```python
################################################################################
# IMPORTS
################################################################################
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib as mpl
from kalman import kalman_filter
##############################################################################
# Variable declaration
##############################################################################
isPlotReqd = True
isPlotPdf = True
##############################################################################
# Settings for plot
##############################################################################
if (True == isPlotReqd):
    if (True == isPlotPdf):
        mpl.use('pdf')
        fig_width  = 3.487
        fig_height = fig_width / 1.618
        rcParams = {
            'font.family': 'serif',
            'font.serif': 'Times New Roman',
            'text.usetex': True,
            'xtick.labelsize': 8,
            'ytick.labelsize': 8,
            'axes.labelsize': 8,
            'legend.fontsize': 8,
            'figure.figsize': [fig_width, fig_height]
           }
        plt.rcParams.update(rcParams)
##############################################################################
# CODE STARTS HERE
##############################################################################
##############################################################################
# Parameters
##############################################################################
x_0 = np.array([[650.], [250.]])
x_0_hat = np.array([[600.], [200.]])
P_0_hat = np.array([[500., 0.], [0., 200.]])
Q = np.array([[0., 0.], [0., 10.]])
H = np.array([[1, 0]])
R = 10.
F = np.array([[0.5, 2],[0, 1]])
G = np.zeros(x_0_hat.shape)
##############################################################################
# Time steps
##############################################################################
num_step = 10
t = np.linspace(0, num_step, num=num_step + 1)
##############################################################################
# Dynamic system simulation
##############################################################################
class dynamic_system():
    def __init__(self, num_step, x_0, F, G, H, Q, R):
        self.num_step = num_step
        self.t = np.linspace(0, num_step, num=num_step + 1)
        self.x_0 = x_0
        self.F = F
        self.G = G
```

```python
        self.H = H
        self.Q = Q
        self.R = R # Right not supporting scalar R
    def get_simulated_measurements(self):
        w_f = np.zeros(self.t.shape)
        w_f[1:] = np.sqrt(self.Q[1, 1]) * np.random.normal(0, 1, (self.t.shape[0] - 1,))
        w_k = np.vstack((np.zeros(self.t.shape), w_f))

        x_true = np.zeros((self.x_0.shape[0], self.t.shape[0]))
        x_true[:, 0] = self.x_0[:, 0]
        y_k = np.zeros((1, self.t.shape[0]))
        for step in range(self.num_step):
            x_true[:, step + 1] = self.F @ x_true[:, step] + w_k[:, step]
            y_k[0, step + 1] = self.H @ x_true[:, step + 1] + np.sqrt(self.R) * np.random.normal()
        return y_k
################################################################################
# Kalman filter prediction
################################################################################
ds = dynamic_system(num_step=num_step, x_0=x_0, F=F, G=G, H=H, Q=Q, R=R)
y_k = ds.get_simulated_measurements()
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, num_step=num_step)
kf.run_kalman()
kf.run_fb_smoother(num_meas=10, est_idx=-1)
P_f = kf.get_est_error_cov()
P_fb = kf.get_est_error_cov_fb_sm()
#######################################################################
# Plot of cov of estimation error: population
#######################################################################
if (True == isPlotReqd):
    ###########################################################################
    # Configure axis and grid
    ###########################################################################
    fig = plt.figure()
    ax = fig.add_subplot(111)
    fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)

    ax.set_axisbelow(True)
    ax.minorticks_on()
    ax.grid(which='major', linestyle='-', linewidth='0.5')
    ax.grid(which='minor', linestyle='-.', linewidth='0.5')

    ax.plot(t, P_f[0, 0, :], color='r', label='Population error cov: forward')
    ax.plot(t, P_fb[0, 0, :], color='g', label='Population error cov: smoothed')

    ax.set_xlabel(r'time', fontsize=8)
    ax.set_ylabel(r'standard deviation', fontsize=8)

    plt.legend()
    if (True == isPlotPdf):
        if not os.path.exists('./generatedPlots'):
            os.makedirs('generatedPlots')
        fig.savefig('./generatedPlots/q5_population_cov.pdf')
    else:
        plt.show()
#######################################################################
# Plot of cov of estimation error: food
#######################################################################
if (True == isPlotReqd):
    ###########################################################################
    # Configure axis and grid
```

```python
    ###############################################################################
    fig = plt.figure()
    ax = fig.add_subplot(111)
    fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)

    ax.set_axisbelow(True)
    ax.minorticks_on()
    ax.grid(which='major', linestyle='-', linewidth='0.5')
    ax.grid(which='minor', linestyle='-.', linewidth='0.5')

    ax.plot(t, P_f[1, 1, :], color='b', label='food error cov: forward')
    ax.plot(t, P_fb[1, 1, :], color='m', label='food error cov: smoothed')

    ax.set_xlabel(r'time', fontsize=8)
    ax.set_ylabel(r'standard deviation', fontsize=8)

    plt.legend()
    if (True == isPlotPdf):
        if not os.path.exists('./generatedPlots'):
            os.makedirs('generatedPlots')
        fig.savefig('./generatedPlots/q5_food_cov.pdf')
    else:
        plt.show()
###############################################################################
# (b)Percentage change of cov estimates at initial time due to smoothing
###############################################################################
improvement_popu = ((P_f[0, 0, 0] - P_fb[0, 0, 0])/P_fb[0, 0, 0])*100.
improvement_food = ((P_f[1, 1, 0] - P_fb[1, 1, 0])/P_fb[1, 1, 0])*100.
print('Percentage change of initial estimation error variance due to smoothing: population: ',
improvement_popu)
print('Percentage change of initial estimation error variance due to smoothing: food: ',
improvement_food)
###############################################################################
# (c) Numerical estimate of error cov at initial time
###############################################################################
n_run = 100
init_cov_est_popu = np.zeros((n_run,))
init_cov_est_food = np.zeros((n_run,))
for run_idx in range(n_run):
    y_k = ds.get_simulated_measurements()
    kf.y_k = y_k
    kf.run_fb_smoother(num_meas=10, est_idx=-1)
    P_fb_est = kf.get_est_error_cov_fb_sm()
    init_cov_est_popu[run_idx] = P_fb_est[0, 0, 0]
    init_cov_est_food[run_idx] = P_fb_est[1, 1, 0]
print('Numerical estimate of initial population error cov = ', np.mean(init_cov_est_popu))
print('Theoretical estimate of initial population error cov = ', P_fb[0, 0, 0])
print('Numerical estimate of initial food supply error cov = ', np.mean(init_cov_est_food))
print('Theoretical estimate of initial food supply error cov = ', P_fb[1, 1, 0])
```