Coursework for
4G10 Brain-Machine Interfaces
Assignment 1

Yashar Ahmadian

assigned: November 3, 2023
due: November 18, 2023, 4:00pm

submit through Coursework 1 Submission on the Moodle.
don't forget to attach a cover page

In this coursework you will apply a dynamical dimensionality reduction method to uncover rotational dynamics in the motor cortex. Please carefully read these bullets:

- Your report must be a maximum of six A4 pages excluding any appendix (minimum font size 11pt, minimum margins 1.5cm on each side).
- In the cover page, please clearly include your candidate number, **NOT** your name (anywhere).
- Section 4 Exercises contains the things you have to do or implement. The **Q**'s in bold appearing there indicate questions that you need to answer in some form in your report, which should be structured mirroring the numbering of the exercises/steps in Section 4.
- You are very much encouraged to think of data/results visualizations to best support the exposition of your results. You are also encouraged to report on any specific problems/difficulties that arose in your implementation of the various algorithms, and how you addressed those.
- Include all your code in an appendix to your report; if you find it easier, you can instead include a link to a public repository containing your code. (This code will not be assessed or inspected; it just serves as a guarantee that you have done the work yourself.)
- For non-Python coders: While the few programming tips given in some footnotes are for Python/Numpy, you are not obligated to code in that language (and even if you are coding in Python/Numpy, you don't have to follow the tips). But if you choose not to use Matplotlib/PyPlot for your plotting, then you will need to translate the three short functions in `cond_color.py` (see below) to your language of choice, ideally with similar results (in terms of plot colors and approximate marker shapes).

1 Background

An influential theory of the motor cortex function holds that this cortical network forms a dynamical system that, even in the absence of informative external inputs, can generate its own autonomous dynamics and corresponding activity patterns. These neural activity patterns are then sent downstream as motor commands to the spinal cord, thereby shaping body movements.

Autonomous dynamical systems are characterized by the property that starting at the same state, the instantaneous change in the state is always the same. In other words, the time derivative (or, in the case of discrete time, the one-step change) is a deterministic[1] function of the state vector. In our case (as in the lectures), the latter is the joint activity of motor cortex neurons, or a low-dimensional projection of it.

But what form does this dynamics take? A clue comes from observations of animal motor behaviour. Many animal movements are of a repetitive, periodic nature: examples including walking, running, chewing, the peristalsis of the gut, etc. Indeed, periodic movements are ubiquitous across the animal kingdom and have been evolutionarily preserved, being repurposed for different functions in different species. It is thus reasonable to assume that the neural dynamics underlying these preserved patterns, also underlie movements that are not overtly periodic.

Periodic dynamics are mathematically described by circular or rotational trajectories in the state space. This coursework explores the hypothesis that low-dimensional rotational dynamics underlie the patterns of recorded neural activity in the motor cortex.

As in the models discussed in lectures, we will adopt a discrete-time dynamical framework to model the time series of (in this case trial-averaged) neural activity, $\boldsymbol{x}_t$. For N recorded neurons $\boldsymbol{x}_t \in \mathbb{R}^N$. We hypothesize that there exists a low-dimensional projection $\boldsymbol{z}_t$ (in $\mathbb{R}^M$ with $M \ll N$) of the activity which exhibits linear time-invariant *rotational* dynamics. More precisely, we assume that, after proper centering (since rotations happen around a center), $\boldsymbol{z}_{t+1}$ is a small rotation applied to $\boldsymbol{z}_t$. Let us first consider a 2-dimensional example (i.e. $M = 2$). In two dimensions, a rotation by angle θ around the origin is described by a linear transformation with the matrix representation:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

