Mini Assignment 1
Kalman Filter

Anshul Mahajan

Objective

This assignment aims to implement a Kalman filter to track a target moving in a 2D space. We will apply the Kalman filter algorithm to predict and update the target state based on a dataset and evaluate its performance.

Introduction

Kalman Filter (KF) is a state estimation algorithm that can be used for tracking applications. The KF algorithm can be broadly divided into two stages - Prediction and Update.

Prediction Stage:

$$Eq\ 1:\ \hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

$$Eq\ 2:\ P_k^- = AP_{k-1}A^\top + Q$$

Update Stage:

$$Eq\ 3:\ K_k = \frac{P_k^- C^\top}{CP_k^- C^\top + R}$$

$$Eq\ 4:\ \hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

$$Eq\ 5:\ P_k = (I - K_k C)P_k^-$$

Approach

We begin by initializing a set of constants and variables that are crucial for the Kalman Filter implementation. Some of these values, such as the initial position, are derived directly from the provided dataset. Others, like the maximum velocity of the plane and the process noise covariance, are estimated based on common practices or assumptions about the system being modelled.

The heart of our approach lies in the iterative application of the Kalman Filter algorithm. For each time step in our dataset, we undertake the following steps:

State Prediction: We predict the current state of the system (position and velocity) based on the previous state estimate and the state transition matrix A. This matrix encapsulates our understanding of how the system's state evolves over time, assuming a constant velocity model in this case. We use Equation 1 for this.

Kalman Gain Calculation: We compute the Kalman Gain, denoted by K_k. This gain serves as a weighting factor that determines how much we should trust the new measurement versus our prediction from the previous step. The calculation of K_k (Equation 3) involves the current state covariance matrix P_k given by Equation 2, the measurement matrix H , and the measurement noise covariance matrix R_k.

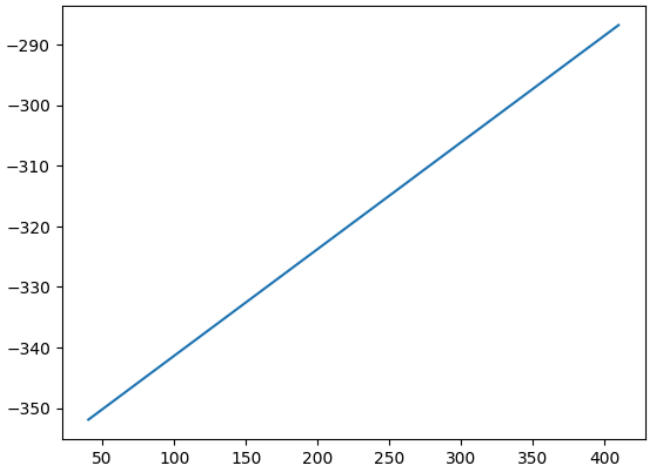
State Update: Using the calculated Kalman Gain, we update our state estimate by incorporating the new measurement from the dataset. This update step essentially blends our prediction with the observed data, giving more weight to the more reliable source of information. We use Equation 4 for this.

Covariance Update: We adjust the state covariance matrix P_k to reflect the uncertainty in our updated state estimate. This matrix captures how confident we are in our current knowledge of the system's state. We use Equation 5 for this.

This process is repeated for each time step, allowing us to track the estimated position of the object over time. Finally, we visually assess the performance of our Kalman Filter by plotting the predicted trajectory against the ground truth trajectory provided in the dataset. This comparison helps us gauge the accuracy and effectiveness of our filtering approach.

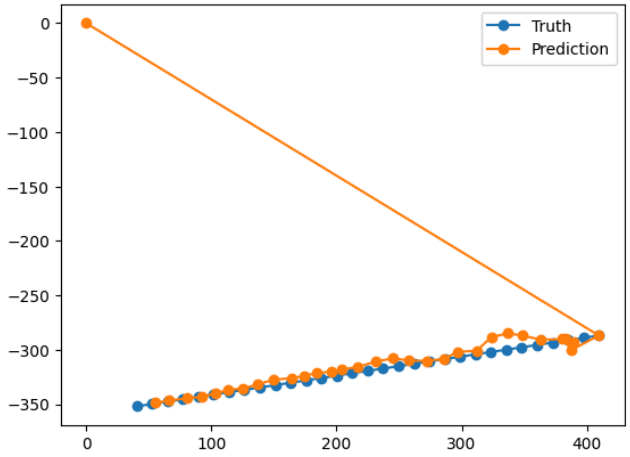
Visualisation

Ground Truth:



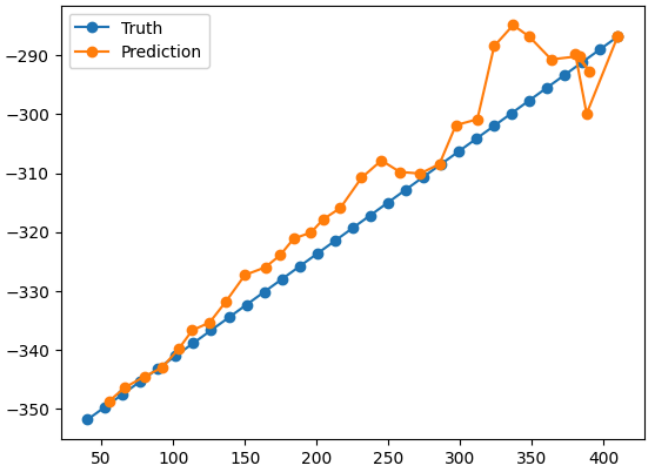
The particle seems to be travelling in a straight line from (387.72, -300.06) to (57.57, -357.99).

