```python
import numpy as np
from numpy.linalg import inv


def wrapper(ang):
    ang -= np.pi * 2 * np.floor((ang + np.pi) / (2 * np.pi))
    return ang


class EIF:
    def __init__(self, Quad, World):
        self.ts = Quad.ts
        self.mu = np.array([[Quad.x0],
                            [Quad.y0],
                            [Quad.theta0]])
        self.SIG = np.diag([1, 1, 1])
        self.landmarks = World.Landmarks
        self.num_landmarks = World.Number_Landmarks

        # given with each function call
        self.Omega = inv(self.SIG)
        self.xi = self.Omega @ self.mu
        v = Quad.v_t
        theta = 0
        # also r and ph, both vectors containing measurements

        # needed for lines 2-7
        self.G = self.calc_G(theta, v)
        self.V = self.calc_V(theta)
        self.M = np.diag([Quad.sig_v ** 2, Quad.sig_omega ** 2])
        self.Q = np.array([[Quad.sigma_r ** 2, 0],
                           [0, Quad.sigma_theta ** 2]])
        self.Q_inv = inv(self.Q)

    def calc_G(self, theta, v):
        G = np.array([[1, 0, - v * np.sin(theta) * self.ts],
                      [0, 1, v * np.cos(theta) * self.ts],
                      [0, 0, 1]])
        return G

    def calc_V(self, theta):
        V = np.array([[np.cos(theta) * self.ts, 0],
                      [np.sin(theta) * self.ts, 0],
                      [0., self.ts]])
        return V

    def calc_g(self, v, theta, omega):
        mat = np.array([[v * np.cos(theta) * self.ts],
                        [v * np.sin(theta) * self.ts],
                        [omega * self.ts]])
        mu_bar = self.mu + mat
        return mu_bar

    def update(self, xi, Omega, v, omega, r, ph):
        self.xi = xi
        self.Omega = Omega
        self.SIG = inv(self.Omega)
        self.mu = self.SIG @ self.xi
        theta = self.mu[2][0]
        self.G = self.calc_G(theta, v)
        self.V = self.calc_V(theta)
        # lines 2-5
```

```
        self.SIG_bar = self.G @ self.SIG @ self.G.T + self.V @ self.M @ self.V.T
        self.Omega_bar = inv(self.SIG_bar)
        self.mu_bar = self.calc_g(v, theta, omega)
        self.xi_bar = self.Omega_bar @ self.mu_bar

        self.features(r, ph)

        self.Omega = self.Omega_bar
        self.xi = self.xi_bar

    def features(self, r, ph):
        for spot in range(0, self.num_landmarks):
            diff_x = self.landmarks[0][spot] - self.mu_bar[0][0]
            diff_y = self.landmarks[1][spot] - self.mu_bar[1][0]
            q = diff_x ** 2 + diff_y ** 2
            z_hat = np.array([[np.sqrt(q)],
                              [wrapper(np.arctan2(diff_y, diff_x) - self.mu_bar[2]
[0])]])
            z_diff = np.array([r[spot] - z_hat[0],
                               wrapper(ph[spot] - z_hat[1])])
            H = np.array([[-diff_x / np.sqrt(q), -diff_y / np.sqrt(q), 0],
                          [diff_y / q, -diff_x / q, -1]])
            S = H @ self.SIG_bar @ H.T + self.Q
            K = self.SIG_bar @ H.T @ np.linalg.inv(S)
            self.mu_bar = self.mu_bar + K @ z_diff
            self.SIG_bar = (np.eye(3) - K @ H) @ self.SIG_bar
            self.Omega_bar = self.Omega_bar + H.T @ self.Q_inv @ H
            self.xi_bar = self.xi_bar + H.T @ self.Q_inv @ (z_diff + H @ self.mu_bar)
```

