

```

1 #####
2 # IMPORTS
3 #####
4 import numpy as np
5 import os
6 import matplotlib.pyplot as plt
7 import matplotlib as mpl
8 from kalman import kalman_filter
9 #####
10 # Variable declaration
11 #####
12 isPlotReqd = True
13 isPlotPdf = True
14 #####
15 # Settings for plot
16 #####
17 if (True == isPlotReqd):
18     if (True == isPlotPdf):
19         mpl.use('pdf')
20         fig_width = 3.487
21         fig_height = fig_width / 1.618
22         rcParams = {
23             'font.family': 'serif',
24             'font.serif': 'Times New Roman',
25             'text.usetex': True,
26             'xtick.labelsize': 8,
27             'ytick.labelsize': 8,
28             'axes.labelsize': 8,
29             'legend.fontsize': 8,
30             'figure.figsize': [fig_width, fig_height]
31         }
32         plt.rcParams.update(rcParams)
33 #####
34 # CODE STARTS HERE
35 #####
36 #####
37 # Parameters
38 #####
39 x_0 = np.array([[650.], [250.]])
40 x_0_hat = np.array([[600.], [200.]])
41 P_0_hat = np.array([[500., 0.], [0., 200.]])
42 Q = np.array([[0., 0.], [0., 10.]])
43 H = np.array([[1, 0]])
44 R = 10.
45 F = np.array([[0.5, 2.], [0, 1]])
46 G = np.zeros(x_0_hat.shape)
47 #####
48 # Time steps and process noise
49 #####
50 num_step = 10
51 n_historical_data = 1000
52 t = np.linspace(0, n_historical_data, num=n_historical_data+1)
53 w_f = np.zeros(t.shape)
54 w_f[1:] = np.sqrt(10.) * np.random.normal(0, 1, (t.shape[0]-1,))
55 w_k = np.vstack((np.zeros(t.shape), w_f))
56 #####
57 # True population and generate measurements
58 #####
59 x_true = np.zeros((x_0.shape[0], t.shape[0]))
60 x_true[:, 0] = x_0[:, 0]
61 y_k = np.zeros((1, t.shape[0]))

```

```

62 # We will generate history for 1000 steps
63 for step in range(n_historical_data):
64     x_true[:, step+1] = F @ x_true[:, step] + w_k[:, step]
65     y_k[0, step+1] = H @ x_true[:, step+1] + np.sqrt(R) * np.random.normal()
66
67 #####
68 # Kalman filter prediction
69 #####
70 kf = kalman_filter(init_state=x_0_hat, init_est_err=P_0_hat, Q=Q, R=R, F=F, G=G, H=H, y_k=y_k,
71                   num_step=num_step)
72 kf.run_kalman()
73 P_k = kf.get_est_error_cov()
74 K_k = kf.get_kalman_gains()
75 x_k_0 = kf.get_predicted_state(0)
76 x_k_1 = kf.get_predicted_state(1)
77 #####
78 # Plot of true and estimated population
79 #####
80 if (True == isPlotReqd):
81     #####
82     # Configure axis and grid
83     #####
84     fig = plt.figure()
85     ax = fig.add_subplot(111)
86     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
87
88     ax.set_axisbelow(True)
89     ax.minorticks_on()
90     ax.grid(which='major', linestyle='-', linewidth='0.5')
91     ax.grid(which='minor', linestyle='-', linewidth='0.5')
92     ax.plot(t[0:num_step+1], x_true[0, 0:num_step+1], label='True population')
93     ax.plot(t[0:num_step+1], x_k_0, label='estimated population')
94
95     ax.set_xlabel(r'time', fontsize=8)
96     ax.set_ylabel(r'population', fontsize=8)
97
98     plt.legend()
99     if (True == isPlotPdf):
100         if not os.path.exists('./generatedPlots'):
101             os.makedirs('generatedPlots')
102         fig.savefig('./generatedPlots/q4_population.pdf')
103     else:
104         plt.show()
105 #####
106 # Plot of true and estimated food supply
107 #####
108 if (True == isPlotReqd):
109     #####
110     # Configure axis and grid
111     #####
112     fig = plt.figure()
113     ax = fig.add_subplot(111)
114     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
115
116     ax.set_axisbelow(True)
117     ax.minorticks_on()
118     ax.grid(which='major', linestyle='-', linewidth='0.5')
119     ax.grid(which='minor', linestyle='-', linewidth='0.5')
120     ax.plot(t[0:num_step+1], x_true[1, 0:num_step+1], label='True food supply')
121     ax.plot(t[0:num_step+1], x_k_1, label='estimated food supply')
122