We thus assume

$$\boldsymbol{z}_{t+1} \approx R(\theta)\, \boldsymbol{z}_t,$$

for some rotation angle θ. If the dynamics are smooth and Δt is small, we expect θ to be small. In this case expanding the matrix elements to 1st order in θ we obtain $R(\theta) \approx \begin{pmatrix} 1 & -\theta \\ \theta & 1 \end{pmatrix} = I + A$, where I is the 2×2 identity matrix, and we defined the *antisymmetric* matrix

$$A = \begin{pmatrix} 0 & -\theta \\ \theta & 0 \end{pmatrix}.$$

Plugging this approximate form for $R(\theta)$ in the equation above we obtain $\boldsymbol{z}_{t+1} \approx \boldsymbol{z}_t + A\boldsymbol{z}_t$ or

$$\Delta \boldsymbol{z}_{t+1} \approx A\, \boldsymbol{z}_t \tag{1}$$

where $\Delta \boldsymbol{z}_{t+1} = \boldsymbol{z}_{t+1} - \boldsymbol{z}_t$. It turns out that this approximation generalizes to rotations in higher dimensions, namely matrices describing *small* rotations can always be written approximately as the identity matrix plus an antisymmetric matrix.

2 The model

As in the LDS models developed in the lectures, one could construct a generative model for $\boldsymbol{x}_t$, in which the low-dimensional latent variables obey Equation (1), or the following, more mathematically precise, stochastic version of it:

$$\Delta \boldsymbol{z}_{t+1} = A\, \boldsymbol{z}_t + \sigma \boldsymbol{\epsilon}_t \qquad \text{with} \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I_{M\times M}), \tag{2}$$

[1] When analysing biological data, we of course expect this to only be approximately true.

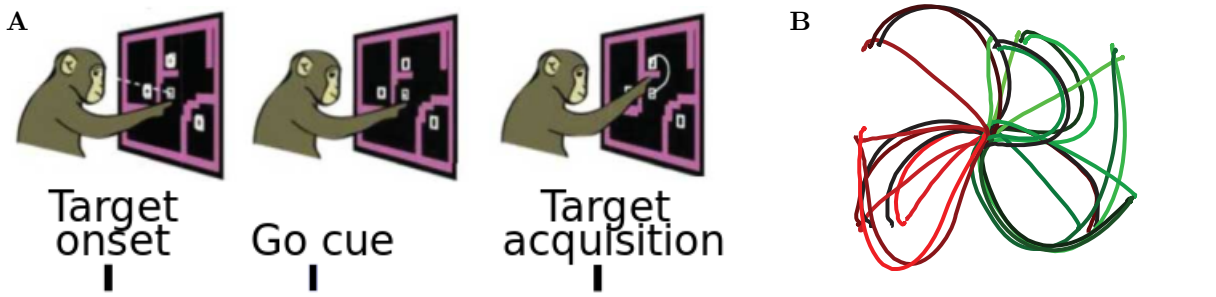

Figure 1: **A:** The sequence of steps in the delayed center-out reach task performed by the monkeys. The monkey has to point to targets appearing on the screen by moving its finger on the screen. The screen may also show the (pink) walls of a "maze" which have to be avoided by the monkey. The target or targets appear first at "target onset", but the monkey is tasked to move its finger only after a "go cue" appears, with a variable delay (0.5-1 seconds) after target onset. The provided times (in milliseconds) for the PSTH bins are measured *relative to the movement onset*, which in turn follows the go cue after a variable reaction time. **B:** Examples of a monkey's trial-averaged hand trajectories in different task conditions.

where, as above, A is an *antisymmetric* matrix, such that $A_{ij} = -A_{ji}$. In general, in such a model, one infers the latent $\boldsymbol{z}_t$ from the observed $\boldsymbol{x}_t$ using Bayes' rule. In the method that you will develop here, however, such a generative model for the high-dimensional $\boldsymbol{x}_t$ is *not* constructed. Instead, the method proceeds in two steps:

1. First, (non-probabilistic) PCA is used to obtain $\boldsymbol{z}_t$ as the M-dimensional projection of $\boldsymbol{x}_t$ onto the first M principal components of $\boldsymbol{x}_t$.
2. We then treat the obtained $\boldsymbol{z}_t$ as observed variables in their own right, and fit the matrix A of the model Equation (2) to the $\boldsymbol{z}_t$ time series.

