

```

1  ###
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from matplotlib import style
5  style.use('classic')
6  from sklearn.decomposition import PCA
7  from cond_color import get_colors, plot_start,
   plot_end
8  ###
9  np.random.seed(41)
10 ### md
11 Loading Data
12 ###
13 data_orig = np.load('psth.npz')
14 X, times = data_orig['X'], data_orig['times']
15 ### md
16 # Exercise 1: Plotting raw PSTHs
17 ###
18 def plot_psths(data, timeintervals, n_rows, n_cols,
   n_cond):
19     np.random.seed(41)
20     cond = np.random.randint(low=data.shape[1], size=
   n_cond)
21
22     fig, ax = plt.subplots(n_rows, n_cols,
   tight_layout=True, figsize=(8,12))
23     for row in ax:
24         n = np.random.randint(low=data.shape[0], size
   =1)[0]
25         for c in cond:
26             row.plot(timeintervals, data[n, c,:],
   label = 'Condition = '+str(c))
27             row.set_title(f'Neuron {n}')
28             row.set_xlabel('Time relative to onset of
   hand movement')
29             row.set_ylabel('Trial-averaged spike rate
   (Hz)')
30             row.legend(loc='upper left', prop={'size'
   : 10})
31     return
32 ###

```

```

33 plot_psths(X, times, 3, 1, 4)
34 ###
35 pop_mean = data_orig['X'].mean(axis=(0,1))
36 plt.plot(times, pop_mean)
37 plt.title('Original Population')
38 plt.xlabel('Time relative to onset of hand movement (
    ms)')
39 plt.ylabel('Population-averaged firing rate (Hz)')
40
41 plt.show()
42 ### md
43 # Exercise 2: Pre-processing
44 ### md
45 ## Normalisation
46 ###
47 def plot_max_hist(data):
48     plt.hist(data.max(axis=(1,2)), bins=20)
49     plt.title('Histogram of neuron maximum values')
50     plt.ylabel('Frequency')
51     plt.xlabel('Max value for a given neuron across
    time and conditions')
52
53     plt.show()
54     return
55 ###
56 plot_max_hist(X)
57 ###
58 def norm_data(data):
59     a, b = data.max(axis=(1,2)), data.min(axis=(1,2))
60     a, b = a.reshape(data.shape[0],1,1), b.reshape(
    182,1,1)
61     data_normed = (data - b) / (a - b + 5 )
62
63     return data_normed
64 ###
65 X_norm = norm_data(X)
66 ###
67 plot_psths(X_norm, times, 2, 1, 4)
68 ### md
69 ## Mean centering
70 ###

```

```

71 def center_data(data):
72     mean = data.mean(axis=(0,2))
73     mean = mean.reshape(1, 108, 1)
74     data_centered = data - mean
75     return data_centered
76 ###
77 X_norm_mean = center_data(X_norm)
78 ###
79 plot_psths(X_norm_mean, times, 3, 1, 4)
80 ###
81 pop_mean = X_norm_mean.mean(axis=(0,1))
82 plt.plot(times, pop_mean)
83 plt.title('Pre-processed Population')
84 plt.xlabel('Time relative to onset of hand movement
            (ms)')
85 plt.ylabel('Population-averaged firing rate (Hz)')
86
87 plt.show()
88 ### md
89 ## Dimensionality reduction by PCA
90 ###
91 # times = times[65:111]
92 ###
93 X_trunc = X_norm_mean[:, :, 65:111]
94 X = X_trunc.reshape(X_trunc.shape[0], X_trunc.shape[
    1]*X_trunc.shape[2])
95 ###
96 pca = PCA(n_components=12)
97 pca.fit(X.T)
98 Z = pca.transform(X.T).T
99 ### md
100 # Plotting PC space trajectory
101 ###
102 Z = Z.reshape(12, X_trunc.shape[1], X_trunc.shape[2
    ])
103 ###
104 def plot_pca_psths(data1, pca_axis_1, pca_axis_2,
    data2=None, alpha=1):
105     np.random.seed(41)
106     pca_axis_1 -= 1
107     pca_axis_2 -= 1