MeEn 595R – Autonomous Systems
Mid-term Exam

You are to design an extended information filter (EIF) to estimate the location of an autonomous quadrotor operating 30 m by 30 m space. Located in this space are six landmarks that are continuously visible to the quadrotor. The quadrotor can measure the range and bearing to each of these landmarks as it moves about. The two-dimensional dynamics of the flying quadrotor are described by the following equations

$$x_t = x_{t-1} + (v_t \cos\theta_{t-1})dt$$
$$y_t = y_{t-1} + (v_t \sin\theta_{t-1})dt$$
$$\theta_t = \theta_{t-1} + \omega_t dt.$$

The inputs to the quadrotor are commanded velocity v^c and commanded angular velocity ω^c. These commands are implemented on the quadrotor by low-level control loops operating on the quadrotor. The actual velocities that result can be modeled as

$$\begin{pmatrix} v_t \\ \omega_t \end{pmatrix} = \begin{pmatrix} v_t^c \\ \omega_t^c \end{pmatrix} + \begin{pmatrix} \varepsilon_v \\ \varepsilon_\omega \end{pmatrix},$$

where ε_v and ε_ω are normally distributed random variables with standard deviations given by $\sigma_v = 0.15$ m/s and $\sigma_\omega = 0.1$ rad/s respectively. You may assume that v and ω are uncorrelated (i.e., the motion noise covariance matrix has zeros for the off-diagonal covariance terms. This motion model is slightly different (and simpler) than the motion model we have used for the mobile robot.

The quadrotor has initial conditions x_0 = –5 m, y_0 = 0 m, θ_0 = 90 deg. You may assume that the altitude of the quadrotor is constant and unchanging.

.

The landmark locations are (6, 4), (–7, 8), (12, –8), (–2, 0), (–10, 2) and (13, 7). The standard deviation of the range and bearing sensor noise is given by $\sigma_r = 0.2$ m and $\sigma_\phi = 0.1$ rad respectively.

You will be provided with a data file midterm_data.mat that provides the time vector velocity commands, the actual velocities, the resulting pose states, as well as range and bearing measurements.

You are to implement an EIF to localize the quadrotor with a sample period of 0.1 s for a duration of 30 s. You can research information filters and extended information filters in section 3.5 of your text. Professor Stachniss also has a brief lecture introducing information filters that may be of interest: https://www.youtube.com/watch?v=wD6BGkGOW4A.

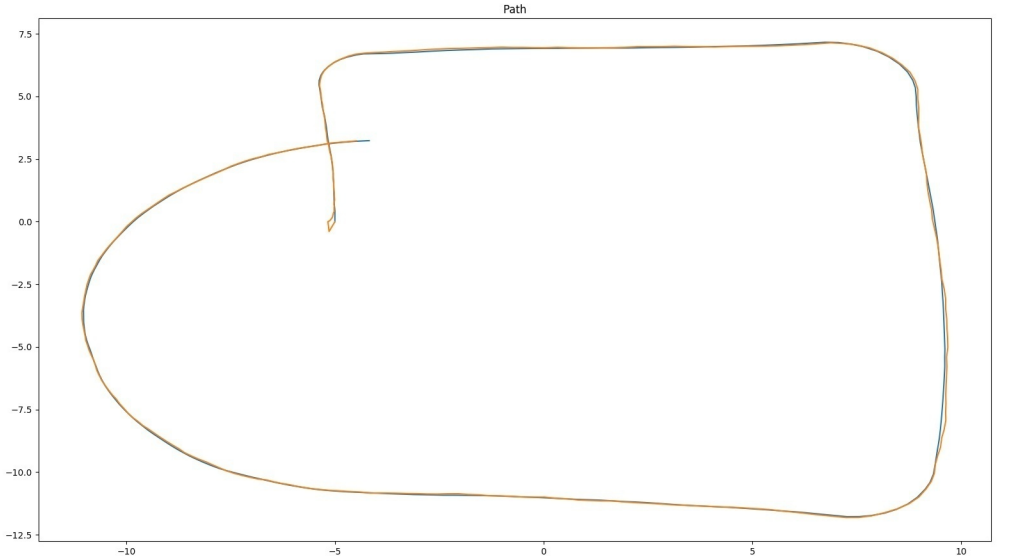
Tasks:

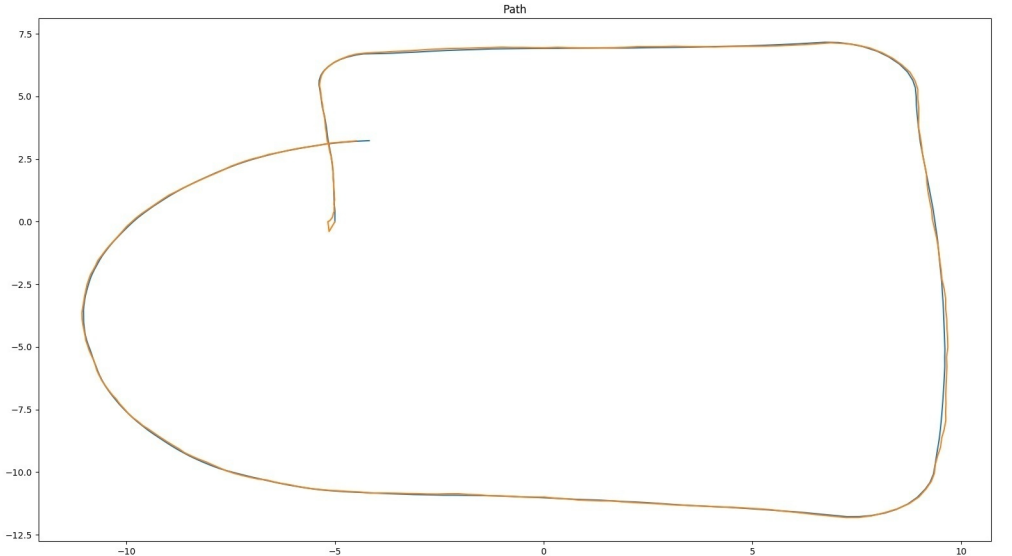
1. Calculate the necessary Jacobians to implement your EIF, specifically G_t, V_t, and H_t (M_t has been provided).
2. Implement your EIF localization algorithm. Create a plot of the information vector values versus time.
3. Create plots comparing your true states to the estimated states versus time. Plot estimation error and 95-percent uncertainty bounds (from your variances) versus time for each of your states.
4. Plot a 2-D map of the landmark locations, the true positions of the quadrotor, and the estimated position of the quadrotor.

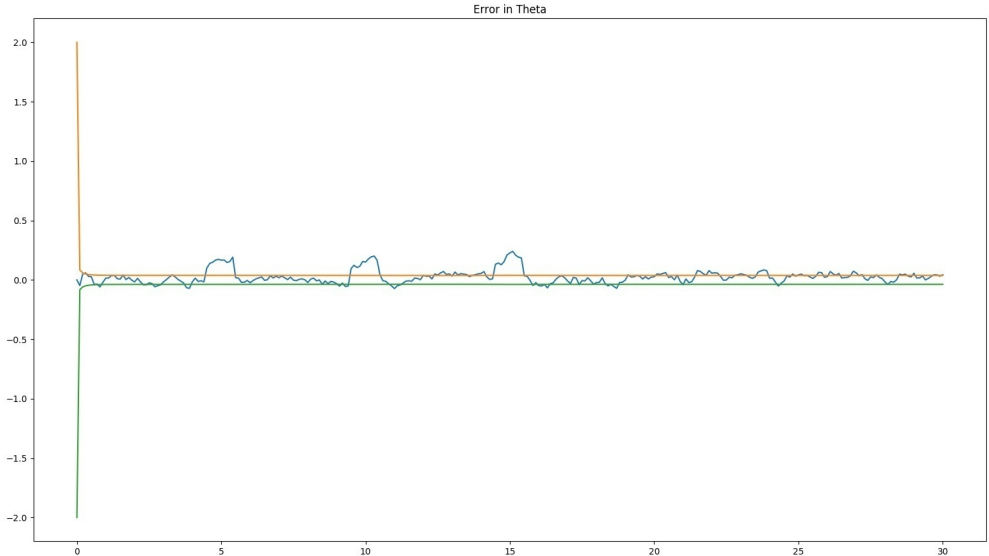
   Submit the above information as a single, well-organized PDF file within Learning Suite.