Here you will apply this method to a dataset of trial-averaged spike counts for N neurons, recorded in C experimental conditions corresponding to different settings of a motor task performed by a monkey. The task is a so-called delayed center-out reach task which is described in Fig. 1; the conditions correspond to different target and obstacle locations (Fig. 1A) resulting in different average hand trajectories (Fig. 1B).

Let $\boldsymbol{z}_{c,t}$ denote the low-dimensional trajectory in condition c (where $c \in \{1, \ldots, C\}$). A key assumption (informed by our hypothesis) will be that the autonomous dynamics is the same in all conditions and thus $z_{c,t}$, for all c, are governed by Equation (2) with the same A and σ (but independent $\boldsymbol{\epsilon}_{c,t}$, which are also independent across time).

3 Dataset

Download the data file `psths.npz` and the Python module `cond_color.py` from the "Coursework 1 Materials" folder on the course's Moodle page. The data file contains:[2]

- a 3D Numpy array, X, with shape $N \times C \times T$, containing the so-called PSTH's of $N = 182$ neurons in $T = 130$ time-bins and $C = 108$ task conditions. PSTH stands for "peristimulus time histogram"

[2]You can load the data using: `data = numpy.load('psths.npz')` followed by `X, times = data['X'], data['times']`.

and refers to the sequence of average spike counts or firing rates of a neuron in different time bins in a time interval around the onset of a stimulus or a movement. In our case, the interval goes from -800 ms (milliseconds) to +500 ms relative to the onset of hand movement; the interval was divided into 130 time bins of 10 ms width. As is commonly done, the trial-averaged spike counts have been divided by the bin width (in units of seconds), such that `X[i, c, t]` is the average firing rate of neuron `i` in the `t`-th time bin in condition `c` (in units of Hz or spikes per second).

- A 1D array, `times`, with the start time (in milliseconds) of the different PSTH bins relative to movement onset (see Fig. 1A).

4 Exercises

1. **Plotting raw PSTHs:** Plot the neurons' PSTHs. In your report include plots for a reasonably representative handful of neurons in a small subset of conditions. Use the `times` array to label the times on the x-axis.
   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.
   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.

2. **Preprocessing:**

   (a) **Normalization:** Plot a histogram of the neurons' maximum (across conditions and time) firing rates. Then, separately for each neuron,[3] normalize its PSTH according to:
   `psth = (psth - b) / (a - b + 5)`
   where `psth` is the PSTH of the neuron in all conditions, and `a` and `b` are, respectively, the maximum and minimum value of this neuron's PSTH across *both* times and conditions. This step ensures that the normalized activities of different neurons have the same scale and approximate range of variations. Henceforth (unless otherwise stated) we will work with this mean-centered and normalized PSTH array, which we keep denoting by X.
   Q: Why do you think this normalization step will be helpful? (You can come back to this question after going through all exercises.)

   (b) **Mean centering:** Next, remove from X its cross-condition mean (calculated and subtracted separately for each time bin and neuron[4]).

   (c) **Dimensionality reduction by PCA:** From this step until exercise 6 below we will only work with the PSTHs limited to the interval from -150ms to $+300$ms relative to movement onset (we will however keep using X to denote the corresponding slice of the normalized and mean-removed PSTH array, and use T to denote the number of time bins, now equal to 46,[5] in this interval). Reshape X into a $N \times CT$ matrix by combining its condition and time axes, then use PCA to obtain a dimensionality-reduced version of it by projecting onto the first $M = 12$ principle components in the neuron activity space. Denote the resulting $M \times CT = 12 \times 4968$ array by Z and its components by $Z_{i,n}$. To obtain Z, use Equation (18) of the notes on the pPCA lecture, `02_pPCA_notes.pdf` (and NOT Eq. (17) of the notes). Keep the constructed matrix V_M (appearing in Eq. (18) of the notes and defined therein) for later use below in exercise 6.

