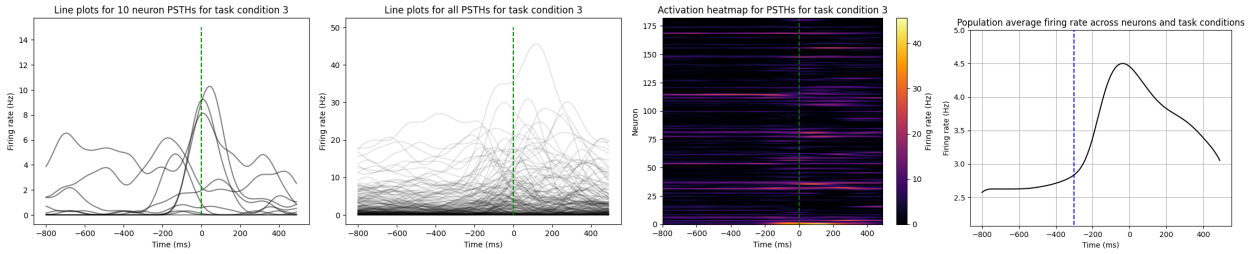



Figure 1: Neural data plots (from left to right: line plots of few neurons, all neurons for specified task, heat-map activation plot for specified task, population average across neurons and tasks

Referring to figure 1, it can be observed that 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 300 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 stabilizing (presumably as the activity being performed is constant in that time frame - e.g. moving in a straight line to target).

The population average plot shows that on average, there is increased neural activity as we approach the 'go' cue. What is interesting is that from qualitative analysis - though this can be asserted by using something like an edge-identifying filter like a Laplacian of Gaussian - around -300 ms is the time when neuron activity changes significantly. This is interesting as it suggests a sort of non-causality between the timing of the go cue and the neuron activity. If one is to use a linear causal 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 clues and this could result in some sort of 'training-induced' anticipation. From the problem description - while the go cue is given at variable times, this is constrained to the interval (0.5, 1) seconds.

An interesting field of the literature to explore here is the effect of Reward Prediction Error signals (RPE) in dictating the anticipation. Given the training set-up (the task itself is non-probabilistic) this wouldn't really be a key factor to explore here. Though what could be happening is that the monkey is envisioning performing the action it needs to perform. Literature regarding RPE signals indicate that repetitive training induce beta oscillations in the monkey stratium which encode RPE signals - such that after target on-set but before the go cue, the neural activity may already transition to the initial task-execution state [1]. This is expanded on more in part 6.

## 2 Part 2

One can now work on reducing the dimensionality of the data, the goal is to go from 182 neurons across conditions and time bins to a lower dimension (for this report we set M = 12) 2D representation with the conditions and time bins axes concatenated. Before the steps for PCA are followed, some data pre-processing is performed. The data is normalized with respect to the maximum and minimum observed firing rates. This then constrains the firing rates to be between 0 and 1. The data is then mean centered (across neurons and time bins). Normalization ensures that each neuron is represented equally in the dataset and that the neural activity decomposition focuses on the *change* in neural activity rather than its magnitude (some neurons may be predisposed to be frequent firing). Mean centering then equally weights the 'on' and 'off' state of the neuron.

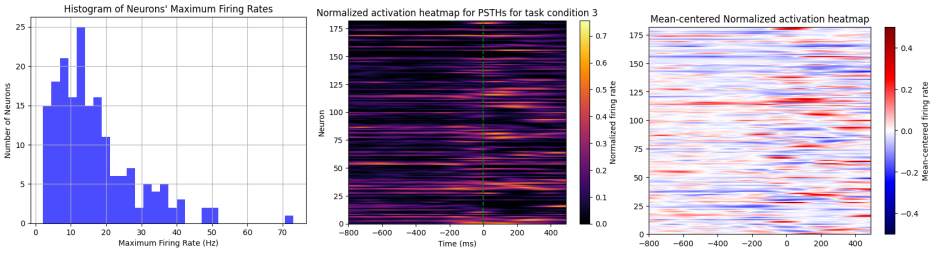



Figure 2: From left to right: PSTH neuron firing rate histogram, task-specific normalization, task-specific mean centering

PCA decomposition is performed then by performing a covariance operation over the data. The data is truncated to time bins corresponding to the [-150 ms, 300 ms] interval, then reshaped across the conditions and time bins axes. It is then a 2D array of shape 182x4968. The 182 x 182 covariance matrix is calculated by taking the matrix product of the 2D array with its transpose. Eigenvalue decomposition is performed on this matrix, then the last 12 eigenvectors (corresponding to the largest 12 eigenvalues) form the 12 x 182 matrix that is $V_m$ this then multiplies with the reshaped truncated PSTH array to yield a 12 x 4928 array.

## 3 Part 3

The M-dimensional trajectories can further be distilled into the first two PCA components. The trajectories of these components can be plotted - per task - on a 2D plane. A color-mapper will map colors according to the maximum spread direction of the initial points. When plotted with markers - figure 3 is produced. One can see that per-task trajectories are somewhat separable, though they are often quite stochastic in their heading with no cohesive structure. A key observation is that the start and end points of these trajectories are somewhat distinctly clustered (and are close to each other).

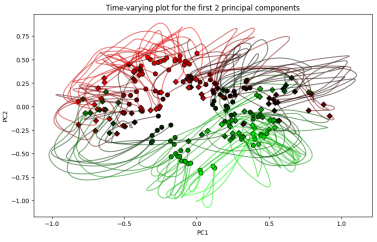



Figure 3: Trajectories of first 2 PCA components

## 4 Part 4

One can now begin with the required derivations to generate the maximum likelihood estimate for A. We start with the model definition. An attempt to fit an autonomous rotatory dynamical system to the neural data in projection space implies $z_{t+1} \approx R(\theta) z_t$ which - for small $\theta$ - can be simplified to $\Delta z_{t+1} \approx A z_t$ for an anti-symmetric A. This can be converted into an LDS equation: $\Delta z_{t+1} = A z_t + \sigma \epsilon_t$. So the maximum log-likelihood estimate for general A can be derived to be:

$$\mathcal{L} = \frac{-1}{2\sigma^2} \sum_t \left( (\Delta z_{t+1} - A z_t)^T (\Delta z_{t+1} - A z_t) \right) + D_{const} = \frac{-1}{2\sigma^2} \sum_t \left( \Delta z_{t+1}^T \Delta z_{t+1} + z_t^T A^T A z_t - 2 \Delta z_{t+1}^T A z_t \right)$$

The log property converts the product of probabilities into sums of log-probabilities, for brevity the constant part of the log-likelihood is ignored in later parts (since we will be differentiating these anyway). Here $\Delta z_{t+1}$ and $z_t$ correspond to the the $t^{th}$ time bin PCA projection. Taking the derivative of this with respect to the matrix A:

$$\frac{\partial \mathcal{L}}{\partial A} = \frac{\partial}{\partial A}\frac{-1}{2\sigma^2}\sum_t \left(z_t^T A^T A z_t - 2\Delta z_{t+1}^T A z_t\right) = \frac{-1}{\sigma^2}\sum_t \left(A z_t z_t^T - \Delta z_{t+1}^T z_t^T\right) = \frac{1}{\sigma^2}\left(\Delta Z - AZ\right)Z^T$$

The second step of this calculation can indeed be verified if we break down the matrix derivative into the derivative of the scalar with respect to the $i^{th}$ and $j^{th}$ components:

$$\frac{\partial \mathcal{L}}{\partial A_{ij}} = \frac{\partial}{\partial A_{ij}}\left(\frac{1}{\sigma_2}\sum_j\sum_i z_{t,j}A_{ij}\Delta z_{t+1,i} - \frac{1}{2\sigma^2}\sum_i\left(\sum_j A_{ij}z_{t,j}\right)^2\right)$$

Which yields:

$$\frac{\partial \mathcal{L}}{\partial A_{ij}} = \frac{1}{\sigma^2}\left(z_{t,j}\Delta z_{t+1,i} - z_{t,j}\sum_k A_{ik}z_{t,k}\right)$$

Which - in matrix form - yields the same result as before. However, this is for general A. We do implement a constraint on the model - which is that it is a rotational dynamics model. From this - there is an added constraint that $A_{ij} = -A_{ji}$. To avoid performing a constrained optimization with a Lagrangian (which would be tedious with a matrix), one can instead convert this into an unconstrained problem by re-parameterizing A. We know that with only the upper-triangle part of the matrix (excluding the diagonal) we can reconstruct the entire matrix. In this sense, one can create a vector $\beta$ which contains the data from the upper-triangle of the matrix- flattened out.

Therefore, the number of parameters for an $M \times M$ matrix reduces from $K = M^2$ to $K = \frac{M(M-1)}{2}$. Figure 4 demonstrates this process. More concisely, the reconstruction process can be performed by doing a tensor product between $\beta$ and a 3D tensor $H$ such that:

$$A_{ij} = \sum_{a=1}^{K}\beta_a H_{aij}$$

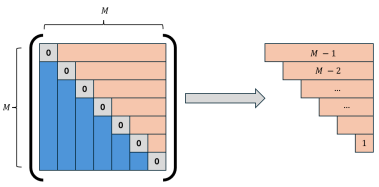



Figure 4: Diagram of constructing $\beta$ vector

Here $H$ is independent of $A$ and only depends on the manner in which one flattens the upper triangle of an $M \times M$ matrix. We can now take the derivative of the log-likelihood with respect to $\beta$ and perform unconstrained optimization - retrieve the optimal $\beta$ and reconstruct $A$. We therefore need to represent the log-likelihood in terms of this $\beta$ vector - this is easier when considering log-likelihood in element-wise fashion:

$$\mathcal{L} = D_{const} + \sum_t\left(\sum_i\left(\sum_j\sum_a \beta_a H_{aij}z_{t,j}\right)^2 - 2\sum_i\sum_j\sum_a \beta_a H_{aij}\Delta z_{t,i}z_{t,j}\right)$$

This is a fairly tedious equation to work with, it is made easier with the introduction of an auxiliary tensor $W$ where the elements of $W$ are defined such that: $W_{ait} = \sum_j H_{aij}z_{t,j}$. This then simplifies the above expression into:

$$\mathcal{L} = D_{const} + \sum_t\left(\sum_i\left(\sum_a \beta_a W_{ait}\right)^2 - 2\sum_i\sum_a \beta_a W_{ait}\Delta z_{t,i}\right)$$

Then taking derivatives with respect to elements in $\beta \to \beta_k$:

$$\frac{\partial \mathcal{L}}{\partial \beta_k} = \frac{-1}{2\sigma^2} \sum_t \left( \sum_i 2 \left( \sum_a W_{ait} \beta_a \right) W_{kit} - 2 \sum_i W_{kit} u_{it} \right)$$

$$\frac{\partial \mathcal{L}}{\partial \beta_k} = \sum_t \sum_i u_{it} W_{kit} - \sum_t \sum_i \left( \sum_a W_{ait} \beta_a \right) W_{kit}$$

We see that this is an element wise representation of the form $b_i - (\beta Q)_i$, where the row-vector $b$ and matrix $Q$ are represented by tensor products such that:

$$b_k = \sum_i \sum_t W_{kit} u_{it} \quad ; \quad Q_{b,a} = \sum_i \sum_t W_{bit} W_{ait}$$

Then solving for the optimal (ML) $\beta$ is simply solving the linear equation: $\beta_{optimal} = Q^{-1} b$. The $Q$ matrix and $b$ vector are solved for - in code - by taking tensor products across the relevant axes. Careful attention had to be given to ensuring that the shape is not mismatched in either term - so employing a strategy of regularly printing the shape of tensors as calculations were carried out was an important step in ensuring coherence of code. Matching matrix shapes is fairly crucial while debugging code in Python as Python has the undesirable behavior of sometimes switching row-wise and column-wise representations. Single vectors are printed as row-wise vectors but when eigenvalue decomposition is performed - for example - the eigenvectors are the columns of the returned matrix. the NumPy library fortunately preserves standard notation and matrix multiplicative rules.