```

```

122 ax.set_xlabel(r'time', fontsize=8)
123 ax.set_ylabel(r'food supply', fontsize=8)
124
125 plt.legend()
126 if (True == isPlotPdf):
127     if not os.path.exists('./generatedPlots'):
128         os.makedirs('generatedPlots')
129     fig.savefig('./generatedPlots/q4_food.pdf')
130 else:
131     plt.show()
132 #####
133 # Plot of std dev of estimation error
134 #####
135 if (True == isPlotReqd):
136     #####
137     # Configure axis and grid
138     #####
139     fig = plt.figure()
140     ax = fig.add_subplot(111)
141     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
142
143     ax.set_axisbelow(True)
144     ax.minorticks_on()
145     ax.grid(which='major', linestyle='-', linewidth='0.5')
146     ax.grid(which='minor', linestyle='-', linewidth='0.5')
147
148     ax.plot(t[0:num_step+1], np.sqrt(P_k[0, 0, :]), color='g', label='Population estimation error std')
149     ax.plot(t[0:num_step+1], -np.sqrt(P_k[0, 0, :]), color='g')
150     ax.plot(t[0:num_step+1], np.sqrt(P_k[1, 1, :]), color='b', label='food estimation error std')
151     ax.plot(t[0:num_step+1], -np.sqrt(P_k[1, 1, :]), color='b')
152
153     ax.set_xlabel(r'time', fontsize=8)
154     ax.set_ylabel(r'standard deviation', fontsize=8)
155
156     plt.legend()
157     if (True == isPlotPdf):
158         if not os.path.exists('./generatedPlots'):
159             os.makedirs('generatedPlots')
160         fig.savefig('./generatedPlots/q4_std_dev.pdf')
161     else:
162         plt.show()
163 #####
164 # Plot of Kalman gains
165 #####
166 if (True == isPlotReqd):
167     #####
168     # Configure axis and grid
169     #####
170     fig = plt.figure()
171     ax = fig.add_subplot(111)
172     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
173
174     ax.set_axisbelow(True)
175     ax.minorticks_on()
176     ax.grid(which='major', linestyle='-', linewidth='0.5')
177     ax.grid(which='minor', linestyle='-', linewidth='0.5')
178
179     ax.plot(t[0:num_step+1], K_k[0, :], label='Kalman gain for Population')
180     ax.plot(t[0:num_step+1], K_k[1, :], label='Kalman gain for food')
181
182     ax.set_xlabel(r'time', fontsize=8)