[3] Feel free to use Numpy broadcasting/indexing to nicely achieve this in one code step, without using a for loop over neurons.
[4] You can again achieve this more efficiently by using Numpy broadcasting, without for loops over neurons and time bins.
[5] Or 45, depending on whether you include the last time bin at (or more precisely, starting at) +300ms or not.

3. **Plotting PC space trajectories:** Make a plot of trajectories in the PC1-PC2 plane (corresponding to 0 and 1 left-indices of Z, which for plotting you would reshape back into a 3D array). Superimpose the trajectories for all conditions in the same plot. To standardize plots and make them more readable, use the provided function `cond_color.get_colors`[6] which receives as input the x and y coordinates of the initial points (*i.e.*, at -150 ms) of the trajectories in different conditions, and outputs the colors in which you should plot the trajectories of different conditions.[7] The returned colors are in a format that can be used with `matplotlib` plotting functions. Also use the `cond_color.plot_start` and `cond_color.plot_end` functions to mark the start and end of each trajectory with special markers, in the same color as the trajectory (play with the `markersize` input to these functions to achieve a reasonable marker size).

4. **Finding the maximum-likelihood estimate for A:**

   (a) **Log-likelihood and its (naive) gradient:** Write an expression for the log-likelihood (to be maximized in terms of A) of the model in Equation (2), given the data contained in Z; ignore terms that are independent of A, and set σ to 1 for simplicity. Derive an expression for the gradient of the log-likelihood with respect to A; you may find it easier to write matrix products, etc., as explicit sums over indices and then take partial derivatives with respect to individual matrix elements A_{ij}. Eventually express the gradient in matrix form in terms of Z and ΔZ, where ΔZ is defined to be the 2D array containing the temporal-differences of Z.[8] (Do read footnote 8!)

   (b) **Parametrising an antisymmetric A:** The vanishing of the gradient obtained in the previous part yields a vector equation (equivalent to a set of linear equations) that is satisfied by the maximum-likelihood estimate of an *unconstrained* A. However, our aim is to obtain an *antisymmetric* matrix that maximizes the likelihood, and thus A is not unconstrained (it satisfies the constraint $A_{ij} = -A_{ji}$). We can avoid performing constrained optimization by constructing the antisymmetric A from its above-diagonal elements, which constitute K unconstrained and non-redundant parameters.
   Q: What is K (for general M and for $M = 12$)?
   Denote the K-dimensional vector of these parameters by $\boldsymbol{\beta}$, with components β_a ($a = 1, \ldots K$) corresponding to A_{ij} ($j > i$) ordered in a fixed way, say, according to the row-major order; we take $\boldsymbol{\beta}$ to be a row vector. Since the elements of A are linearly related to β_a (in an obvious way), we can write the elements of A as linear combinations of β_a:

$$A_{i,j} = \sum_{a=1}^{K} \beta_a H_{a,i,j} \tag{3}$$

   where the coefficients $H_{a,i,j}$ take values in $\{-1, 0, +1\}$. Construct the 3D array H that achieves this, and test it by making sure it constructs the correct A via Equation (3) applied to an example $\boldsymbol{\beta}$ of your choice (with non-repetitive components); you can run the test for small M, say 4.[9]

   (c) **Gradient with respect to $\boldsymbol{\beta}$:** Derive an expression for the gradient of the log-likelihood with respect to $\boldsymbol{\beta}$, using Equation (3) within the expression derived in exercise 4a for the log-likelihood.

[6] Or your own equivalent, if you will use another language, ideally with reasonably close color matches.

[7] This function finds the direction along which the initial points are most widely spread across conditions, and then bases the color of different conditions on the location of that condition's initial point along this direction.