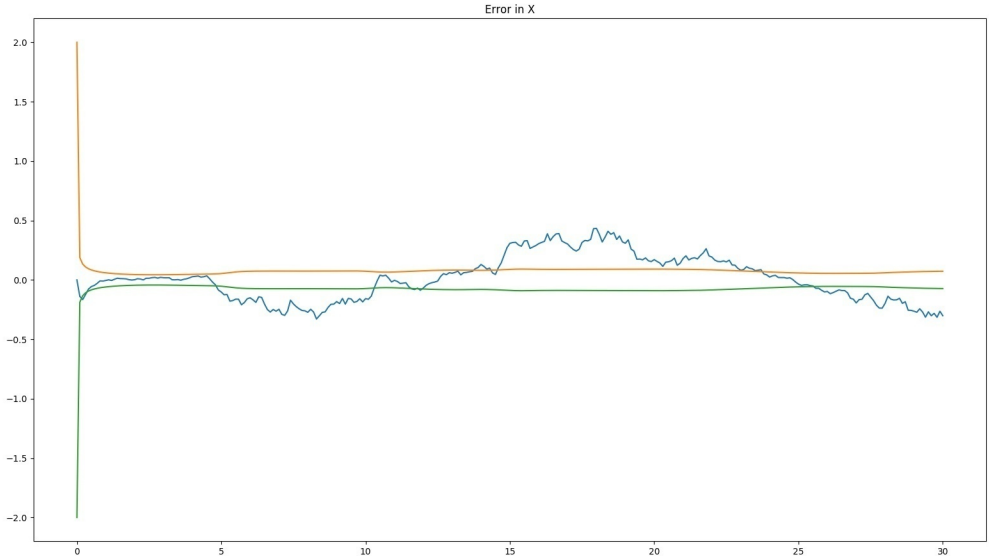
Hints:

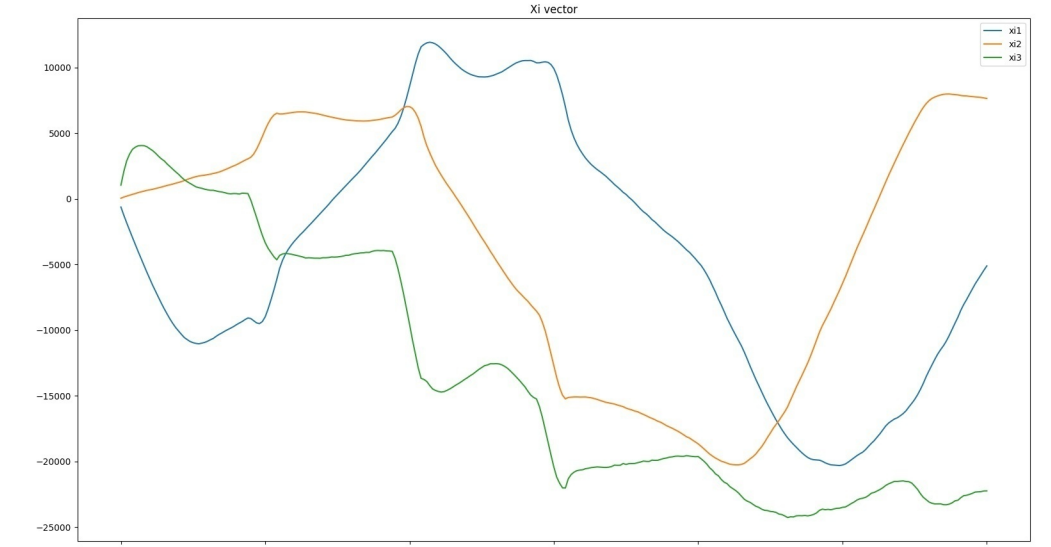
- As you perform calculations involving angles, you may need to normalize your angles to fall within $(-\pi, \pi]$.