From the MLE of $\beta$, the A matrix can simply be recovered by the equation involving it and the $H$ tensor. Figure 5 gives the ML estimate of the rotational A matrix using the algorithm described thus far. From tests of known data an absolute mean difference of $2.02e-6$ is observed, which is sufficiently small to confirm the workings of this algorithm. Now, the A matrix can be used to distill the data into 'rotation' hyper-planes in M-dimensional space.

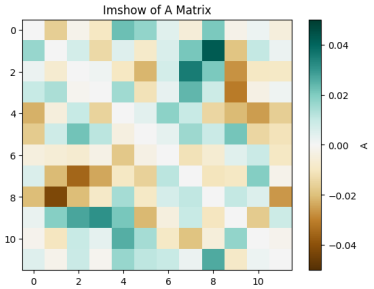



Figure 5: A matrix colormap

## 5 Part 5

From theory, one can describe rotation in M-dimensional space as rotations in M/2 (for even M) 2D planes that are spanned by the real and imaginary parts of the eigen-vectors of the MxM matrix A. The eigenvalues are purely imaginary (in executing code the real parts are in the order of magnitude of $e-18$, whereas the imaginary parts are $e-1$) and this is expected. They describe pure rotation in each of their planes.

Taking the eigenvectors corresponding to the eigenvalue of largest magnitude (in the complex plane) and decomposing it into a pair of normalized real and imaginary vectors - yields two orthogonal vectors that span a plane. One can think of this plane as projecting the neural data onto a isolated rotational plane. Using the largest imaginary eigenvalue's eigenvector would produce projections on the plane of fastest rotation (FR plane). Similarly, the plane of second, third fastest rotation can be found (see figure 6).

For the projection on the plane of fastest rotation (leftmost in figure 6) we see better path separation, and more 'extreme' trajectories. This makes sense given the projection onto the high-angular speed plane. There is more 'rotational movement' in this plane - which is expected. Naturally, the biggest difference is the overarching curved shape of all the trajectories - this homogeneity 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 tasks.

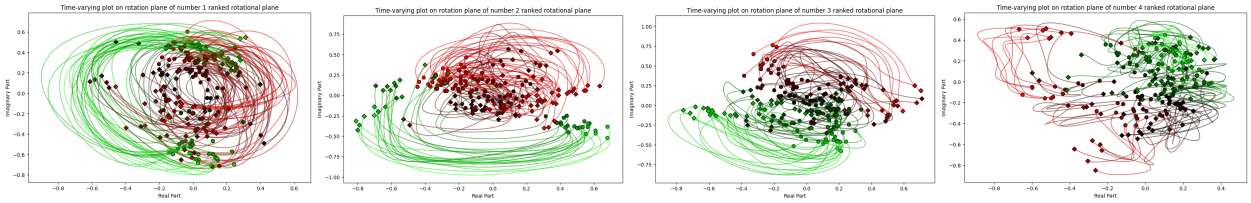



Figure 6: Projections onto the First, Second, Third, and Fourth Fastest Rotation Plane

As we filter down in the eigenvalues the structure gradually deteriorates, becoming less ellipsoidal and more stochastic. Task paths are not as discernible 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 eigenvalues dominate the matrix, the linearity of the operation means that on the faster plane they are smooth and ellipsoidal. For slower eigenvalues, these planes - while orthogonal to planes of faster rotation - are not driving the neural response, so are instead weakly rotational.

What is also key to note here is that similar colors have similar trajectories and different colors have separable 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 - task-wise - is a good indicator that the rotation dynamics have modeled the neural data well.

## 6 Part 6

What is also interesting to apply the PCA projection along with $F_{PR}$ projection in the pre-movement period. In doing so - one captures what the anticipatory neural activity is presenting in the time before the movement. Figure 7 demonstrates this. A similar color-coding has been performed to distinguish pre-movement activity. It is quite clear that the pre-movement trajectories are not rotational, which supports evidence that the rotational nature of trajectories is due to condition on-set and is not the natural state of movement.

If we assume autonomous rotational dynamics one can infer the pre-movement interval as the anticipation interval. The monkey will be trained to recognize 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 internalizing it can make one better trained to execute it better.

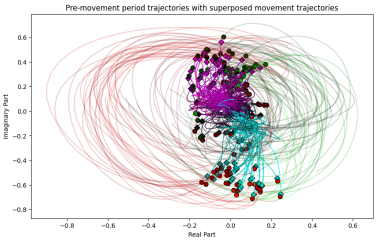



Figure 7: Pre-movement period trajectories

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 gradually return to the source region (roughly around 0,0 in the plane of fastest rotation) would indicate how anticipation plays into this rotational dynamics model. On this note, researchers [2] 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 expected, there is higher uncertainty and variability in the monkey's response.

# 7 Part 7

Naturally, one needs to question whether or not these dynamical models are simply over-fitted to the data (hallucinated) or are natural distillations of the latent space of the neural activity. To test the rigidity of the rotational dynamics model - the PSTH array is modified. For each neuron - at half of the task conditions (chosen at random) the firing rate is inverted along the time bin at -150 ms. This inversion, preserves continuity, but greatly increases the difference between the maximum and minimum values of the PSTH. Inverting movements for random tasks for each neuron essentially destroys the correlation between neurons (per task) and so this will corrupt latent features. In particular, the eigenvectors and values will change. This is sure to affect the rotational dynamic model if one assumes that this model underpinned neural activity. If the model was instead hallucinated and rotational trajectories were simply over-fitted, this would be invariant to the latent space correlations in the neural data.

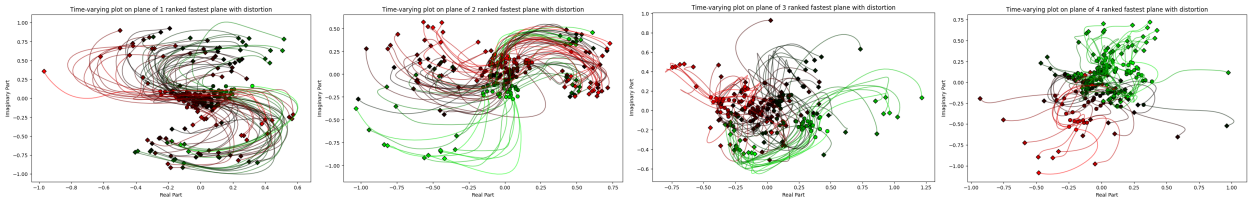



Figure 8: Trajectories on First, Second, Third, and Fourth Fastest Rotational Planes

What is interesting to observe is the fact that rotational trajectories are still present - though the trajectories themselves are far less separable. This means that even in the distorted data, the maximum likelihood estimate of the A matrix is able to discern some rotational underlying the data. And therefore the projection onto the plane of rotations still has some - albeit - weak rotational movements. The weak presence of this - is purely a qualitative statement, though it is apparent that trajectories are far less separable.

One can argue that this indicates a presence of some amount of hallucination and also some good representation of the latent model of the data. Again these can only be - so far - asserted qualitatively. Studies on rotational structures [3] have found quantitative measures for the 'strength' of presence of rotational dynamics and dubbed it a gyration number. Comprehensive studies of jPCA and other methods have concluded that these projective methods can often lead to "false positives". Another study [4] also stated: "we have concluded that developing a model of cortical dynamics based on such limited data may be an endless process, and debates about the nature of structural rotations may be unproductive". Attempts to use rotational dynamics in BCI and kinematic decoding systems have not been popularly adopted.

Yet, with all the results - one can make a good argument that rotational dynamical model is a good model for approximation. When compared to the pure MLE estimate of A, the rotational model is not too far. Computing the log-likelihood for the pure MLE estimate ($A_{MLE} = (\Delta ZZ^T)(ZZ^T)^{-1}$) yields $\mathbf{-6.484}$ while the $A_{rot}$ matrix yields: $\mathbf{-7.326}$. Naturally we expect it to be lower (due to the rotational a-priori assumption), but empirically the values are close, indicating strong evidence for the rotational model. Trajectory separability in the plane of fastest rotation also indicates goodness of the model. However, it is apparent that some amount of rotational trajectories are simply fitted onto data. The extent of this is not as strong as when there is no corruption, but it does indicate some amount of hallucination. The extent of this can be tested by applying this algorithm on synthetic datasets (with known underlying models) to generate a truth-table. Given the evidence with the data at hand, however, the rotational dynamics model can be said to be a strong descriptor for the underling neural activity.

# References


[1] Basanisi, Ruggero, et al. "Beta Oscillations in Monkey Striatum Encode Reward Prediction Error Signals." Journal of Neuroscience, Society for Neuroscience, 3 May 2023, doi.org/10.1523/JNEUROSCI.0952-22.2023. Accessed 20 Nov. 2024.

[2] De Hemptinne, Coralie, et al. "How Do Primates Anticipate Uncertain Future Events?" The Journal of Neuroscience, vol. 27, no. 16, 18 Apr. 2007, pp. 4334–4341, doi:10.1523/jneurosci.0388-07.2007.

[3] Kuzmina, Ekaterina, et al. "On the Rotational Structure in Neural Data." bioRxiv, Cold Spring Harbor Laboratory, 1 Jan. 2023, doi.org/10.1101/2023.09.11.557230. Accessed 20 Nov. 2024.

[4] Libby, Alexandra, and Timothy J. Buschman. "Rotational Dynamics Reduce Interference between Sensory and Memory Representations." Nature Neuroscience, vol. 24, no. 5, 5 Apr. 2021, pp. 715–726, doi:10.1038/s41593-021-00821-9.