[8] When you code this up in later parts, make sure to avoid taking differences across the end and start time bins in different conditions. So first reshape Z back into a 3D array with shape $M \times C \times T$, before taking differences along the time axis. The corresponding ΔZ will be of shape $M \times C \times (T-1)$ and therefore you should now understand Z (as it enters the expression for the gradient) as denoting an *appropriate* slice (or sub-array) of Z with the same shape as ΔZ. Finally, you will appropriately reshape these into 2D arrays of the same shape.

[9] In Numpy you can achieve the multiplication in Equation (3) using `numpy.tensordot(beta, H, axes=1)`.

Hint: before taking any derivatives, express the $M \times C(T-1)$ matrix product AZ, which appears in the log-likelihood, in terms of $\boldsymbol{\beta}$ and the $K \times M \times C(T-1)$ 3D array W, defined via[10]

$$W_{a,i,n} = \sum_{j=1}^{M} H_{a,i,j} Z_{j,n}. \tag{4}$$

Eventually, express the gradient in the form $\boldsymbol{b} - \boldsymbol{\beta} Q$ for some $K \times K$ matrix Q and K-dimensional row vector $\boldsymbol{b}$; find and express Q and $\boldsymbol{b}$ in terms of W and ΔZ, properly reshaped if needed.

(d) **An antisymmetric estimate for A:** Using the obtained expression for the gradient, compute the maximum-likelihood estimate for $\boldsymbol{\beta}$ and the corresponding A.[11] Make a color plot of A.[12]

(e) **Test:** To test your full algorithm for estimating A (written as a function that only receives Z and outputs the estimate for A), download the file `test.npz`, which contains two arrays, `Z_test` and `A_test`. Pass `Z_test` to your function and check that the maximum absolute value differences between elements of your estimated A and the `A_test` array from the file (the ground truth A), are small (say, do not exceed 10^{-8}, although achieving a much better accuracy is also possible[13]).

5. **2D projections with rotational dynamics:**

(a) For even M, every rotation in M-dimensional space can be decomposed into independent 2D rotations in $M/2$ orthogonal 2D planes (that only meet at the origin). In the case of our matrix A, which represents an infinitesimal rotation, these special 2D planes are related to the eigenvectors of A, as follows. The eigenvalues of an antisymmetric matrix are all pure imaginary, and come in complex conjugate pairs; in other words, they are all of the form $\pm j\omega$ for real and positive ω (where $j = \sqrt{-1}$). The pair of eigenvectors corresponding to a pair of complex conjugate eigenvalues are also complex conjugates of each other. It turns out that the real and imaginary parts of (either one) of these two eigenvectors constitute a pair of orthogonal real vectors that span one of the $M/2$ special 2D planes. The imaginary part of the corresponding eigenvalue (*i.e.*, ω in the above notation) then provides the angular velocity of the rotation in that special 2D plane.
Find the eigenvalues and eigenvectors of the estimated A.[14]

(b) Focus first on the eigenvalue with the largest imaginary part; this eigenvalue corresponds to the fastest special 2D rotation induced by A. Construct the $2 \times M$ matrix P with its two rows given by the *normalized* real and imaginary parts of the eigenvector corresponding to this eigenvalue. (Note that the real and imaginary part vectors need to be first normalized by you to have unit length.) By applying P to Z obtain the special 2D projection of the M-dimensional trajectories, in the special plane with the fastest rotation. We will call this special plane the plane of fastest rotation, or FR plane for short, and will call the corresponding projection matrix P_{FR}.

(c) Plot these 2D trajectories using the same plotting and coloring conventions as in exercise 3. To obtain a less busy plot, plot the trajectories in the sub-interval [-150ms, +200ms].
Q: How do these trajectories differ (qualitatively) from those in exercise 3?

(d) Repeat the previous two steps 5b-5c, but now plot the 2D projections onto the planes corresponding to the eigenvalues with the second and third largest imaginary parts.
Q: How do these trajectories differ from those projected in the plane corresponding to the fastest 2D rotation? Speculate why.