```

```

183 ax.set_ylabel(r'Kalman gains', fontsize=8)
184
185 plt.legend()
186 if (True == isPlotPdf):
187     if not os.path.exists('./generatedPlots'):
188         os.makedirs('generatedPlots')
189         fig.savefig('./generatedPlots/q4_kal_gain.pdf')
190 else:
191     plt.show()
192 #####
193 # (b) Theoretical estimation standard dev from covariance analysis
194 #####
195 # cov_ana_x_0_hat = None
196 n_steps_steady_state = 1000 # Just to calculate theoretical steady state value
197 cov_ana_P_0_hat = np.zeros(P_0_hat.shape)
198 P_k_theoretical = kf.calculate_theoretical_cov(x_0=None, P_0=cov_ana_P_0_hat, num_step=
n_steps_steady_state)
199 #####
200 # Plot of std dev of estimation error and theoretical value
201 #####
202 if (True == isPlotReqd):
203     #####
204     # Configure axis and grid
205     #####
206     fig = plt.figure()
207     ax = fig.add_subplot(111)
208     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
209
210     ax.set_axisbelow(True)
211     ax.minorticks_on()
212     ax.grid(which='major', linestyle='-', linewidth='0.5')
213     ax.grid(which='minor', linestyle="-.", linewidth='0.5')
214
215     ax.plot(t[0:num_step+1], np.sqrt(P_k[0, 0, :]), color='g', label='Population est s.d.')
216     ax.plot(t[0:num_step+1], -np.sqrt(P_k[0, 0, :]), color='g')
217     ax.hlines(np.sqrt(P_k_theoretical[0, 0, -1]), t[0], t[num_step],
218             color='r', label='Population theoretical ss s.d.')
219     ax.hlines(-np.sqrt(P_k_theoretical[0, 0, -1]), t[0], t[num_step], color='r')
220     ax.plot(t[0:num_step+1], np.sqrt(P_k[1, 1, :]), color='b', label='food est s.d.')
221     ax.plot(t[0:num_step+1], -np.sqrt(P_k[1, 1, :]), color='b')
222     ax.hlines(np.sqrt(P_k_theoretical[1, 1, -1]), t[0], t[num_step],
223             color='m', label='food theoretical ss s.d.')
224     ax.hlines(-np.sqrt(P_k_theoretical[1, 1, -1]), t[0], t[num_step], color='m')
225
226     ax.set_xlabel(r'time', fontsize=8)
227     ax.set_ylabel(r'standard deviation', fontsize=8)
228
229     plt.legend()
230     if (True == isPlotPdf):
231         if not os.path.exists('./generatedPlots'):
232             os.makedirs('generatedPlots')
233             fig.savefig('./generatedPlots/q4_std_dev_theoretical.pdf')
234     else:
235         plt.show()
236     #####
237 # (c) Theoretical estimation standard dev from covariance analysis
238 #####
239 kf.num_step = 1000
240 kf.run_kalman()
241 P_k = kf.get_est_error_cov()
242 P_k_theoretical = kf.calculate_theoretical_cov(x_0=None, P_0=cov_ana_P_0_hat)

```

```

243 # t = np.linspace(0, kf.num_step, num=kf.num_step+1)
244 #####
245 # Plot of std dev of estimation error and theoretical value
246 #####
247 if (True == isPlotReqd):
248     #####
249     # Configure axis and grid
250     #####
251     fig = plt.figure()
252     ax = fig.add_subplot(111)
253     fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
254
255     ax.set_axisbelow(True)
256     ax.minorticks_on()
257     ax.grid(which='major', linestyle='-', linewidth='0.5')
258     ax.grid(which='minor', linestyle='-.', linewidth='0.5')
259
260     ax.plot(t, np.sqrt(P_k[0, 0, :]), color='g', label='Population est s.d.')
261     ax.plot(t, -np.sqrt(P_k[0, 0, :]), color='g')
262     ax.hlines(np.sqrt(P_k_theoretical[0, 0, -1]), t[0], t[-1],
263              color='r', label='Population theoretical ss s.d.')
264     ax.hlines(-np.sqrt(P_k_theoretical[0, 0, -1]), t[0], t[-1], color='r')
265     ax.plot(t, np.sqrt(P_k[1, 1, :]), color='b', label='food est s.d.')
266     ax.plot(t, -np.sqrt(P_k[1, 1, :]), color='b')
267     ax.hlines(np.sqrt(P_k_theoretical[1, 1, -1]), t[0], t[-1],
268              color='m', label='Food theoretical ss s.d.')
269     ax.hlines(-np.sqrt(P_k_theoretical[1, 1, -1]), t[0], t[-1], color='m')
270
271     ax.set_xlabel(r'time', fontsize=8)
272     ax.set_ylabel(r'standard deviation', fontsize=8)
273
274     plt.legend()
275     if (True == isPlotPdf):
276         if not os.path.exists('./generatedPlots'):
277             os.makedirs('generatedPlots')
278         fig.savefig('./generatedPlots/q4_std_dev_theoretical_1000.pdf')
279     else:
280         plt.show()
281

```