# Part 1

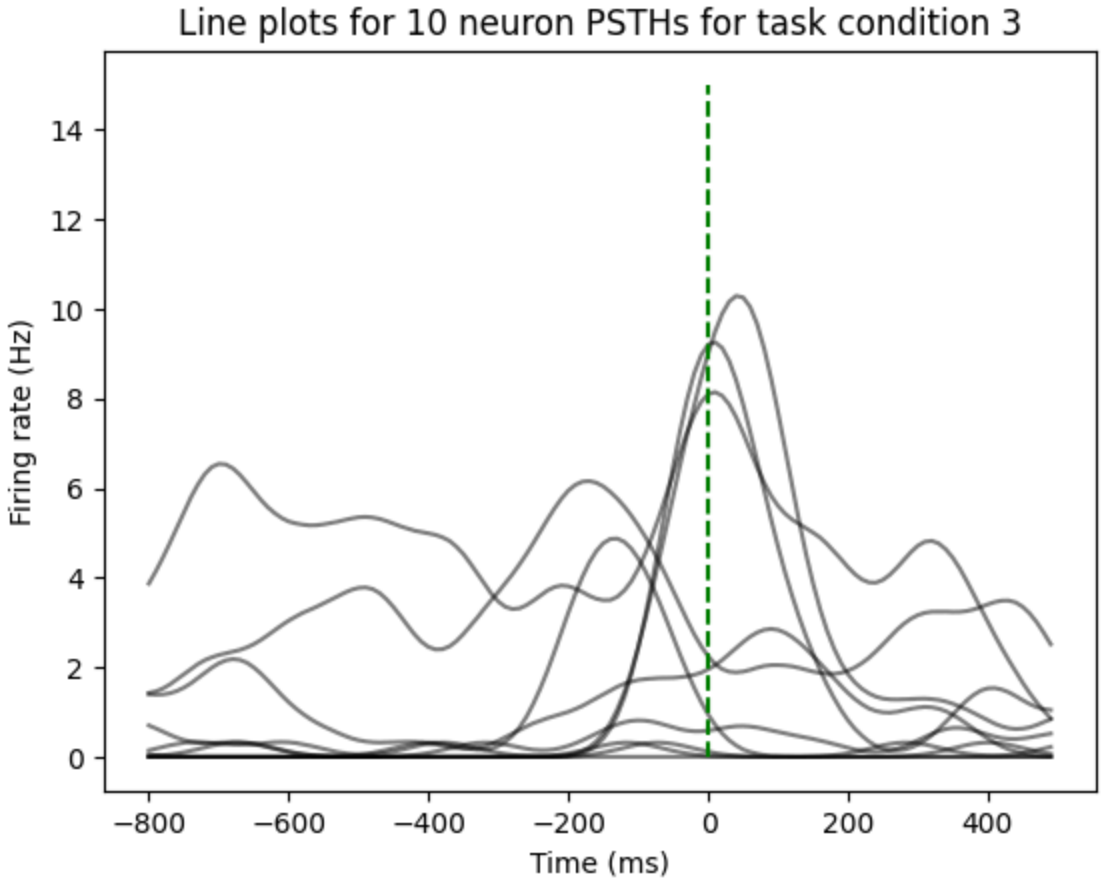
In [648…

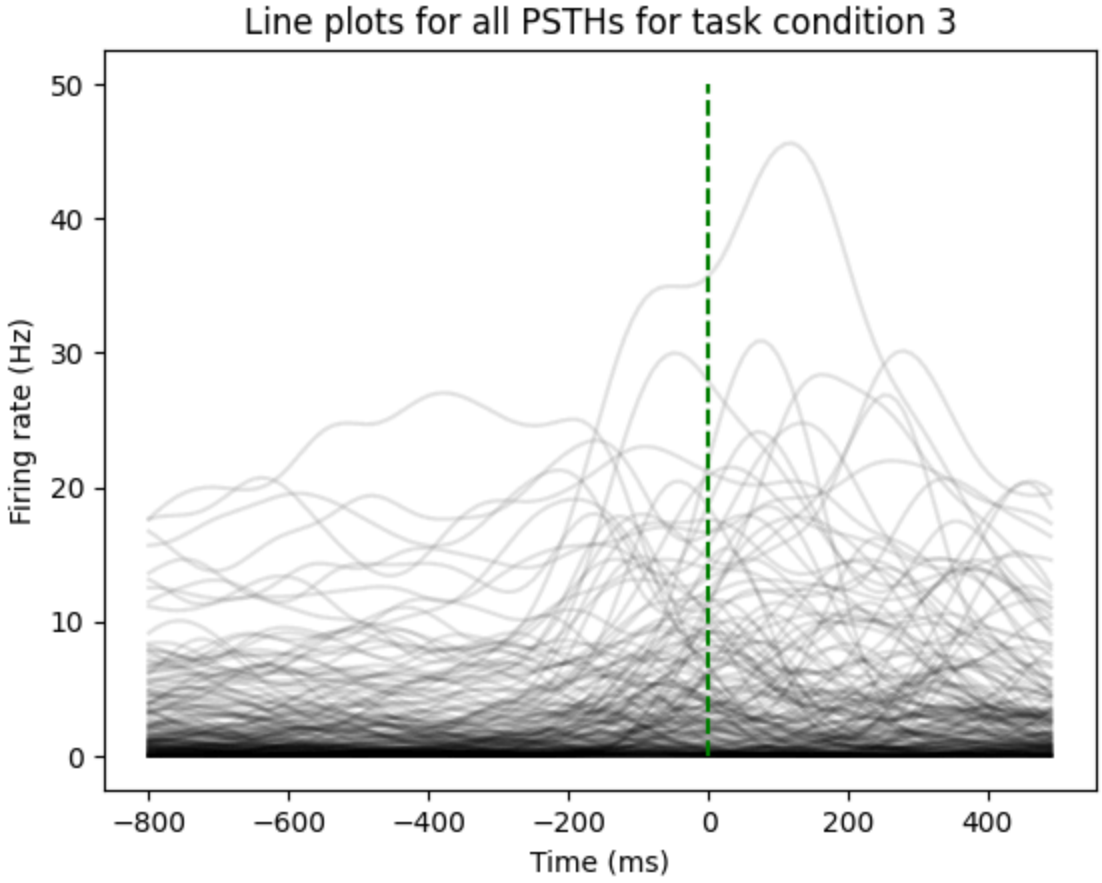
```
import numpy as np
from data import cond_color
from matplotlib import pyplot as plt


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']))
```

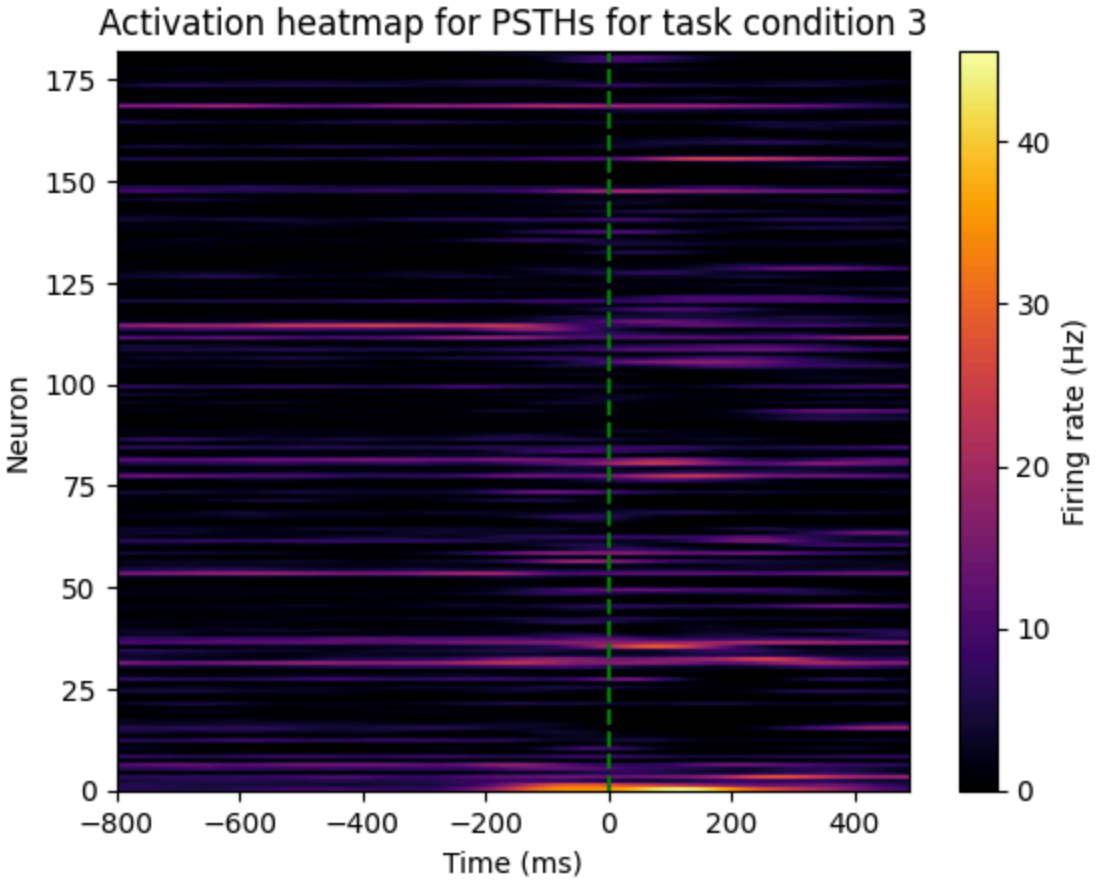
```
Shape for X (182, 108, 130)
Shape for times (130,)
```

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)
0
2
4
6
8
10
12
14
−800
−600
−400
−200
0
200
400
Time (ms)


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

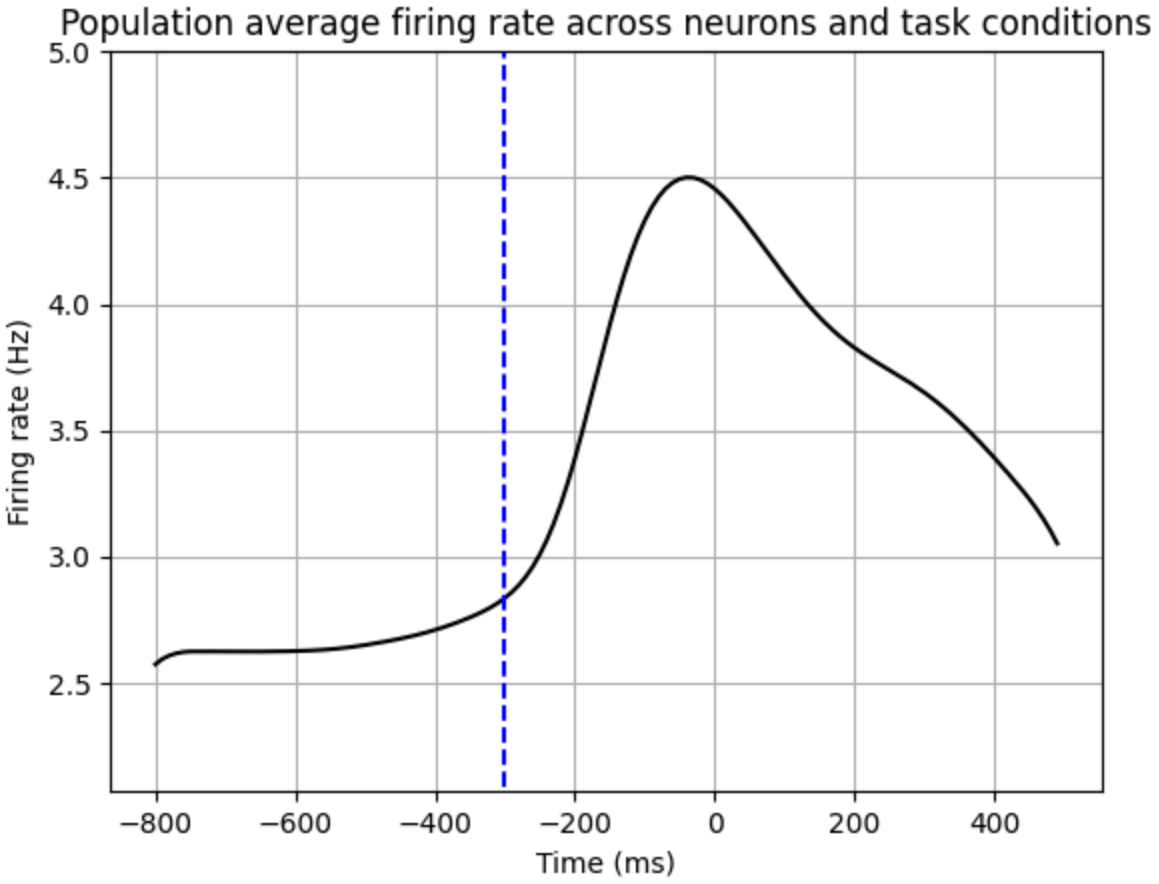



```
(182, 130)
```

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.