Tracking with initial state as (0, 0):



The KF does a great job of tracking the plane and quickly converges from a random state to the correct trajectory.
X MSE: 141.1776834722509
Y MSE: 50.525922677741654

Tracking with the initial ground truth as the initial state (409.58, -286.79):



The KF algorithm does a decent job here as well however it seems to overshoot and undershoot before converging to a semi-accurate trajectory. But the MSE is much larger than when it starts from a random state.
X MSE: 15610.498602888589
Y MSE: 593.055867252926

Challenges

1. Finding initial values of P and Q:

a. Although the values were covered in the tutorial session, I did not remember and needed to research and then ask friends from the course.

2. Loading data:
   a. It was initially hard to load the data with its already existing structure.

```
df = pd.read_csv('/content/drive/MyDrive/KF_Measurement.csv',
names = ['time', 'x', 'y', 'x_truth', 'y_truth'])
df = df.reset_index()
df = df.drop('y_truth', axis = 'columns')
df = df.rename(columns={"index": "time", "time": "x", "x" :
'y', "y" : 'x_truth', "x_truth" : 'y_truth'})
```

   b. I utilised a “hacky” workaround for this.
3. Matrix division:
   a. It did not strike me that the Kalman Gain equation uses division but in matrices that is akin to taking the inverse of one of the matrices. Normally dividing using / resulted in a typeError.
   b. I leveraged ChatGPT’s help in finding this out.

```
numerator = (current_P @ H.T)
denominator = (H @ current_P @ H.T + R)
K = numerator @ np.linalg.inv(denominator)
```

   c. The above solution was then found.

Appendix

Code:

```
df = pd.read_csv('/content/drive/MyDrive/KF_Measurement.csv', names =
['time', 'x', 'y', 'x_truth', 'y_truth'])
df = df.reset_index()
df = df.drop('y_truth', axis = 'columns')
df = df.rename(columns={"index": "time", "time": "x", "x" : 'y', "y" :
'x_truth', "x_truth" : 'y_truth'})
plt.plot(df['x_truth'], df['y_truth'])
plt.show()
# Constants and Variables

t = 1 # time step
std = 10 # standard deviation of measurement
var = std**2 # variance of measurement
v_max = 250 # max velocity of plane
Q_o = 0.1 # process noise covariance

# initial state
x = np.array([df.iloc[0][3:5].values[0], df.iloc[0][3:5].values[1], 0, 0])
```

```
# state transition matrix
A = np.array([[1, 0, t, 0],
              [0, 1, 0, t],
              [0, 0, 1, 0],
              [0, 0, 0, 1]])

#
P = np.array([[var, 0, 0, 0],
              [0, var, 0, 0],
              [0, 0, (v_max/3)**2, 0],
              [0, 0, 0, (v_max/3)**2]])

# process noise covariance
Q = Q_o * np.eye(4)

# measurement noise covariance
R = np.array([[var, 0],
              [0, var]])

# measurement matrix
H = np.array([[1, 0, 0, 0],
              [0, 1, 0, 0]])
x_estimate = [np.array([x_i, y_i])]

for dt in range(len(df)):
 # x(k) = A*x_k-1 + B*u(k)
 # Here u(k) = 0
 current_x = A @ x

 # Find Kalman Gain K:
 # 1. find P(k+1|k) = A * P(k|k)AT + Q*R
 current_P = A @ P @ A.T + Q

 # 2. find K = (P(k+1|k) * HT)/(H*P(k+1|k)*HT + R)
 numerator = (current_P @ H.T)
 denominator = (H @ current_P @ H.T + R)
 K = numerator @ np.linalg.inv(denominator)

 # 3. find x(k+1|k) = x(k+1|k-1) + K*(z(k) - H*x(k+1|k-1))
```

```
 z = df.iloc[dt][1:3].values # get x and y from dataset
 new_x = current_x + K @ (z - H @ current_x)

 # Update P = (I - K*H)*P_k-1
 P = (np.eye(4) - K @ H) @ current_P

 x_estimate.append(x)

 x = new_x

x_predict = [pt[0] for pt in x_estimate]
y_predict = [pt[1] for pt in x_estimate]
plt.plot(df['x_truth'], df['y_truth'], marker='o')
plt.plot(x_predict, y_predict, marker='o')
plt.legend(['Truth', 'Prediction'])
plt.show()
```