```

```

108     alt_colors = [False, True]
109
110     for index, data in enumerate([data1, data2]):
111         if data is not None:
112             colors = get_colors(data[pca_axis_1, :,
113                                     0], data[pca_axis_2, :, 0], alt_colors[index//2])
114             for cond in range(0, data.shape[1]):
115                 plt.plot(data[pca_axis_1, cond, :],
116                         data[pca_axis_2, cond, :], label = 'C = '+str(cond
117                             ), color=colors[cond], alpha=alpha)
118                 plt.title(f'PCA plot')
119                 plt.xlabel(f'PC - {pca_axis_1 + 1}')
120                 plt.ylabel(f'PC - {pca_axis_2 + 1}')
121                 plot_start(data[pca_axis_1, :, 0], data[
122                     pca_axis_2, :, 0], colors, markersize=50, ax=None)
123                 plot_end(data[pca_axis_1, :, -1], data[
124                     pca_axis_2, :, -1], colors, markersize=50, ax=None)
125             else:
126                 pass
127         plt.show()
128     return
129
130 #%%
131 plot_pca_psths(Z, 1, 2, alpha=0.75)
132 #%%
133 plot_pca_psths(Z, 1, 3, alpha=0.75)
134 #%% md
135 # Exercise 4: Finding the max. likelihood estimate
136 for A
137 #%% md
138 ## Log-likelihood and its (naive) gradient
139 #%% md
140 
$$\Delta z_{t+1} = Az_t + \sigma \epsilon_t \text{ where } \sigma=1$$

141 #%% md
142 
$$\Delta z_{t+1} = N(Az_t, I_{12 \times 12})$$

143 #%% md
144 
$$P(\Delta z_{1:T-1} \mid z_{0:T-1}) = \prod_{t=0}^{T-1} N(Az_t, I_{12 \times 12})$$

145 #%% md
146 
$$\log(P(\Delta z_{1:T} \mid z_{0:T-1})) = \sum_{t=0}^{T-1} \log(N(Az_t, I_{12 \times 12}))$$


```

```

140 ### md
141  $\log(P(\Delta z_{1:T} | z_{0:T-1})) = \sum_{t=0}^T \log(\exp(-(\Delta z_{t+1} - Az_t)^T I_{12 \times 12} (\Delta z_{t+1} - Az_t))) + \text{const}$ 
142 ### md
143  $\log(P(\Delta z_{1:T} | z_{0:T-1})) = -\sum_{t=0}^T (\Delta z_{t+1} - Az_t)^T I_{12 \times 12} (\Delta z_{t+1} - Az_t) + \text{const}$ 
144 ### md
145  $\log(P(\Delta Z | Z)) \approx -(\Delta Z - AZ)^T (\Delta Z - AZ) \approx -Z^T A^T A Z + 2 \Delta Z^T A Z$ 
146 ### md
147  $\frac{d}{dA} (\log(P(\Delta Z | Z))) = -2 AZZ^T + 2 \Delta Z Z^T$ 
148 ### md
149 ## Parametrising an antisymmetric
150 ### md
151 K will equal 6, the number of matrix entries above the diagonal
152 ### md
153  $M \times M = 2K + M$ 
154 ### md
155  $K = \frac{M(M-1)}{2}$ 
156 ###
157 beta = np.array([[0.0001, 1, 1, 0.0001, 1, 1]])
158
159 def create_h(m):
160     k = int((m**2 - m)/2)
161     h = np.zeros((k, m, m))
162     row, column = 0, 1
163     for i in range(0, k):
164         h[i][int(row)][int(column)], h[i][int(column)
165         ][int(row)] = 1, -1
166         column += 1
167         if column >= m:
168             row += 1
169             column = row + 1
170     return h
171 H = create_h(4)
172 print(beta.shape)

```

```

173 print(H.shape)
174 print(H)
175
176 A = np.tensordot(beta, H, axes=1)
177 ### md
178 # Gradient with respect to  $\beta$ 
179 ### md
180  $\log(P(\Delta Z | Z)) \approx - (\sum_{a=1}^K \beta_a \sum_{j=1}^M H_{a,i,j} Z_{j,n})^T (\sum_{a=1}^K \beta_a \sum_{j=1}^M H_{a,i,j} Z_{j,n}) + 2 \Delta Z^T (\sum_{a=1}^K \beta_a \sum_{j=1}^M H_{a,i,j} Z_{j,n})$ 
181 ### md
182  $\log(P(\Delta Z | Z)) \approx - (\sum_{a=1}^K \beta_a W_{a,i,n})^T (\sum_{a=1}^K \beta_a W_{a,i,n}) + 2 \Delta Z^T (\sum_{a=1}^K \beta_a W_{a,i,n}) = - (\beta W)^T (\beta W) + 2 \Delta Z^T (\beta W)$ 
183 ### md
184  $\frac{d}{d\beta} (\log(P(\Delta Z | Z))) \approx - 2 \beta W^T W + 2 \Delta Z^T W$ 
185 ### md
186  $Q = W^T W$  and  $b = \Delta Z^T W$ 
187 ### md
188 ## An antisymmetric estimate for A
189 ### md
190 Solve  $\beta = b Q^{-1}$ 
191 ###
192 def a_estimate(z):
193     m = z.shape[0]
194     z_plus, z_ = z[:, :, 1:], z[:, :, :-1]
195     z_plus = z_plus.reshape((m, z_plus.shape[1]*
196                             z_plus.shape[2]))
197     z_ = z_.reshape((m, z_.shape[1]*z_.shape[2]))
198     h = create_h(z_.shape[0])
199     w = np.tensordot(h, z_, axes=1)
200     q = np.tensordot(w, w, axes=([1, 2], [1, 2]))
201
202     delta_z = z_plus - z_
203     b = np.tensordot(delta_z, w, axes=([0, 1], [1, 2]))
204     b = b.reshape((1, 66))
205