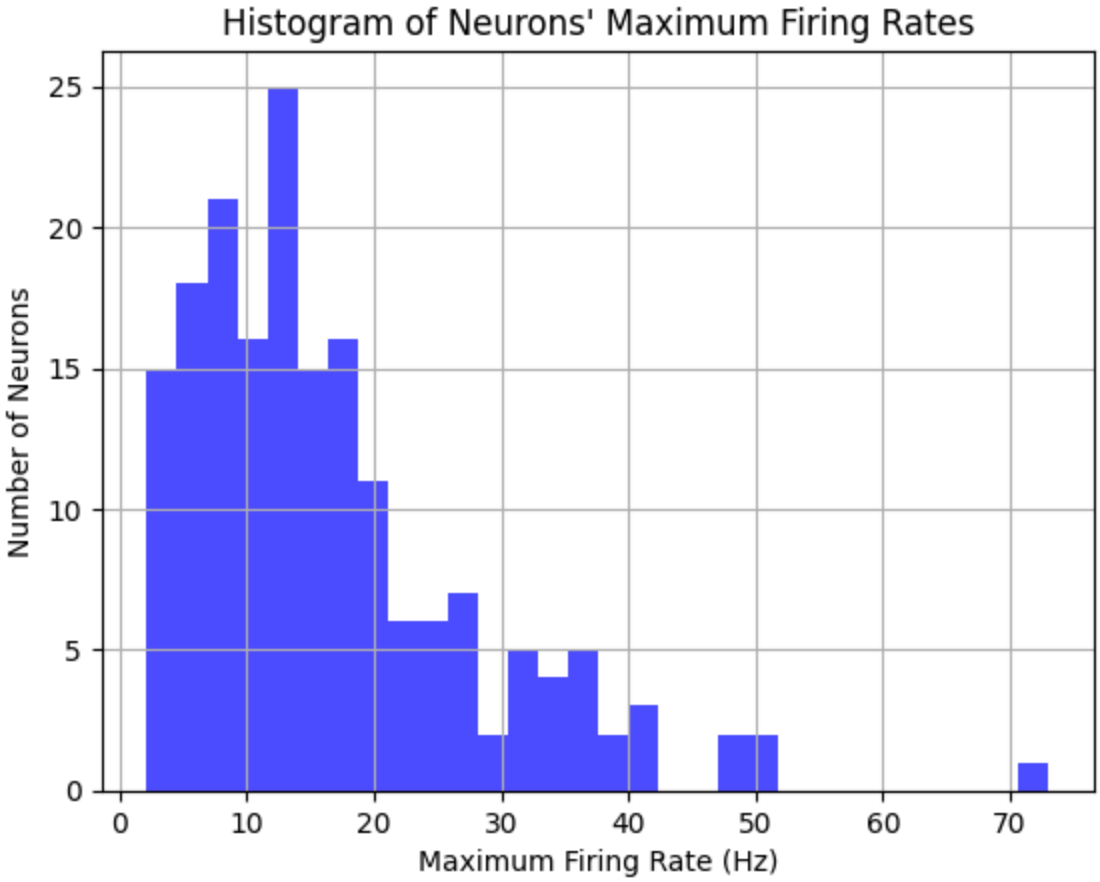
From qualitiative analysis - though this can be asserted by using something like an edge-identifying filter like a LoG - around -300 ms is the time when neuron activity changes significantly. This is interesting as it suggests non-causality between the timing of the go cue and the neuron activity. LDS 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…

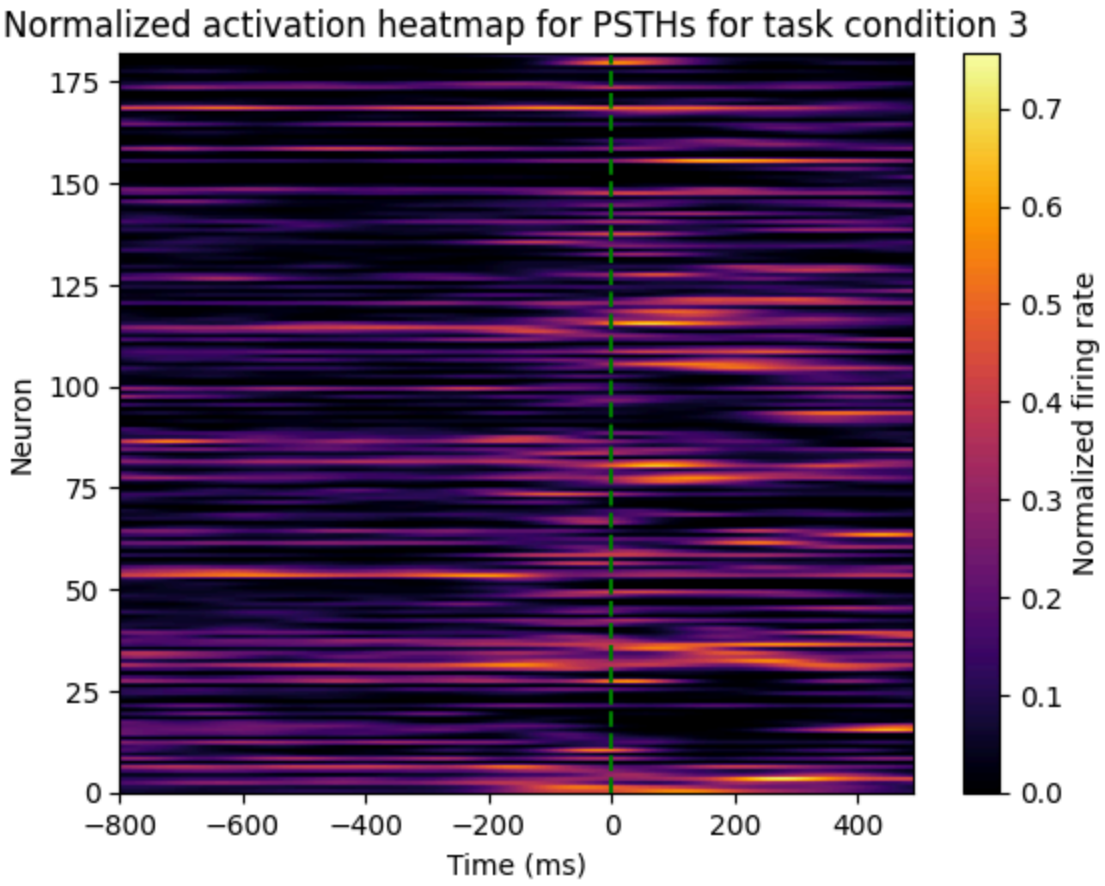
```
# Calculate the maximum firing rate for each neuron across conditions and time
max_firing_rates = np.max(X, axis=(1, 2))
min_firing_rates = np.min(X, axis=(1, 2))

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()
```

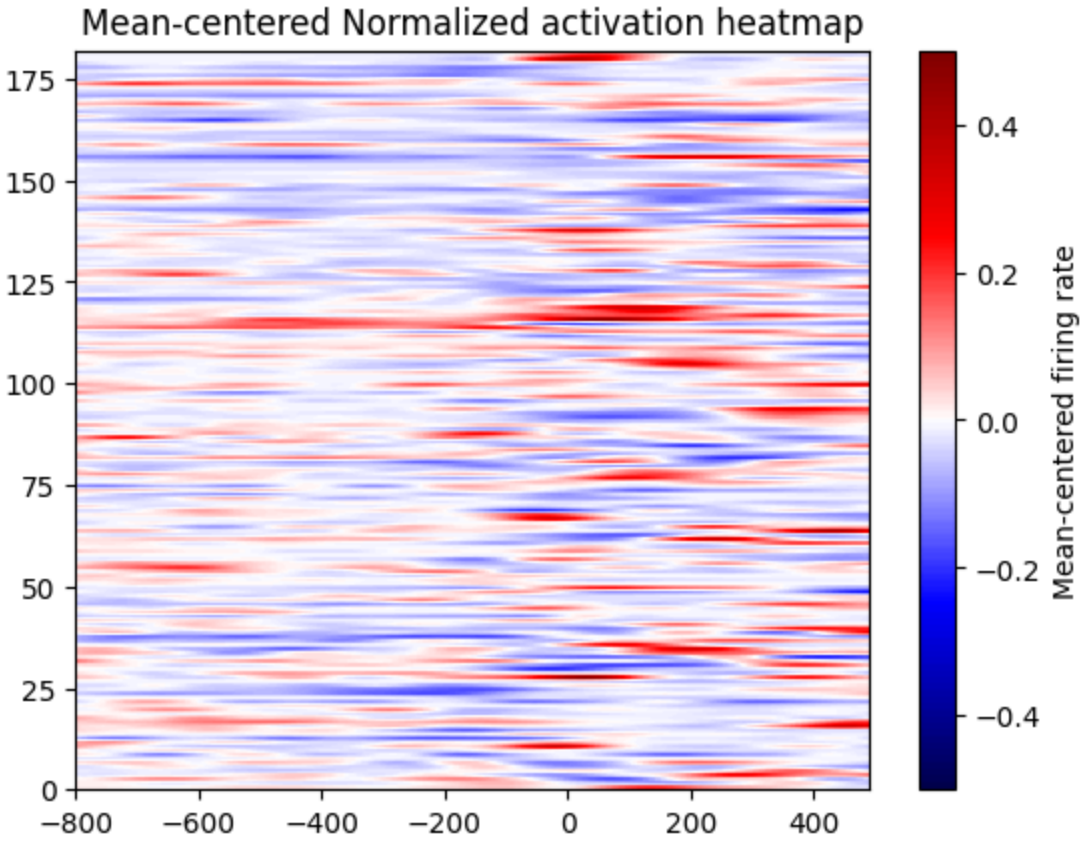



> Q: Why do you think this normalization step will be helpful? (You can come back to this question after going through all exercises.)

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))
```

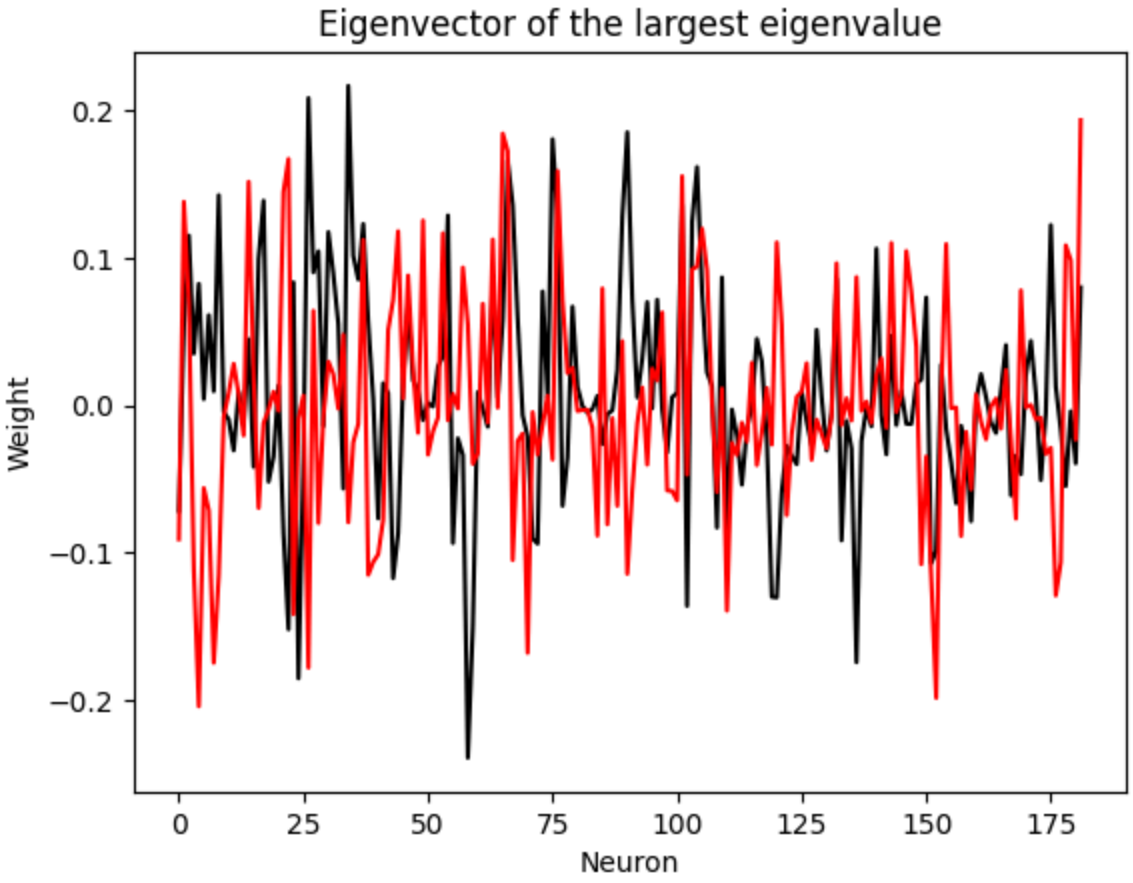
```
Shape of S (182, 182)
Shape of Z_sk (12, 4968)
```

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)
```