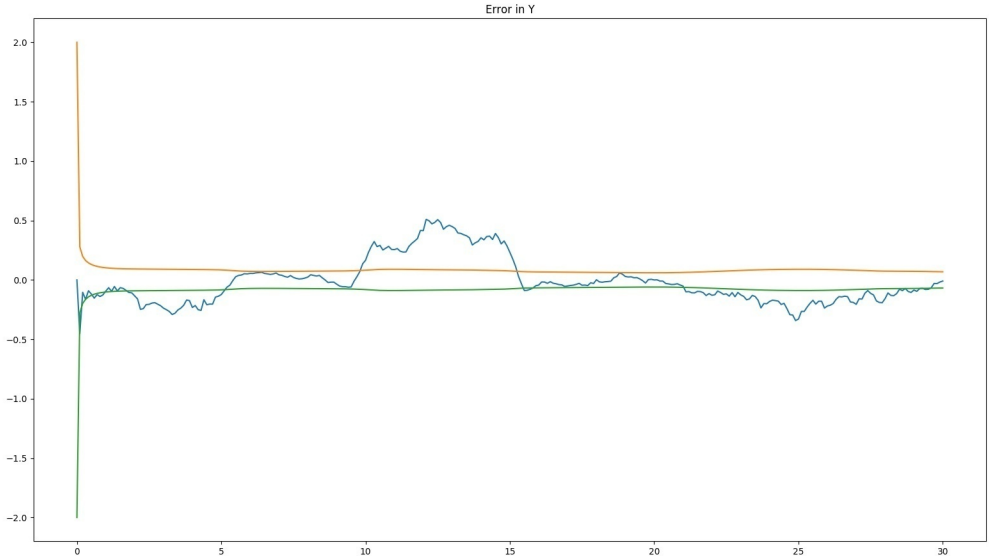

Path
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−10
−5
0
5
10


Path
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−10
−5
0
5
10


Error in Theta
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
30


Error in X
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
30


Xi vector
xi1
xi2
xi3
10000
5000
0
−5000
−10000
−15000
−20000
−25000
0
5
10
15
20
25
30


Error in Y
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
30

MeEn 595R – Autonomous Systems
Mid-term Exam

You are to design an extended information filter (EIF) to estimate the location of an autonomous quadrotor operating 30 m by 30 m space. Located in this space are six landmarks that are continuously visible to the quadrotor. The quadrotor can measure the range and bearing to each of these landmarks as it moves about. The two-dimensional dynamics of the flying quadrotor are described by the following equations

$$x_t = x_{t-1} + (v_t \cos\theta_{t-1})dt$$
$$y_t = y_{t-1} + (v_t \sin\theta_{t-1})dt$$
$$\theta_t = \theta_{t-1} + \omega_t dt.$$

The inputs to the quadrotor are commanded velocity v^c and commanded angular velocity ω^c. These commands are implemented on the quadrotor by low-level control loops operating on the quadrotor. The actual velocities that result can be modeled as

$$\begin{pmatrix} v_t \\ \omega_t \end{pmatrix} = \begin{pmatrix} v_t^c \\ \omega_t^c \end{pmatrix} + \begin{pmatrix} \varepsilon_v \\ \varepsilon_\omega \end{pmatrix},$$

where ε_v and ε_ω are normally distributed random variables with standard deviations given by $\sigma_v = 0.15$ m/s and $\sigma_\omega = 0.1$ rad/s respectively. You may assume that v and ω are uncorrelated (i.e., the motion noise covariance matrix has zeros for the off-diagonal covariance terms. This motion model is slightly different (and simpler) than the motion model we have used for the mobile robot.