```

```

206     beta = b @ np.linalg.inv(q)
207     a = np.tensordot(beta, h, axes=([1],[0]))
208     a = a.reshape((m, m))
209     return beta, a
210 ###
211 beta, A = a_estimate(Z)
212 ###
213 img = plt.imshow(A, interpolation = 'nearest', cmap= '
    plasma' )
214 plt.colorbar(img)
215 plt.title('Colour plot of A')
216 plt.show()
217 ### md
218 ## Test
219 ###
220 test = np.load('test.npz')
221 Z_test = test['Z_test']
222 A_test = test['A_test']
223 ###
224 beta_test_estimated, A_test_estimated = a_estimate(
    Z_test)
225 ###
226 A_inaccuracy = (A_test_estimated - A_test)
227 print(A_inaccuracy.max())
228 ### md
229 # Exercise 5: 2D projections with rotational
    dynamics
230 ### md
231 ## A) Eigenvalues and Eigenvectors of A
232 ###
233 A_evalue, A_evector = np.linalg.eig(A)
234 print(A_evalue)
235 ### md
236 ## B)
237 ###
238 def get_p(eigen_vectors, plane):
239     evector_real = eigen_vectors[:,plane].real
240     evector_imag = eigen_vectors[:,plane].imag
241     p = np.zeros((2, 12))
242     p[0, :] = evector_real/np.linalg.norm(
        evector_real)

```

```

243     p[1, :] = evector_imag/np.linalg.norm(
        evector_imag)
244     print(p[0, :].T @ p[1, :])
245     return p
246 ###
247 P_FR = get_p(A_evector, 0)
248 ### md
249 ## C)
250 ###
251 def plot_proj_psths(data1, title, data2=None, alpha=
    1):
252     np.random.seed(41)
253     alt_colors = [False, True]
254
255     for index, data in enumerate([data1, data2]):
256         if data is not None:
257             colors = get_colors(data[0, :, 0], data[
1, :, 0], alt_colors[index//2])
258             for cond in range(0, data.shape[1]):
259                 plt.plot(data[0, cond, :], data[1,
cond, :], label = 'C = '+str(cond), color=colors[
cond], alpha=alpha)
260                 plt.title(title)
261                 plt.xlabel(f'Axis 1')
262                 plt.ylabel(f'Axis 2')
263                 plot_start(data[0, :, 0], data[1, :, 0
], colors, markersize=50, ax=None)
264                 plot_end(data[0, :, -1], data[1, :, -1
], colors, markersize=50, ax=None)
265             else:
266                 pass
267     plt.show()
268     return
269 ###
270 Projection_FR = np.tensordot(P_FR, Z, axes=([1],[0
]))
271 ###
272 plot_proj_psths(Projection_FR[:, :, 0:36], 'Z
    projected onto the FR plane', alpha=0.75)
273 ### md
274 ## D)

```



```

275 #%%
276 P_2 = get_p(A_evector, 3)
277 Projection_2 = np.tensordot(P_2, Z, axes=([1], [0]))
278 P_3 = get_p(A_evector, 5)
279 Projection_3 = np.tensordot(P_3, Z, axes=([1], [0]))
280 #%%
281 fig, ax = plt.subplots(2, 1, tight_layout=True,
    figsize=(10,15))
282 data1 = Projection_2[:, :, 0:36]
283 data2 = Projection_3[:, :, 0:36]
284 colors = get_colors(data1[0, :, 0], data1[1, :, 0])
285 for cond in range(0, data1.shape[1]):
286     ax[0].plot(data1[0, cond, :], data1[1, cond
    , :], label = 'C = '+str(cond), color=colors[cond],
    alpha=0.75)
287     ax[0].set_title('Z projected onto the 2nd plane'
    )
288     ax[0].set_xlabel(f'Axis 1')
289     ax[0].set_ylabel(f'Axis 2')
290 plot_start(data1[0, :, 0], data1[1, :, 0], colors,
    markersize=50, ax=ax[0])
291 plot_end(data1[0, :, -1], data1[1, :, -1], colors,
    markersize=50, ax=ax[0])
292
293 colors = get_colors(data2[0, :, 0], data2[1, :, 0])
294 for cond in range(0, data2.shape[1]):
295     ax[1].plot(data2[0, cond, :], data2[1, cond
    , :], label = 'C = '+str(cond), color=colors[cond],
    alpha=0.75)
296     ax[1].set_title('Z projected onto the 3rd plane'
    )
297     ax[1].set_xlabel(f'Axis 1')
298     ax[1].set_ylabel(f'Axis 2')
299 plot_start(data2[0, :, 0], data2[1, :, 0], colors,
    markersize=50, ax=ax[1])
300 plot_end(data2[0, :, -1], data2[1, :, -1], colors,
    markersize=50, ax=ax[1])
301 #%% md
302 # Exercise 6: Pre-movement period
303 #%%
304 X_pre = X_norm_mean[:, :, :66]