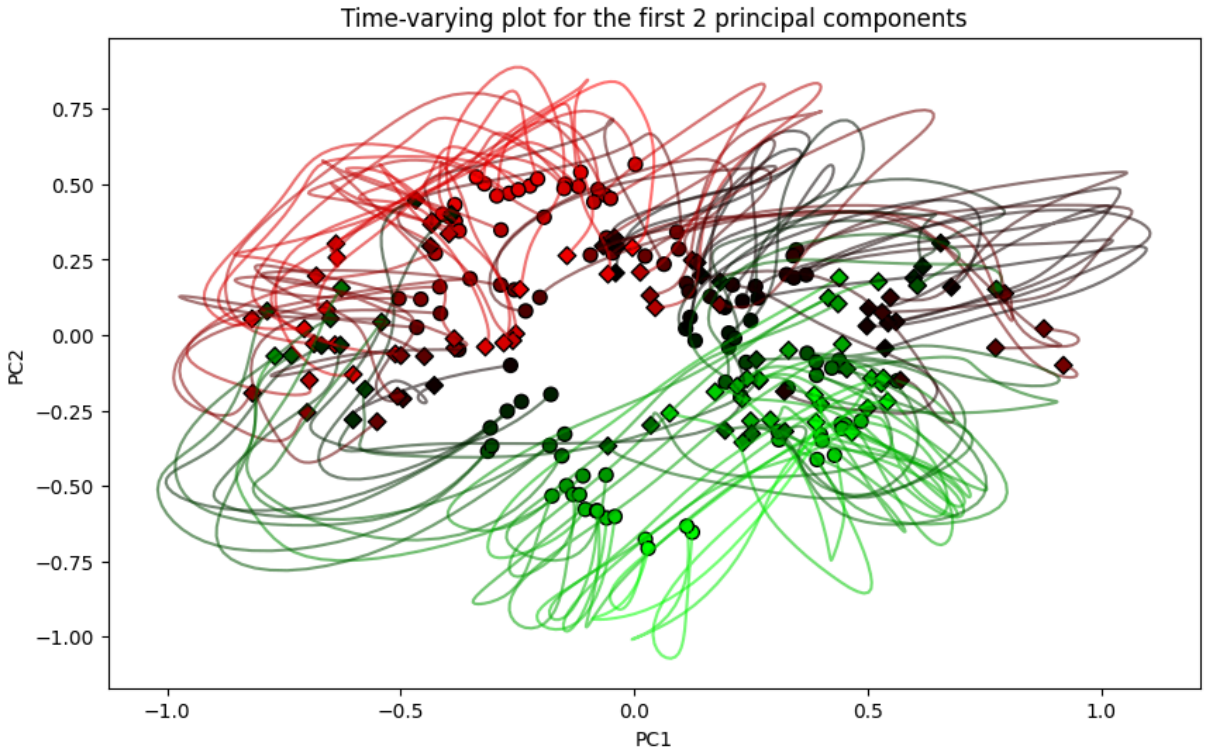
```
Shape of X_new (182, 4968)
Shape of Z_pca (12, 4968)
```

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…

```
# 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…

```
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)
```

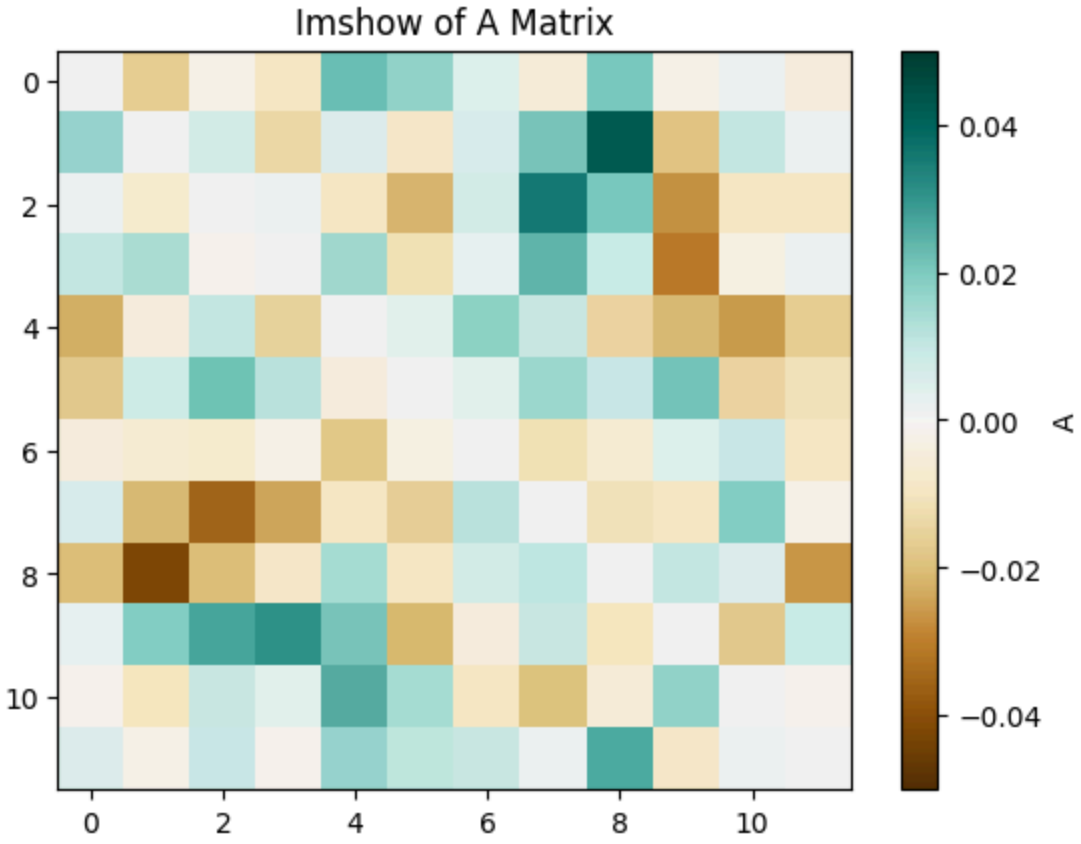
In [662…

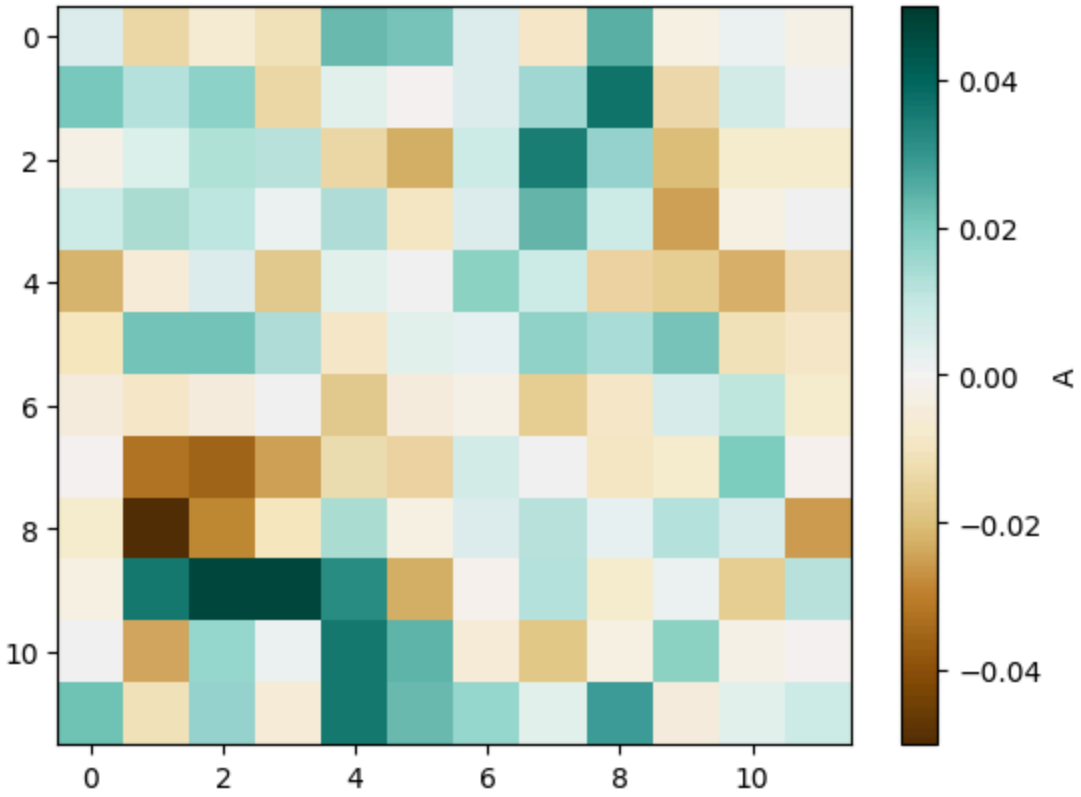
```
# Taking derivative of the loss function with respect to A
# An important note here is that W is constructed with Z (with last shape T) which
def gen_W(H:np.ndarray, Z):
    return np.tensordot(H, Z, axes = 1)


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)
```

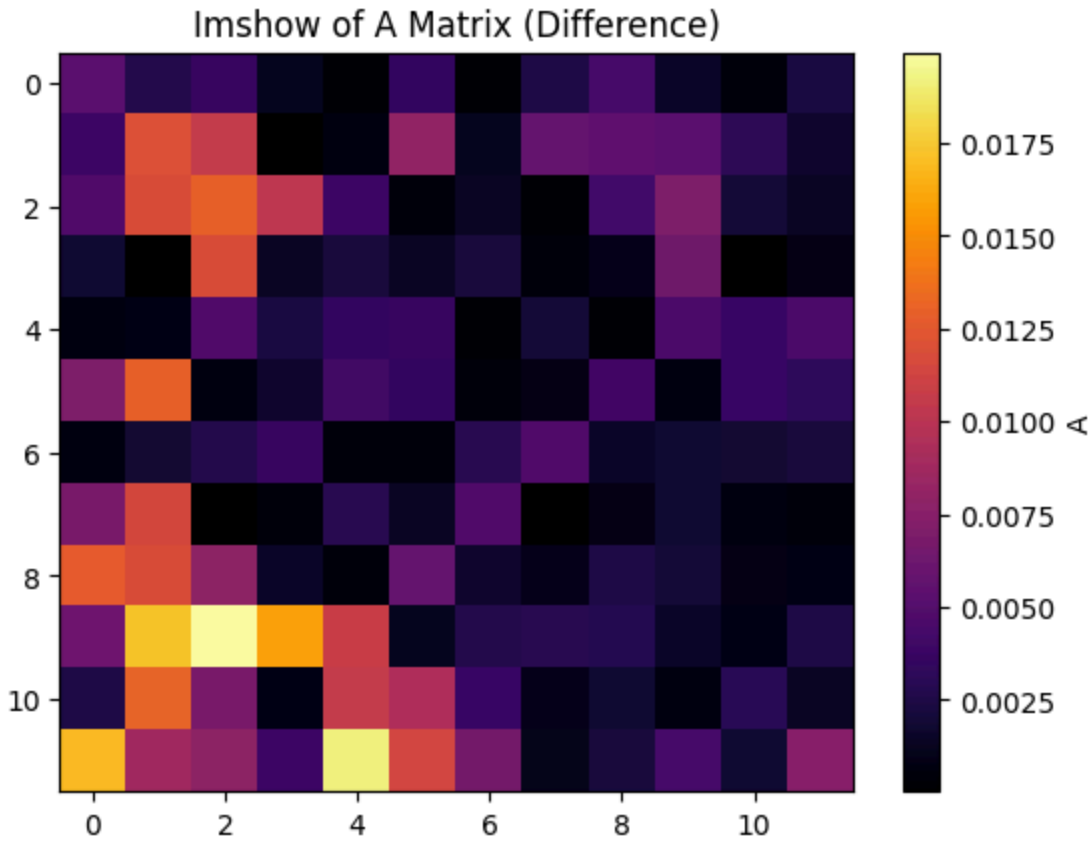
```
Shape of W (66, 12, 4860)
Shape of Z (12, 4860)
Shape of b_vec (66,)
Shape of Q (66, 66)
Shape of beta_pred (66,)
```

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
0
2
4
6
8
10
0
2
4
6
8
10
0.04
0.02
0.00
−0.02
−0.04
A


Imshow of A Matrix (MLE)
0
2
4
6
8
10
0
2
4
6
8
10
0.04
0.02
0.00
−0.02
−0.04
A



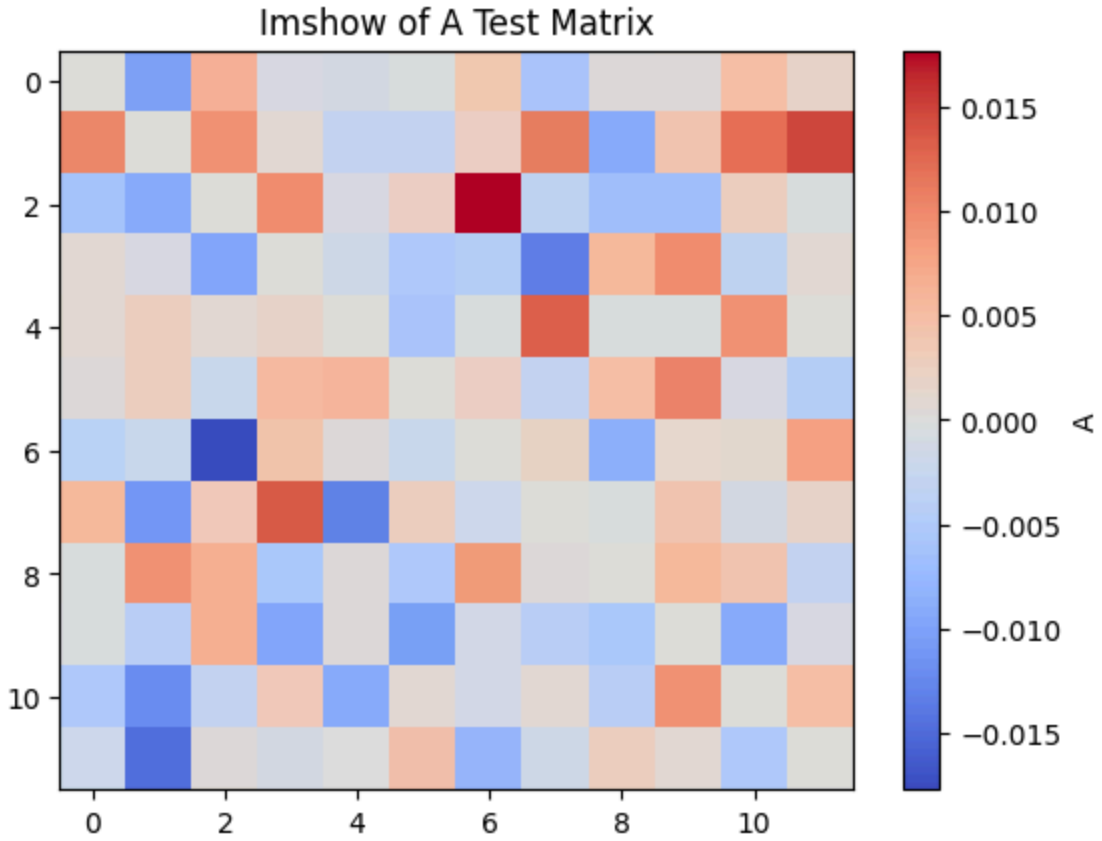