The quadrotor has initial conditions x_0 = –5 m, y_0 = 0 m, θ_0 = 90 deg. You may assume that the altitude of the quadrotor is constant and unchanging.

.

The landmark locations are (6, 4), (–7, 8), (12, –8), (–2, 0), (–10, 2) and (13, 7). The standard deviation of the range and bearing sensor noise is given by $\sigma_r = 0.2$ m and $\sigma_\phi = 0.1$ rad respectively.

You will be provided with a data file midterm_data.mat that provides the time vector velocity commands, the actual velocities, the resulting pose states, as well as range and bearing measurements.

You are to implement an EIF to localize the quadrotor with a sample period of 0.1 s for a duration of 30 s. You can research information filters and extended information filters in section 3.5 of your text. Professor Stachniss also has a brief lecture introducing information filters that may be of interest: https://www.youtube.com/watch?v=wD6BGkGOW4A.

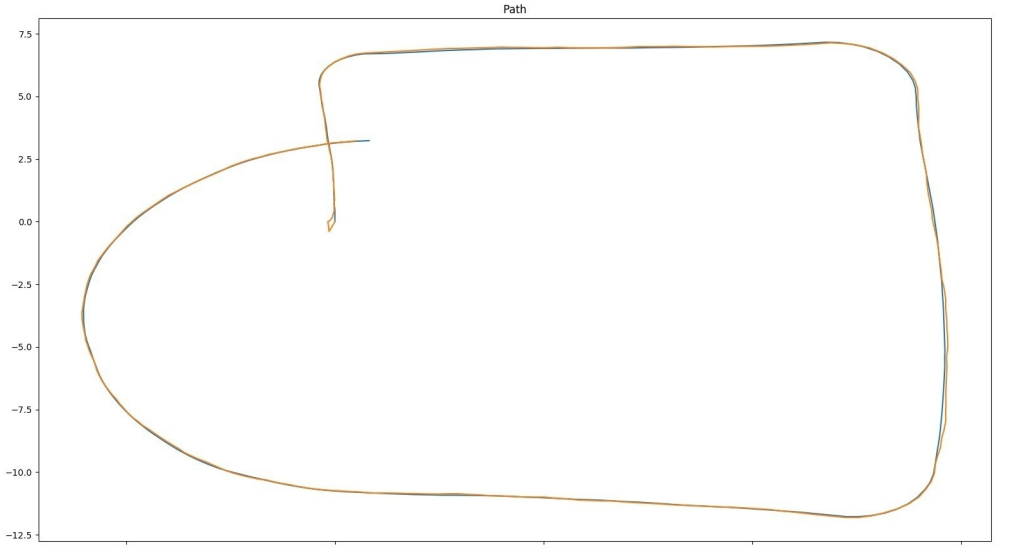
Tasks:

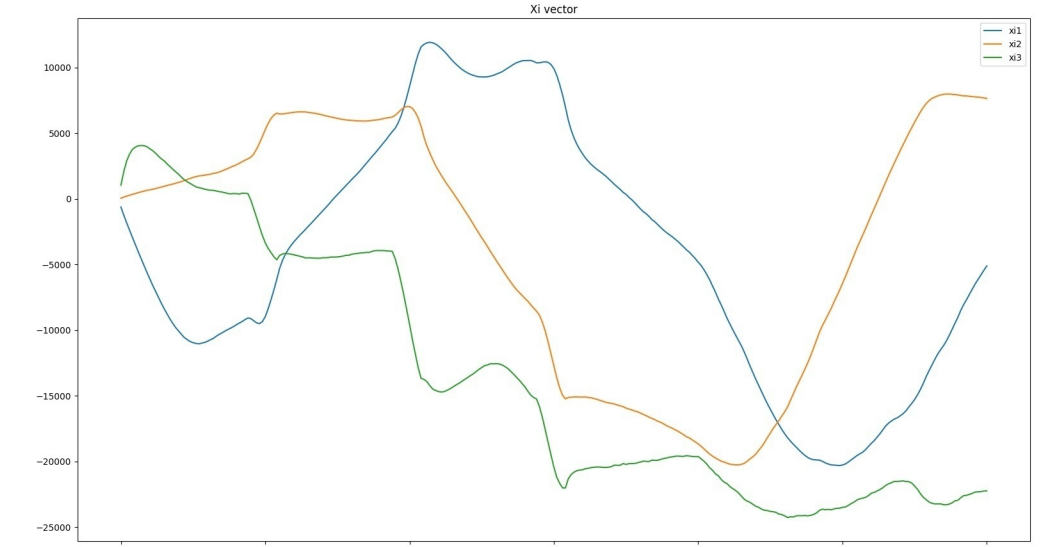
1. Calculate the necessary Jacobians to implement your EIF, specifically G_t, V_t, and H_t (M_t has been provided).
2. Implement your EIF localization algorithm. Create a plot of the information vector values versus time.
3. Create plots comparing your true states to the estimated states versus time. Plot estimation error and 95-percent uncertainty bounds (from your variances) versus time for each of your states.
4. Plot a 2-D map of the landmark locations, the true positions of the quadrotor, and the estimated position of the quadrotor.

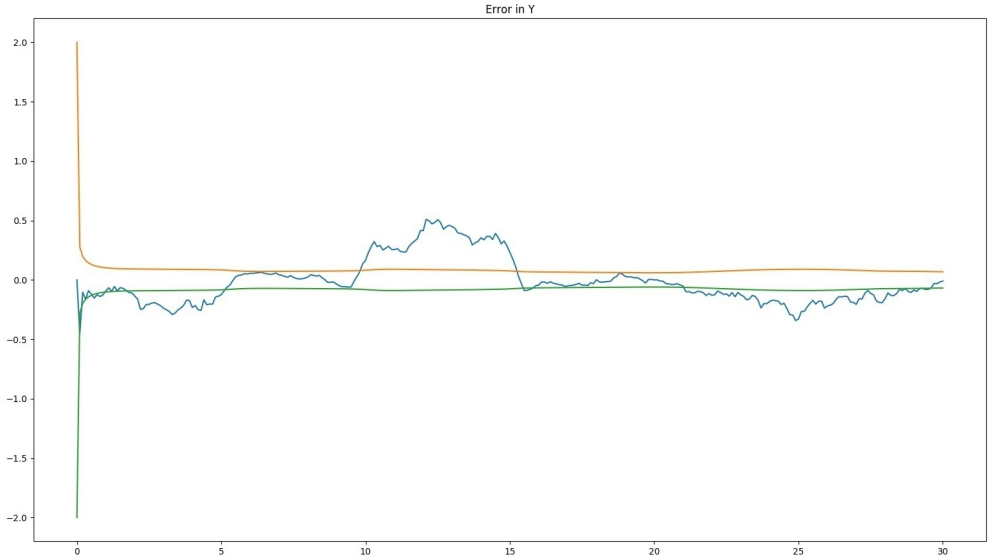
   Submit the above information as a single, well-organized PDF file within Learning Suite.

Hints:

- As you perform calculations involving angles, you may need to normalize your angles to fall within $(-\pi, \pi]$.


Path
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−10
−5
0
5
10


Xi vector
xi1
xi2
xi3
10000
5000
0
−5000
−10000
−15000
−20000
−25000
0
5
10
15
20
25
30


Error in Y
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
30