[10] You can again achieve the multiplication in Equation (4) using `numpy.tensordot(H, Z, axes=1)`.
[11] To solve a linear system of equations you can use `numpy.linalg.solve`; be careful with row- vs. column-vector conventions!
[12] Using, *e.g.*, `plt.imshow`.
[13] That said, as long as the relative errors of your estimate are on the order of 10^{-3} or so, your later results should be fine.
[14] If you use `numpy.linalg.eig`, the complex conjugate pairs of eigenvalues and eigenvectors will have adjacent indices in the returned outputs (but do make sure this is the case, and also read the help carefully).

6. **Pre-movement period:** Apply the projections obtained for the interval [-150ms, 300ms] to the interval [-800ms, -150ms], which we will refer to as the pre-movement period. Specifically, combine the exact same PC-projection matrix V_M obtained in exercise 2c with the projection onto the FR plane found in 5b to obtain the $2 \times N$ projection matrix $P_{\text{FR}} V_M^{\text{T}}$ (where $^{\text{T}}$ denotes transposition), which you will use to directly project the N dimensional trajectories during this pre-movement period (corresponding to an appropriate slice or sub-array of the normalized and mean-centered X obtained at the end of exercise 2b) onto the FR plane. Plot these trajectories superimposed on those plotted in 5c (the movement interval trajectories), but with a different color code: green replaced by cyan and red by magenta. To this end, use `cond_color.get_colors` as in 3 and 5c, but with the optional input `alt_colors` set to `True`, and with the `xs` and `ys` inputs now giving the coordinates of the *final* points of the pre-movement period trajectories. (In order to make this joint plot less busy, feel free to reduce the `alpha` value[15] of the plotted trajectories during the movement period, which were already plotted separately in 5c.)

   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)?

7. **Control analysis:** One concern about fitting our rotational dynamics model and the corresponding "rotational plane" projections to data (particularly when M is large and the number of conditions is not large enough) is that the model may overfit and "hallucinate" rotational dynamics; in other words, the method may have found rotational looking patterns even when rotational dynamics is not truly present. To address this concern, one can run various control analyses, by fitting the model to distorted data (distorted using appropriate shuffles, inversions, etc.) that conserve certain properties of or structures in the data, but are expected to destroy any potential, truly existing rotational dynamics. If the model fit still yields 2D projection plots that exhibit rotational trajectories, we conclude that this rotational dynamics was hallucinated and not real.

   The control distortion you will implement is to invert (only) the movement-period segment of the PSTH's in (only) a randomly picked half of the movement conditions, around their value at the begining of this period (*i.e.*, the value at the -150ms time bin); this assures that the distorted PSTH's remain continuous. This random choice of conditions is to be made independently for different neurons.[16] More precisely, if `x` is the PSTH of a single neuron in one of the randomly picked conditions for that neuron, and `t0` is the index (within the original PSTH with 130 time bins) of the time bin at -150ms, then running `x[t0:T] = 2 * x[t0] - x[t0:T]` will implement the desired inversion of the subsequent PSTH values about `x[t0]`.
   Q: Comment on what sort of statistical structure you expect this distortion to destroy, and what to preserve? (To address the second part, namely what is preserved, you can compare this distortion to, *e.g.*, an extremely drastic distortion where time bins are shuffled randomly, or a less extreme one where the conditions are shuffled independently for different neurons.)

   Rerun your code, *i.e.*, the computational steps from 2 to 5c,[17] after distorting the raw PSTHs (*i.e.*, the original 3D array X of shape $182 \times 108 \times 130$) as described.
   Q: How do the resulting plots differ from those originally obtained in 5c? What do you conclude?

[15] See the description of the optional `alpha` input in the help page for `plt.plot`.

[16] For picking half of the conditions at random, `numpy.random.choice(C, (C//2,), replace=False)` will return the indices of the randomly picked conditions (here $C = 108$ is the number of conditions). And you have to run this anew for each neuron.

[17] You are not asked to repeat intermediate plottings up until the final ones in 5c, just redo the computations needed to calculate the projected trajectories in the (new) FR plane.