```
Processing for  0
Processing for  500
Processing for  1000
Processing for  1500
Processing for  2000
Processing for  2500
Processing for  3000
Processing for  3500
Processing for  4000
Processing for  4500
LL for MLE -6.484043435044501
LL for Rotational -7.327843832956184
```

In [596…

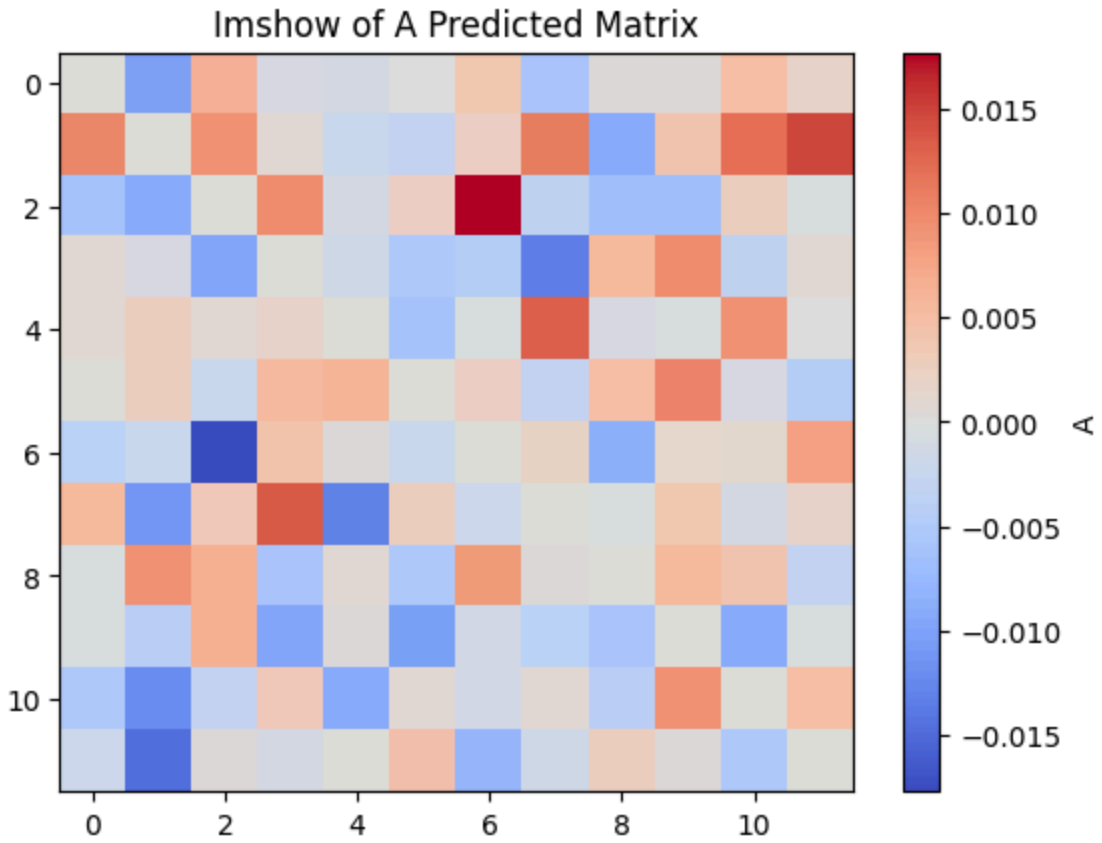
```
test_data = np.load('data/test.npz')
Z_test, A_test = test_data['Z_test'], test_data['A_test']

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)
```

```
    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)
--------------------------------------------------------------------------------
---------------
```

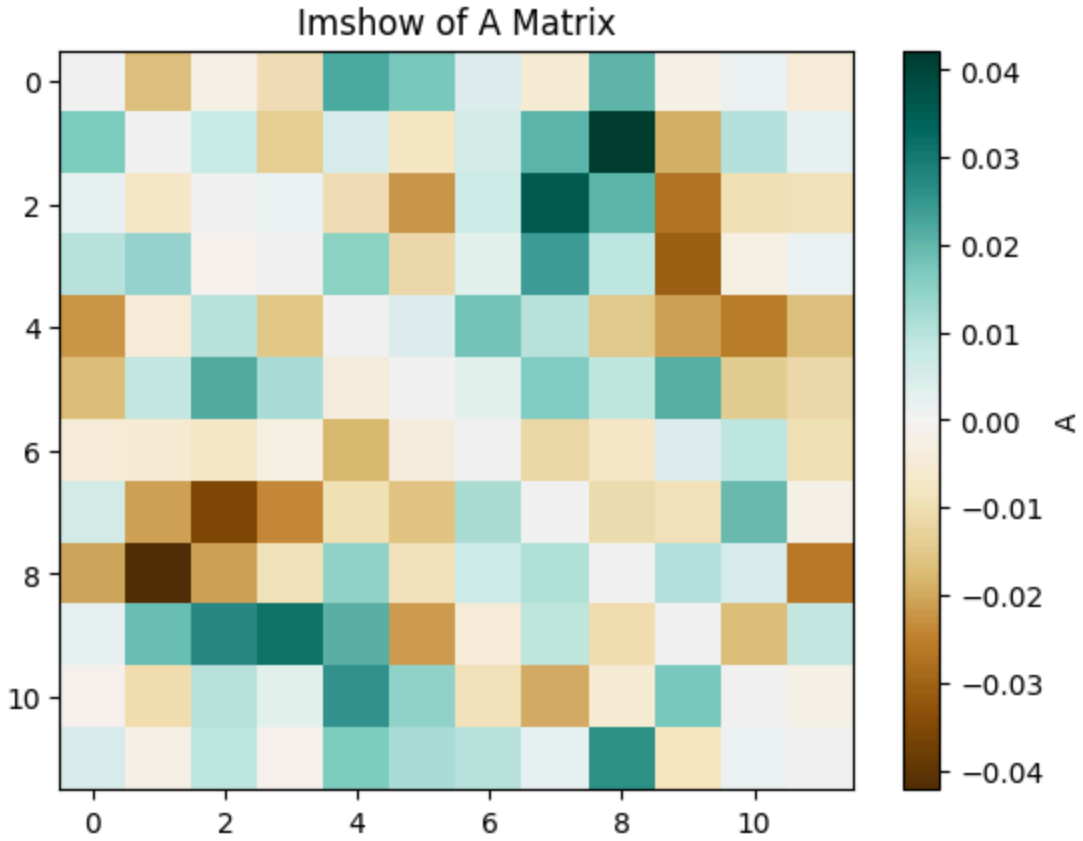



```
Mean Absolute Error for A matrix 2.029546462186608e-05
```

# 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)
```