```

```

305 X = X_pre.reshape(X_pre.shape[0], X_pre.shape[1] *
    X_pre.shape[2])
306 Z_pre = pca.transform(X.T).T
307 Z_pre = Z_pre.reshape(12, X_pre.shape[1], X_pre.
    shape[2])
308 Projection_FR_pre = np.tensordot(P_FR, Z_pre, axes
    =([1], [0]))
309 ###
310 plt.figure(figsize=(10, 10))
311 data = Projection_FR
312 colors = get_colors(data[0, :, 0], data[1, :, 0],
    alt_colors=False)
313 for cond in range(0, data.shape[1]):
314     plt.plot(data[0, cond, :], data[1, cond, :],
        label = 'C = '+str(cond), color=colors[cond], alpha=
        0.5)
315     plot_start(data[0, :, 0], data[1, :, 0], colors
        , markersize=50, ax=None)
316     plot_end(data[0, :, -1], data[1, :, -1], colors
        , markersize=50, ax=None)
317
318 data = Projection_FR_pre
319 colors = get_colors(data[0, :, -1], data[1, :, -1],
    alt_colors=True)
320 for cond in range(0, data.shape[1]):
321     plt.plot(data[0, cond, :], data[1, cond, :],
        label = 'C = '+str(cond), color=colors[cond], alpha=
        0.5)
322     plot_start(data[0, :, 0], data[1, :, 0], colors
        , markersize=50, ax=None)
323     plot_end(data[0, :, -1], data[1, :, -1], colors
        , markersize=50, ax=None)
324
325 plt.title('Z (-800ms to 300ms) projected on the FR
    plane')
326 plt.xlabel(f'Axis 1')
327 plt.ylabel(f'Axis 2')
328
329 plt.show()
330 ### md
331 # Exercise 7: Control Analysis

```

```

332 #%%
333 ## Load data
334 X_seven, times = data_orig['X'], data_orig['times']
335 #%%
336 ## Plot pre-distortion
337 plot_psths(X_seven, times, 2, 1, 4)
338 #%%
339 ## Distortion
340 for N in range(X_seven.shape[0]):
341     conditions = np.random.choice(108, (108//2,),
342     replace=False)
343     X_seven[N, conditions, 65:] = 2*X_seven[N,
344     conditions, 65].reshape((54,1))- X_seven[N,
345     conditions, 65:]
346 #%%
347 ## Normalising
348 X_seven_norm = norm_data(X_seven)
349 #%%
350 ## Mean centering
351 X_seven_norm_mean = center_data(X_seven_norm)
352 plot_psths(X_seven_norm_mean, times, 3, 1, 4)
353 #%%
354 ## Dimensionality reduction by PCA
355 X_seven_trunc = X_seven_norm_mean[:, :, 65:111]
356 X_seven = X_seven_trunc.reshape(X_seven_trunc.shape[
357     0], X_seven_trunc.shape[1] * X_seven_trunc.shape[2])
358 pca_seven = PCA(n_components=12)
359 pca_seven.fit(X_seven.T)
360 Z = pca_seven.transform(X_seven.T).T
361 #%%
362 Z = Z.reshape(12, X_trunc.shape[1], X_trunc.shape[2
363     ])
364 plot_pca_psths(Z, 1, 2)
365 #%%
366 # Exercise 4
367 beta, A = a_estimate(Z)
368 img = plt.imshow(A, interpolation = 'nearest', cmap='
369     plasma' )
370 plt.colorbar(img)
371 plt.title('Colour plot of A')
372 plt.show()

```

```
367 #%%
368 ## A)
369 A_evalue, A_evector = np.linalg.eig(A)
370
371 ## B)
372 P_FR = get_p(A_evector, 0)
373
374 ## C)
375 Projection_FR = np.tensordot(P_FR, Z, axes=([1], [0
    ]))
376 plot_proj_psths(Projection_FR[:, :, 0:36], '
    Projection of disorted Z onto the FR plane', alpha=0
    .75)
377 #%%
378
```