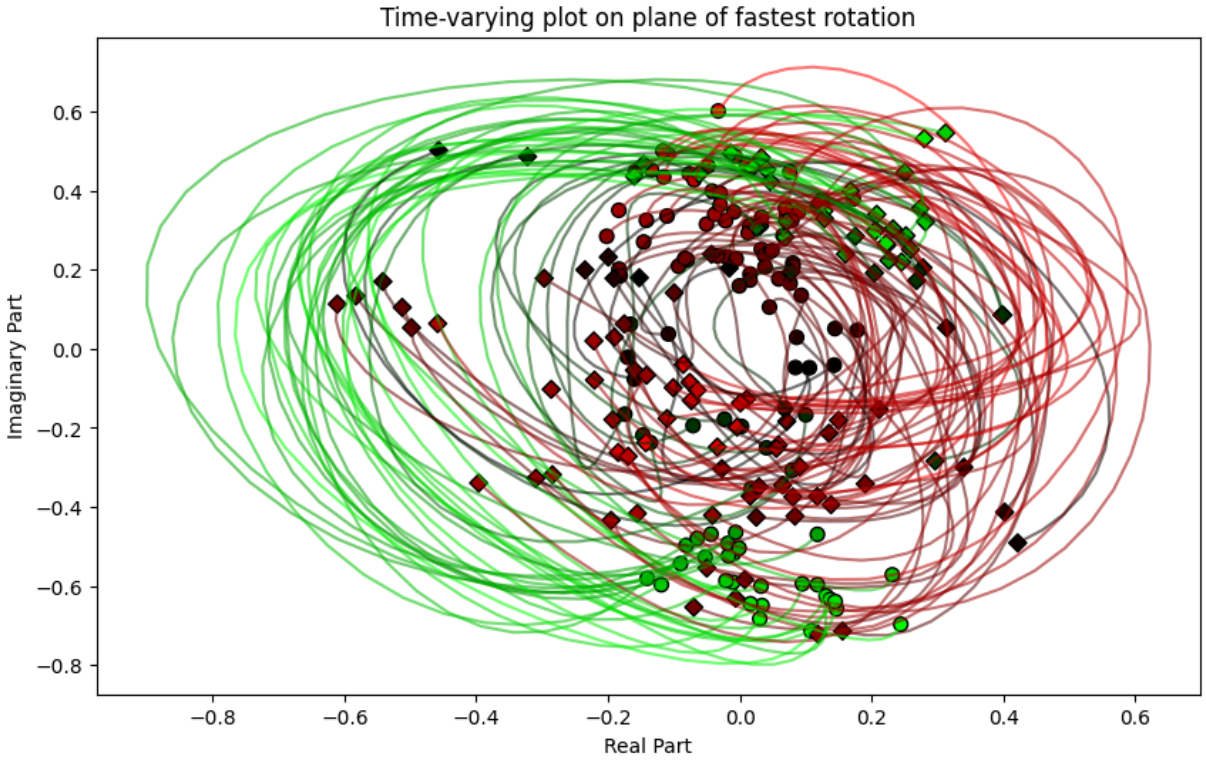

```
Shape of eig (12, 12)
All eigenvalues for A_test
[ 1.05709712e-18+0.09214251j  1.05709712e-18-0.09214251j
 -6.93889390e-18+0.06661613j -6.93889390e-18-0.06661613j
 -2.16840434e-18+0.04475907j -2.16840434e-18-0.04475907j
  1.73472348e-18+0.01605887j  1.73472348e-18-0.01605887j
  5.96311195e-19+0.00284482j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

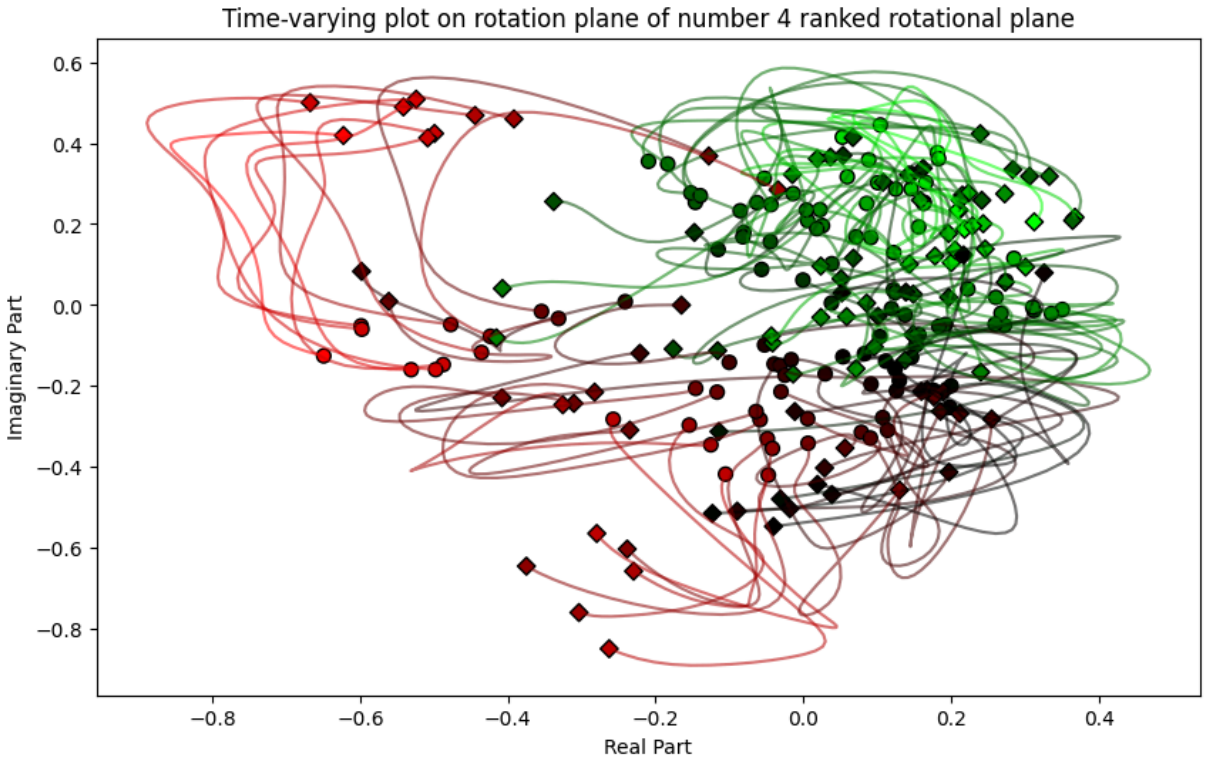
A: We see better path separation, and more 'extreme' trajectories. This makes sense given the projection onto the high-angular speed plane, the trajectories are more skewed and longer lasting (even as the time interval has been cropped to [-150, 200]). There is more 'rotational movement' in this plane - 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))
```

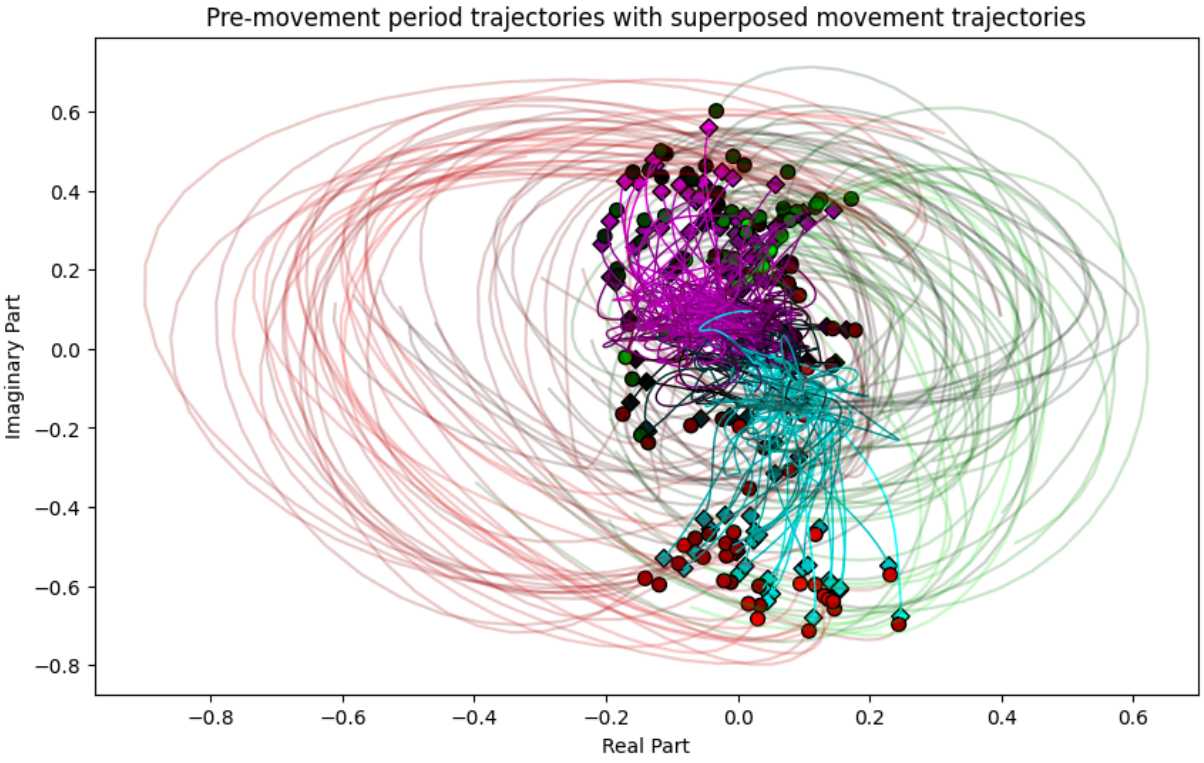
```
Shape of V_m (182, 12)
Shape of P_fr (2, 12)
Shape of pfrvmt (2, 182)
Shape of X (182, 108, 130)
Start index 0
End index 65
Shape of X_6 (182, 108, 66)
```

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…

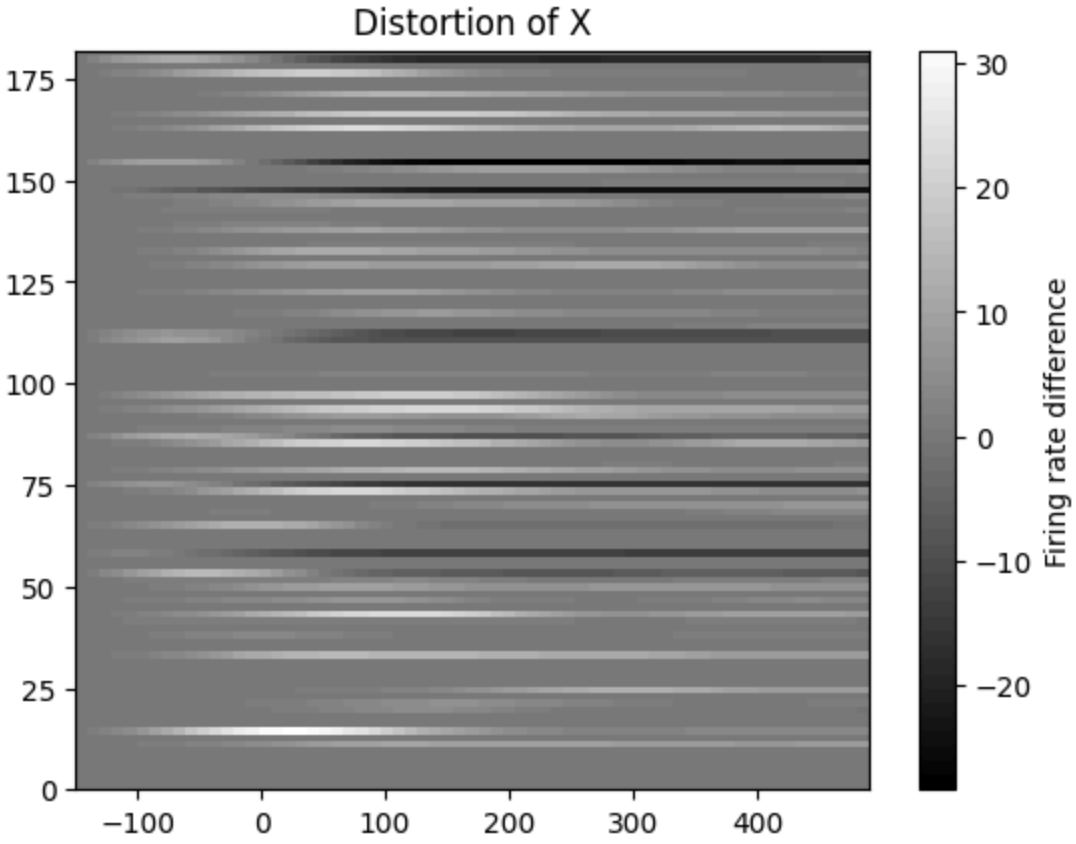
```
# Rerunning with distortion

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…

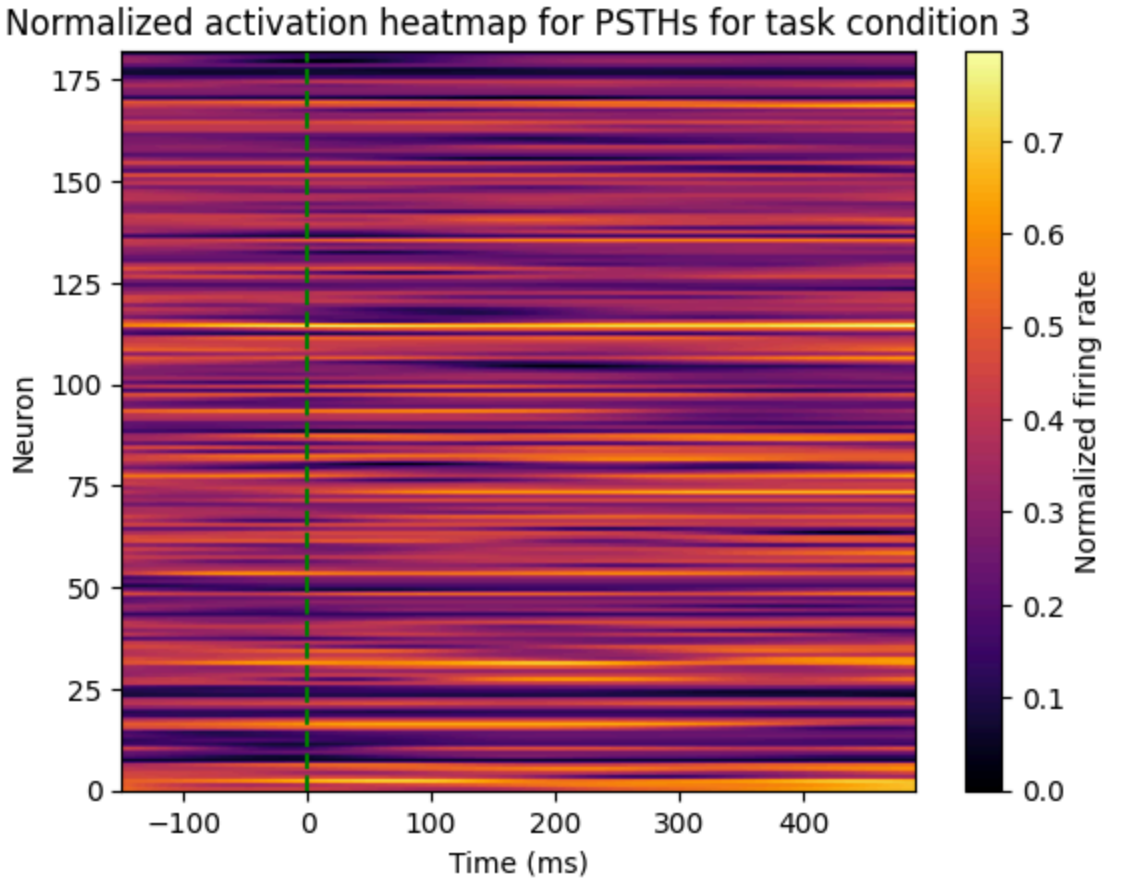
```
# Distortion for X_mp
C_rands = np.random.choice(C, C//2, replace=False)
X_mp_distort = np.zeros_like(X_mp)

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()
```

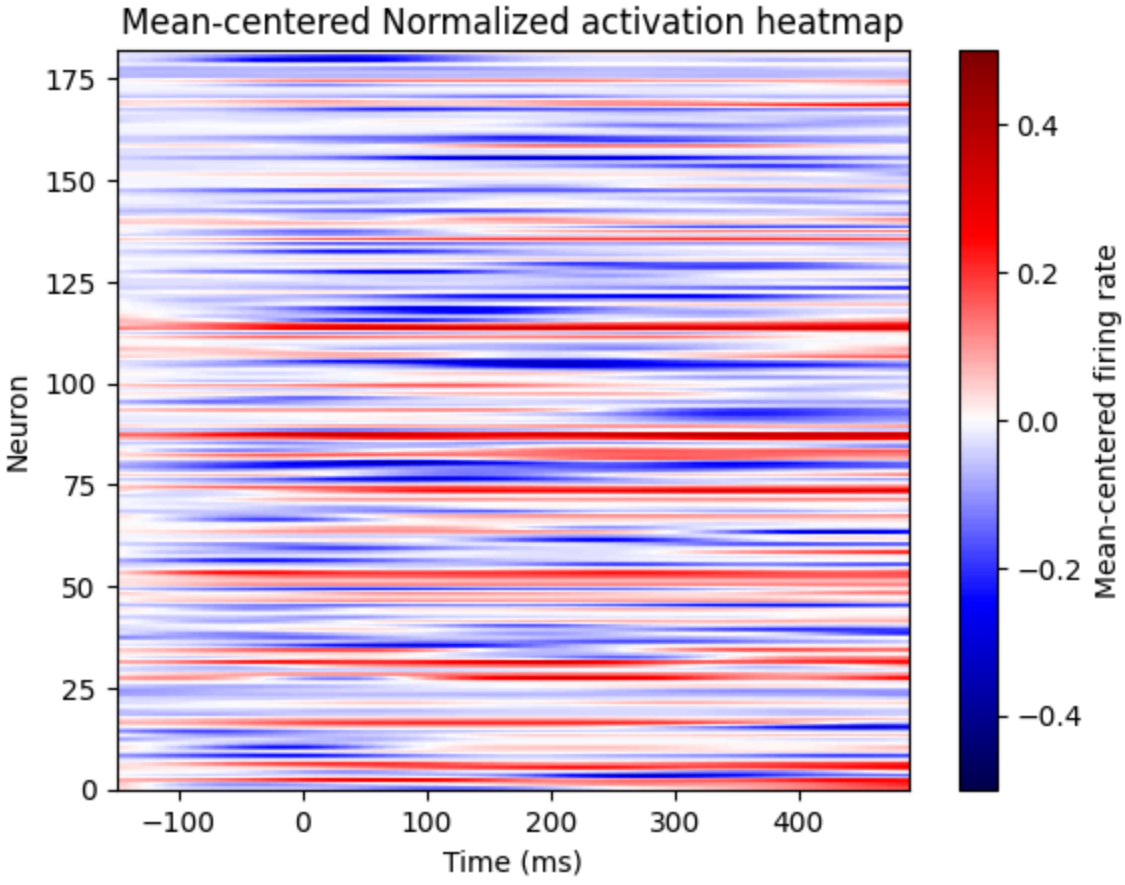
```
Shape of C_rands (54,)
```




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
0
25
50
75
100
125
150
175
−100
0
100
200
300
400
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Mean-centered Normalized activation heatmap
Neuron
Time (ms)
Mean-centered firing rate
0
25
50
75
100
125
150
175
−100
0
100
200
300
400
−0.4
−0.2
0.0
0.2
0.4

In [640…

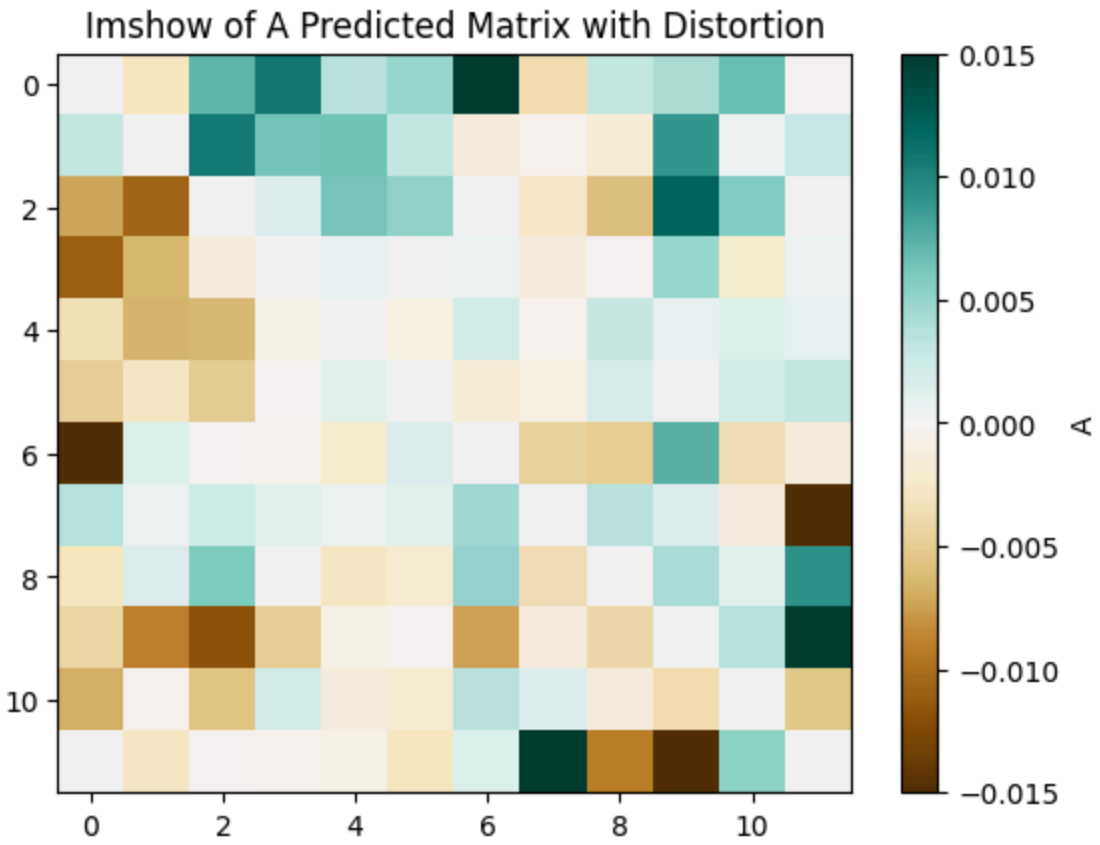
```
X_new = X_mean_centered.reshape(X_mean_centered.shape[0], -1)
print('Shape of X_new', np.shape(X_new))

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()
```

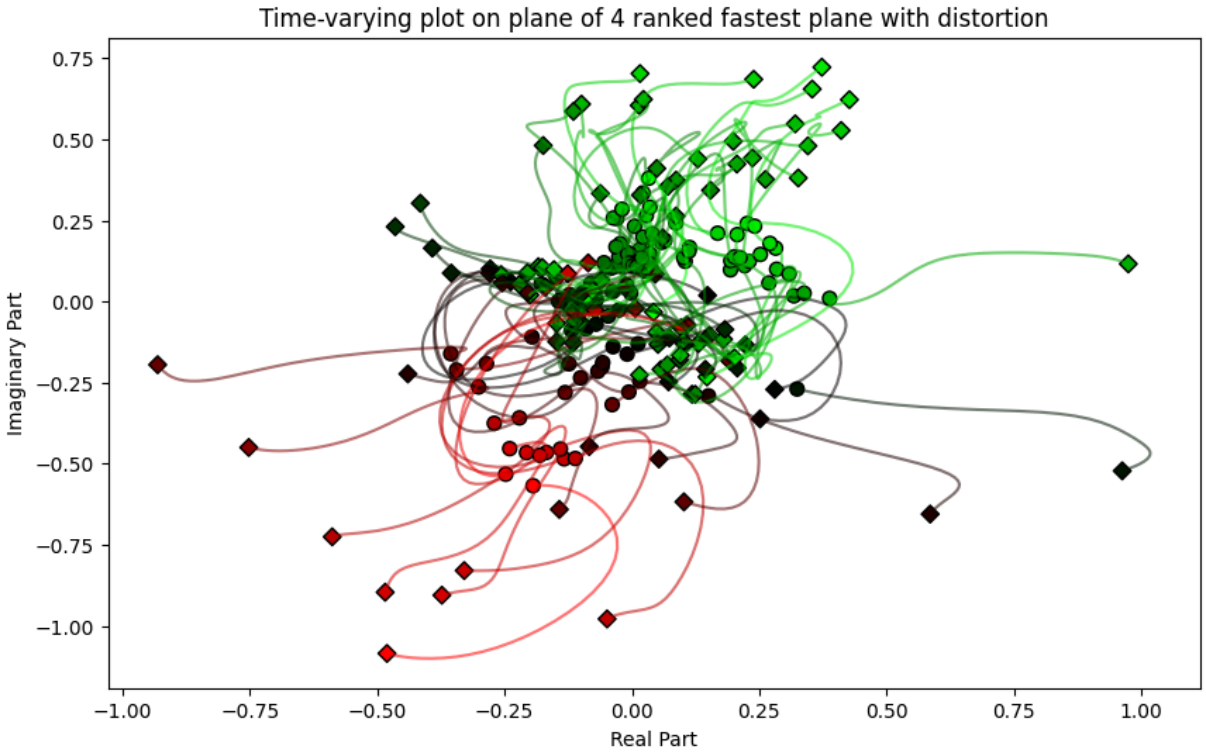
```
Shape of Z_7 (12, 108, 65)
--------------------------------------------------------------------------------
---------------
Shape of Z_pca (12, 108, 65)
Shape of Z (12, 6912)
Shape of delta_Z (12, 6912)
--------------------------------------------------------------------------------
---------------
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 [ ]: