Skoltech

Skolkovo Institute of Science and Technology

Joint assimilation of navigation data coming from different sources

Experimental Data Processing

Assignment №10

Team №10

Submitted by:
Yunseok Park
Ilya Novikov
Ruslan Kalimullin

Instructor:
Tatiana Podladchikova

MA060238 (Term 1B, 2022-2023)

October 24th, 2022

Contents

1 Introduction

The objective of this laboratory work is to develop a navigation filter by assimilating data coming from different sources. Important outcome of this exercise is getting skill to incorporate all available measurement information into assimilation algorithm and develop a tracking filter for nonlinear models.

2 Work progress

Question 1 - 3

We generated true trajectory X_i of an object motion disturbed by normally distributed random acceleration:

$$\begin{aligned} x_i &= x_{i-1} + V^x_{i-1}T \\ V^x_i &= V^x_{i-1} + a^x_{i-1}T \\ y_i &= y_{i-1} + V^y_{i-1}T \\ V^y_i &= V^y_{i-1} + a^y_{i-1}T \end{aligned} \tag{1}$$

Following initial conditions are given to generate true trajectory:

1. size of trajectory $N = 500$
2. Interval between measurements: $T = 1$
3. Initial components of velocity: $V_x = 100$; $V_y = 100$
4. Initial coordinates: $x_0 = 1000$; $y_0 = 1000$
5. Variance of noise σ_i, $\sigma_a^2 = 0.3^2$ for both a^x_i, a^y_i

Second, generated true values of range D and azimuth β:

$$\begin{aligned} D_i &= \sqrt{x_i^2 + y_i^2} \\ \beta_i &= \arctan\frac{x}{y} \end{aligned} \tag{2}$$

Third, generated measurements D^m and β^m of range D and azimuth β with given values of variances $\eta^D_i = 50^2$ and $\eta^\beta_i = 0.004^2$:

$$\begin{aligned} D^2_i &= D_i + \eta^D_i \\ \beta^2_i &= \beta_i + \eta^\beta_i \\ i &= 1, 3, 5, \dots, N \end{aligned} \tag{3}$$

Generated more accurate measurements of azimuth β^m provided by second observer that arrive between measurement times of the first observer with variance of measurement noise $\eta^2_{\beta add} = 0.001^2$:

$$\begin{aligned} \beta^2_i &= \beta_i + \eta^\beta_i \\ i &= 2, 4, 6, \dots, N \end{aligned} \tag{4}$$

Question 4 - 6
Initial conditions for Extended Kalman filter algorithms (initial filtered estimate of state vector $X_{0,0}$ (Equation 4), initial filtration error covariance matrix $P_{0,0}$ (Equation 5)) are given:

$$\begin{aligned} X_0 &= \begin{bmatrix} x_3^m \\ \frac{x_3^m - x_1^m}{2T} \\ y_3^m \\ \frac{y_3^m - y_1^m}{2T} \end{bmatrix} \\ x_1^m &= D_1^m \sin \beta_1^m \\ x_3^m &= D_3^m \sin \beta_3^m \\ y_1^m &= D_1^m \cos \beta_1^m \\ y_3^m &= D_3^m \cos \beta_3^m \end{aligned} \tag{5}$$

$$X_0 = \begin{bmatrix} 10^1 0 & 0 & 0 & 0 \\ 0 & 10^1 0 & 0 & 0 \\ 0 & 0 & 10^1 0 & 0 \\ 0 & 0 & 0 & 10^1 0 \end{bmatrix} \tag{6}$$

At every filtration step in the algorithm we linearized measurment equation by determining:

$$\frac{dh(\hat{X}_{i+1,i})}{dX_{i+1}} \tag{7}$$

We also counted in that measurement noise covariance matrix R that varies with time steps:

1. Observer 1, odd time steps
$$R = \begin{bmatrix} \sigma_D^2 & 0 \\ 0 & \sigma_\beta^2 \end{bmatrix} \tag{8}$$

2. Observer 2, even time steps
$$R = \sigma_{\beta add}^2 \tag{9}$$

At every filtration step depending on observer, measurement vector z_i and observation function $h(X_i)$ have different form.

Question 8
Ran Kalman filter algorithm over $M = 500$ times.

Question 9
Compared estimation result with measurement errors of D and *beta*

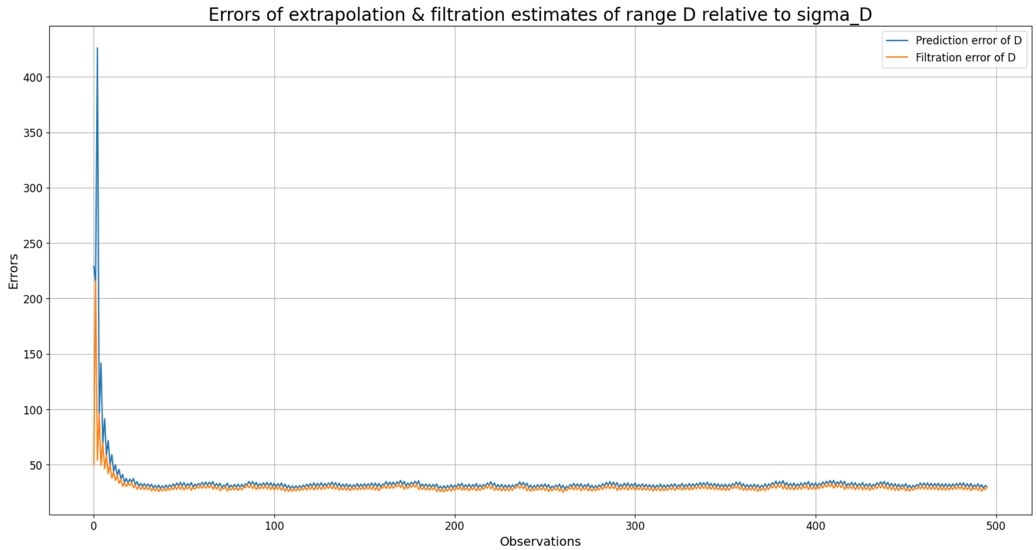


Figure 1: Error of range D

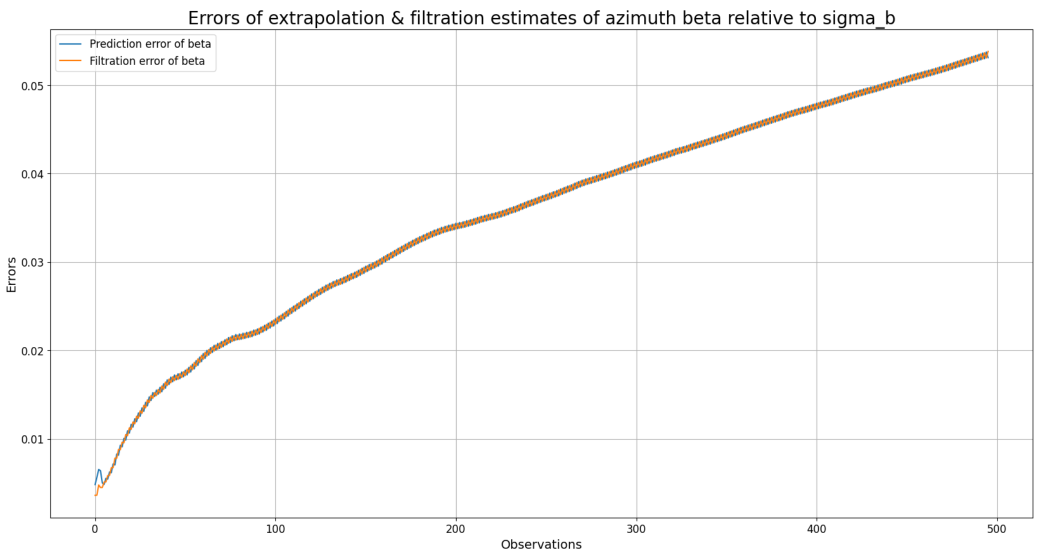


Figure 2: Error of azimuth β

We can see on Figure 3, that the values of predicted and filtered errors are moving in opposite phases during each step. It can be explained as follows. After using the Kalman filter on several dozen of the first measurements filter gain K_i becomes stable value K during the rest steps of using filter. Filtered and predicted estimations can be found using equations 10, 11:

$$X_{filtered,i} = X_{predicted,i} + K(z_i - HX_{predicted,i}) \tag{10}$$

$$X_{predicted,i+1} = \Phi_{i+1,i}X_{filtered,i} \tag{11}$$

Equation 10 part $K(z_i - HX_{predicted,i})$ can be considered as the noise of filtration η_i, which depends on the difference between the measurement and prediction values. If measurement z_i becomes 0, the noise value will rise, increasing filtration error on step i. After that on step $i+1$ due to the direct relation between the $X_{predicted,i+1}$ and $X_{filtered,i}$ (Equation 11) the prediction error $i + 1$ will also increase in comparison with the step i. $X_{filtered,i+1}$ will have smaller filtration error in comparison with the step i because on this step measurement value z_{i+1} will not be equal 0 leading to noise η_{i+1} and also decreasing prediction error on step $i+1$.

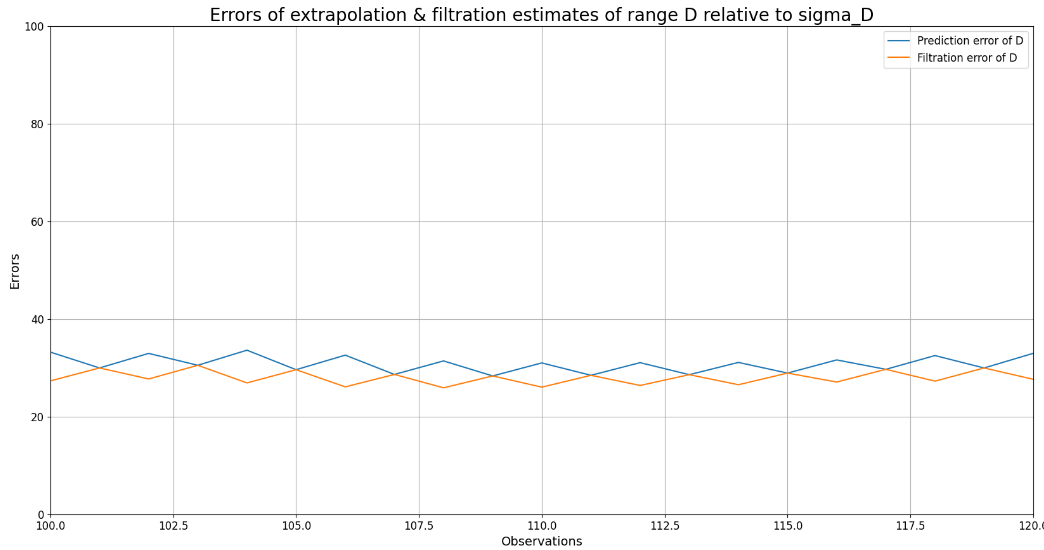


Figure 3: Scaled error of azimuth β

3 Conclusion

What we have learnt and tried:

1. How to develop a navigation filter by assimilating information which is coming from two different sources.

2. How to conduct assimilation algorithm.

3. Learnt how gaps affect on error covariance matrix and prediction.

4. What kind of differences we should observe when using only odd and odd and even steps together.

5. How implement Extended Kalman filter for data with gaps.

What we have reflected upon:

1. The effect of using odd and even steps together on prediction of errors β in every step.

2. We tried to process inaccurate data with a gap using the extended Kalman filter.

3. Reason why the accuracy of estimation varies for odd and even time steps.

4. Extended Kalman filter have really huge equations for observation matrices, where we loss a lot of time.

Contribution of each members:

1. Ilya: wrote the code for Extended Kalman filter.

2. Ruslan: wrote the code for plots and the report.

3. Yunseok: wrote the code and report.

In [122…

```
import numpy as np
import matplotlib.pyplot as plt
import random
```

In [123…

```
#Question 1. True trajectory
T = 2
N = 500
#Initial state vector
X_true_0 = np.array([[1000], [100], [1000], [100]])
#State vector
X_true = np.zeros((4, N))
#Transition matrix
Phi = np.array([[1, T, 0, 0], [0, 1, 0, 0], [0, 0, 1, T], [0, 0, 0, 1]])
#Input matrix
G = np.array([[T**2 / 2, 0], [T, 0], [0, T**2 / 2], [0, T]])
#True noise
sigma_a_1 = 0.3 ** 2

#Observation matrix
#H = np.array([[1, 0, 0, 0], [0, 0, 1, 0]])

def true_trajectory_(X_0_1, sigma_a, N):
    X_0_ = np.zeros((4, N))
    D_ = np.zeros((N, 1))
    betta_ = np.zeros((N, 1))
    a_i = np.zeros((2, 1))

    a_i[0, 0] = np.random.normal(0, np.sqrt(sigma_a))
    a_i[1, 0] = np.random.normal(0, np.sqrt(sigma_a))
    X_0_[:, 0] = (Phi.dot(X_0_1) + G.dot(a_i)).reshape(-1)
    for n in range (1, N):
        a_i[0, 0] = np.random.normal(0, np.sqrt(sigma_a))
        a_i[1, 0] = np.random.normal(0, np.sqrt(sigma_a))

        X_0_[:, n] = (Phi.dot(X_0_[:, n-1].reshape(4, 1)) + G.dot(a_i)).reshape(-1)

    D_ = np.sqrt(X_0_[0, :] ** 2 + X_0_[2, :] ** 2)
    betta_= np.arctan(X_0_[0, :]/X_0_[2, :])
    return X_0_, D_, betta_
```

```python
X_true, D_true, betta_true = true_trajectory_(X_true_0, sigma_a_1, N)
```

In [124…

```python
#Measurments 1
sigma_D = 50**2
sigma_betta_1 = 0.004**2

def measurements_1(D_in, betta_in, sigma_D, sigma_betta, N):
    Z_m = np.zeros((2, N))
    for n in range(0, N-1, 2):
        Z_m[0, n] = D_in[n] + np.random.normal(0, np.sqrt(sigma_D))
        Z_m[1, n] = betta_in[n] + np.random.normal(0, np.sqrt(sigma_betta))

    return Z_m

zm_1 = measurements_1(D_true, betta_true, sigma_D, sigma_betta_1, N)
```

In [125…

```python
#Measurments 2
sigma_betta_2 = 0.001**2

def measurements_2(betta_in, sigma_betta, N):
    Z_m = np.zeros(N)

    for n in range(3, N, 2):
        Z_m[n] = betta_in[n] + np.random.normal(0, np.sqrt(sigma_betta))

    return Z_m

zm_2 = measurements_2(betta_true, sigma_betta_2, N)
```

In [136…

```python
#Kalman

X_0 = np.array([[zm_1[0, 2]*np.sin(zm_1[1, 2])], [(zm_1[0, 2]*np.sin(zm_1[1, 2]) - zm_1[0, 0]*np.sin(zm_1[1, 0])) / (2*T)],
                [zm_1[0, 2]*np.cos(zm_1[1, 2])], [(zm_1[0, 2]*np.cos(zm_1[1, 2]) - zm_1[0, 0]*np.cos(zm_1[1, 0])) / (2*T)]])
P_0 = np.eye(4)*(10**4)

Q = G.dot(G.T) * sigma_a_1
R_1 = np.array([[sigma_D, 0], [0, sigma_betta_1]])
R_2 = sigma_betta_2
```

```python
def Kalman(X_init, P_init, size_, z_1, z_2):
    #Filterd position
    X_pred = np.zeros((4, size_))
    P_pred = np.zeros((4, 4, size_))

    X_filt = np.zeros((4, size_))
    P_filt = np.zeros((4, 4, size_))

    dh_1 = np.zeros((2, 4, size_))
    dh_2 = np.zeros((1, 4, size_))

    K_1 = np.zeros((4, 2, size_))
    K_2 = np.zeros((4, 1, size_))

    X_filt[:, 3] = X_init[:, 0]
    P_filt[:, :, 3] = P_init
    D_betta_out = np.zeros((4, size_-4))

    #Covariance matrix R of measurements noise
    R_1 = np.array([[sigma_D, 0], [0, sigma_betta_1]])
    R_2 = sigma_betta_2
    I = np.eye(4)

    for n in range(4, size_):

        X_pred[:, n] = (Phi.dot(X_filt[:, n-1].reshape(4, 1))).reshape(-1)
        P_pred[:, :, n] = (Phi.dot(P_filt[:, :, n-1])).dot(Phi.T) + Q

        if n % 2 == 0:
            dh_1[:,:,n] = np.array([[X_pred[0, n] / np.sqrt(X_pred[0, n]**2 + X_pred[2, n]**2), 0, X_pred[2, n] / np.sqrt(X_pred[0

            D = np.sqrt(X_pred[0, n]**2 + X_pred[2, n]**2)
            bet = np.arctan(X_pred[0, n]/X_pred[2, n])
            h = np.array([D, bet])

            K_1[:, :, n] = P_pred[:, :, n].dot(dh_1[:, :, n].T).dot(np.linalg.inv(dh_1[:, :, n].dot(P_pred[:, :, n]).dot(dh_1[:, :
            P_filt[:, :, n] = (I - K_1[:, :, n].dot(dh_1[:, :, n])).dot(P_pred[:, :, n])
            X_filt[:, n] = (X_pred[:, n].reshape(4, 1) + K_1[:, :, n].dot(z_1[:, n].reshape(2, 1) - h.reshape(2, 1))).reshape(-1)

        else:
            dh_2[0, 0, n] = X_pred[2, n] / (X_pred[0, n]**2 + X_pred[2, n]**2)
            dh_2[0, 2, n] = -X_pred[0, n] / (X_pred[0, n]**2 + X_pred[2, n]**2)
```

```
            bet = np.arctan(X_pred[0, n]/X_pred[2, n])
            h = np.array([bet])

            K_2[:, :, n] = P_pred[:, :, n].dot(dh_2[:, :, n].T) / (dh_2[:, :, n].dot(P_pred[:, :, n]).dot(dh_2[:, :, n].T) + R_2)
            P_filt[:, :, n] = (I - K_2[:, :, n].dot(dh_2[:, :, n])).dot(P_pred[:, :, n])
            X_filt[:, n] = (X_pred[:, n].reshape(4, 1) + K_2[:, :, n]*(z_2[n] - h)).reshape(-1)
        D_betta_out[0, n-4] = np.sqrt(X_pred[0, n] ** 2 + X_pred[2, n] ** 2) #D predicted
        D_betta_out[1, n-4]= np.arctan2(X_pred[0, n],  X_pred[2, n]) #Betta predicted
        D_betta_out[2, n-4] = np.sqrt(X_filt[0, n] ** 2 + X_filt[2, n] ** 2) #D filtered
        D_betta_out[3, n-4]= np.arctan2(X_filt[0, n], X_filt[2, n]) #Betta filtered

    return X_pred, X_filt, D_betta_out
```

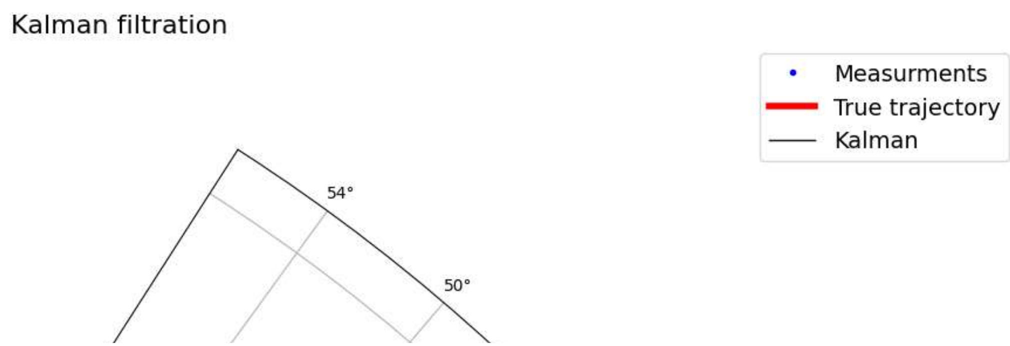
In [137…

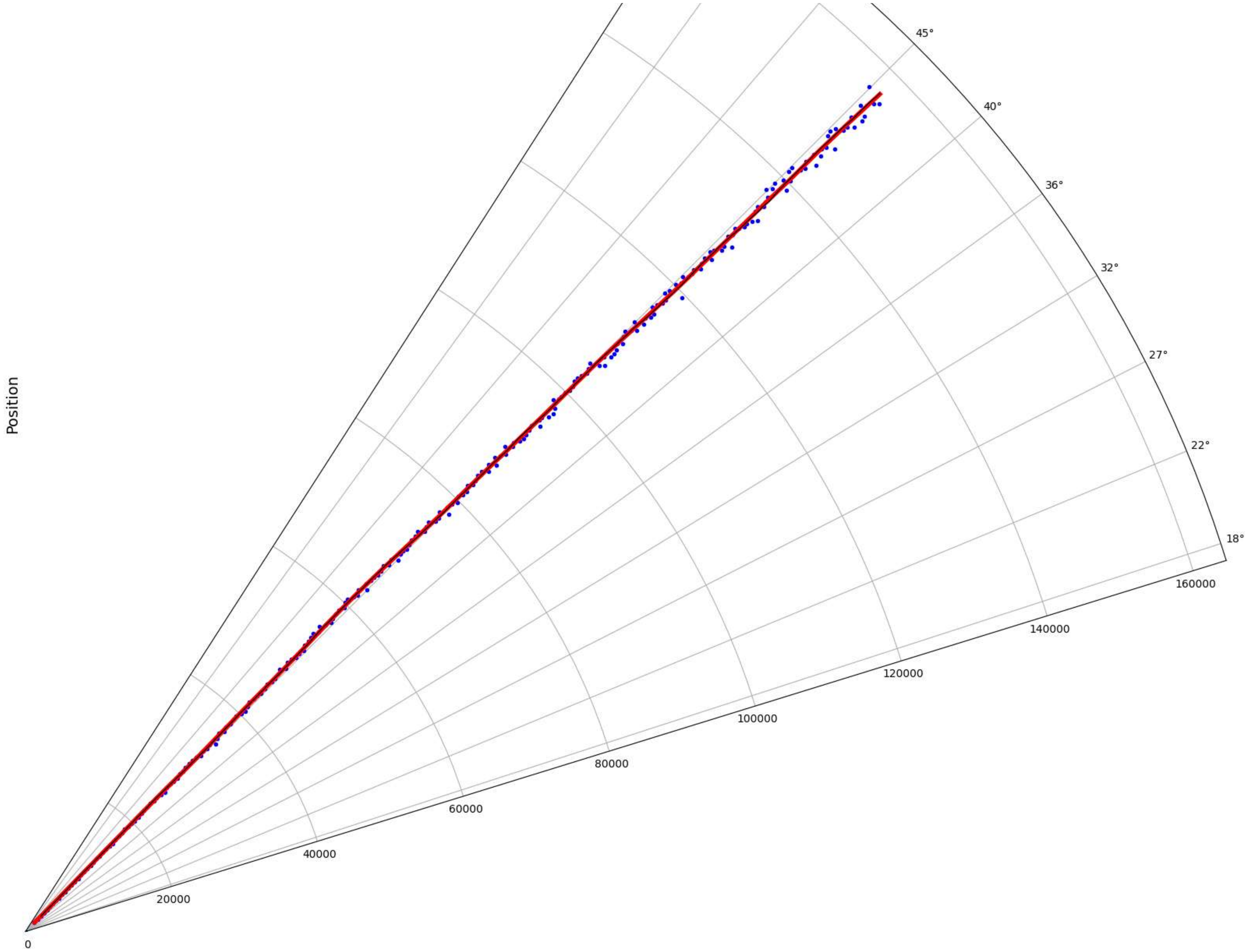
```
X_pred_1, X_filt_1, D_betta1 = Kalman(X_0, P_0, N, zm_1, zm_2)
```

In [138…

```
fig, ax = plt.subplots(figsize=(20,20), subplot_kw={'projection': 'polar'})
ax.set_title("Kalman filtration", fontsize = 16)
ax.set_ylabel("Position", fontsize = 14)
ax.set_xlabel("Observations", fontsize = 14)
ax.set_title("Kalman filtration", fontsize = 16)
ax.set_ylabel("Position", fontsize = 14)
ax.set_xlabel("Observations", fontsize = 14)
ax.plot(zm_1[1, :], zm_1[0, :], '.b',linewidth=1,  label = "Measurments")
ax.plot(betta_true, D_true, linewidth=4, label = "True trajectory", c = 'r')
ax.plot(D_betta1[3, :], D_betta1[2, :], linewidth=1, label = "Kalman", c = 'k')

plt.xlim(0.3, 1)
plt.legend(fontsize = 14,loc = 'best')
```

Out[138… <matplotlib.legend.Legend at 0x2554ad0b340>




Position
45°
40°
36°
32°
27°
22°
18°
0
20000
40000
60000
80000
100000
120000
140000
160000

Observations

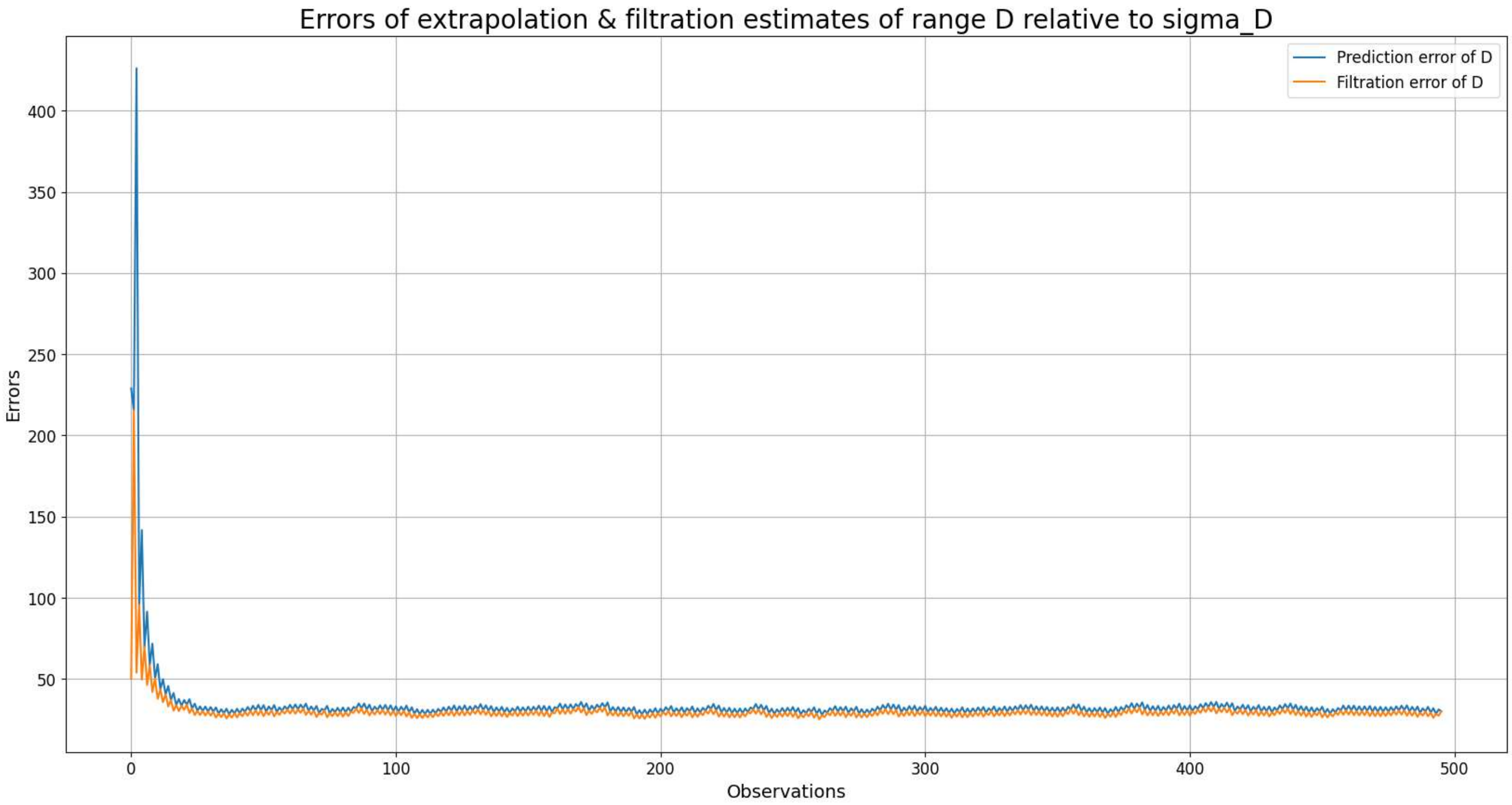
In [140…

```
def Errors(n, M):
    error_pred = np.zeros((2, n-4, M))
    error_filt = np.zeros((2, n-4, M))
    error_meas = np.zeros((3, n, M))

    for i in range(M):
        X_true_, D_true_, betta_true_ = true_trajectory_(X_true_0, sigma_a_1, N)
        zm_1 = measurements_1(D_true_, betta_true_, sigma_D, sigma_betta_1, N)
        zm_2 = measurements_2(betta_true, sigma_betta_2, N)

        X_pred, X_filt, D_betta = Kalman(X_0, P_0, N, zm_1, zm_2)

        error_pred[0, :, i] = (D_true_[4:] - D_betta[0, :]) ** 2
        error_pred[1, :, i] = (betta_true_[4:] - D_betta[1, :]) ** 2
        error_filt[0, :, i] = (D_true_[4:] - D_betta[2, :]) ** 2
        error_filt[1, :, i] = (betta_true_[4:] - D_betta[3, :]) ** 2

        error_meas[0, :, i] = (D_true_ - zm_1[0, :]) ** 2
        error_meas[1, :, i] = (betta_true_ - zm_1[1, :]) ** 2
        error_meas[2, :, i] = (betta_true_ - zm_2) ** 2

    Final_err_pred = np.sqrt(np.sum(error_pred, axis = 2) / (M - 1))
    Final_err_filt = np.sqrt(np.sum(error_filt, axis = 2) / (M - 1))
    Final_err_meas = np.sqrt(np.sum(error_meas, axis = 2) / (M - 1))

    return Final_err_pred, Final_err_filt, Final_err_meas

Final_err_pred, Final_err_filt, Final_err_meas = Errors(N, 500)
```

In [146…

```
fig, x = plt.subplots(figsize=(20,10))
x.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D", fontsize = 20)
x.set_xlabel("Observations", fontsize = 14)
x.set_ylabel("Errors", fontsize = 14)
x.plot(Final_err_pred[0, :], label = "Prediction error of D")
x.plot(Final_err_filt[0, :], label = "Filtration error of D")
x.tick_params(labelsize = 12)
```

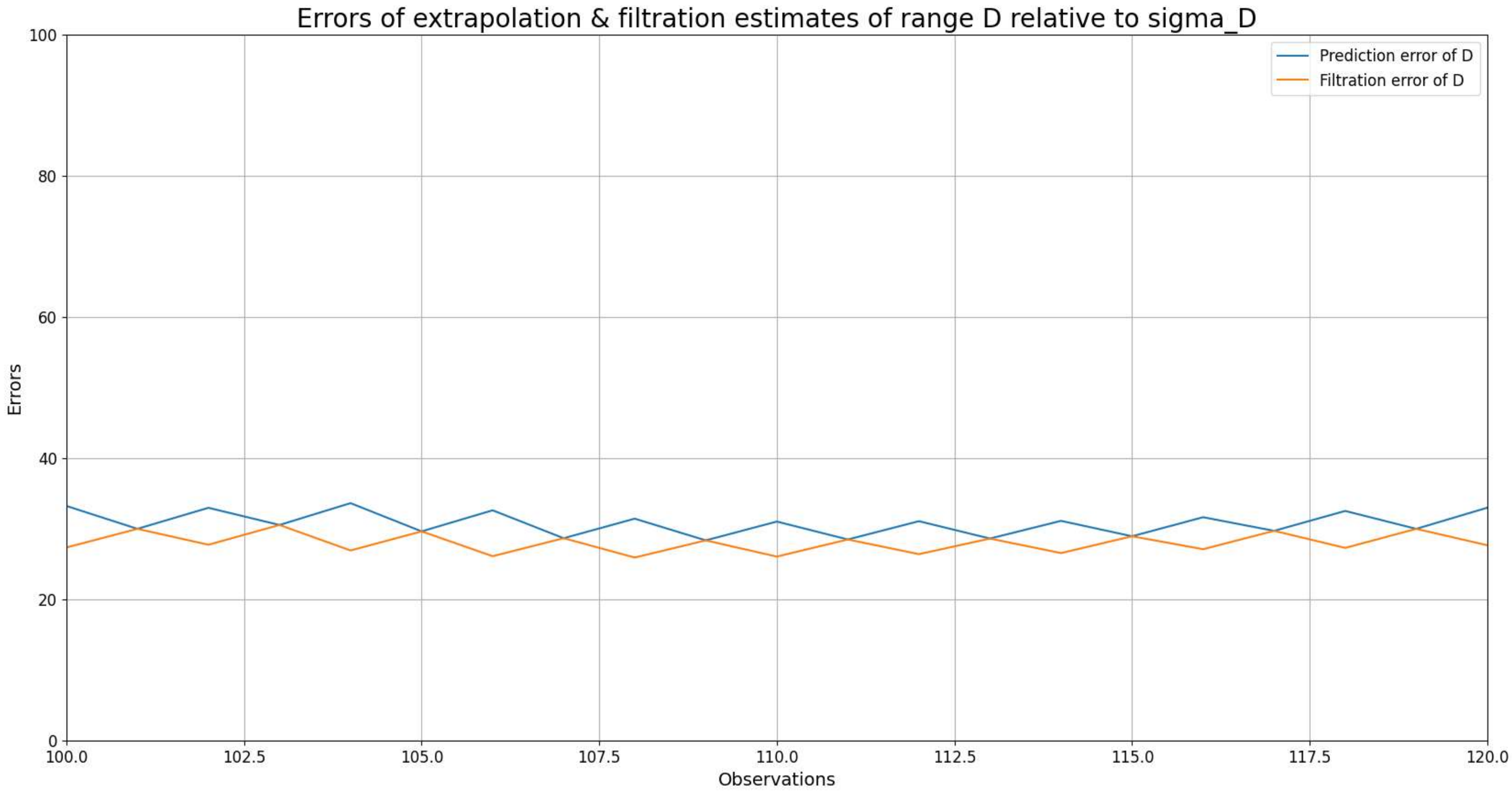
```
x.legend(fontsize = 12)
x.grid()
```



In [147…

```
fig, x = plt.subplots(figsize=(20,10))
x.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D", fontsize = 20)
x.set_xlabel("Observations", fontsize = 14)
x.set_ylabel("Errors", fontsize = 14)
x.plot(Final_err_pred[0, :], label = "Prediction error of D")
x.plot(Final_err_filt[0, :], label = "Filtration error of D")
x.tick_params(labelsize = 12)
x.legend(fontsize = 12)
```

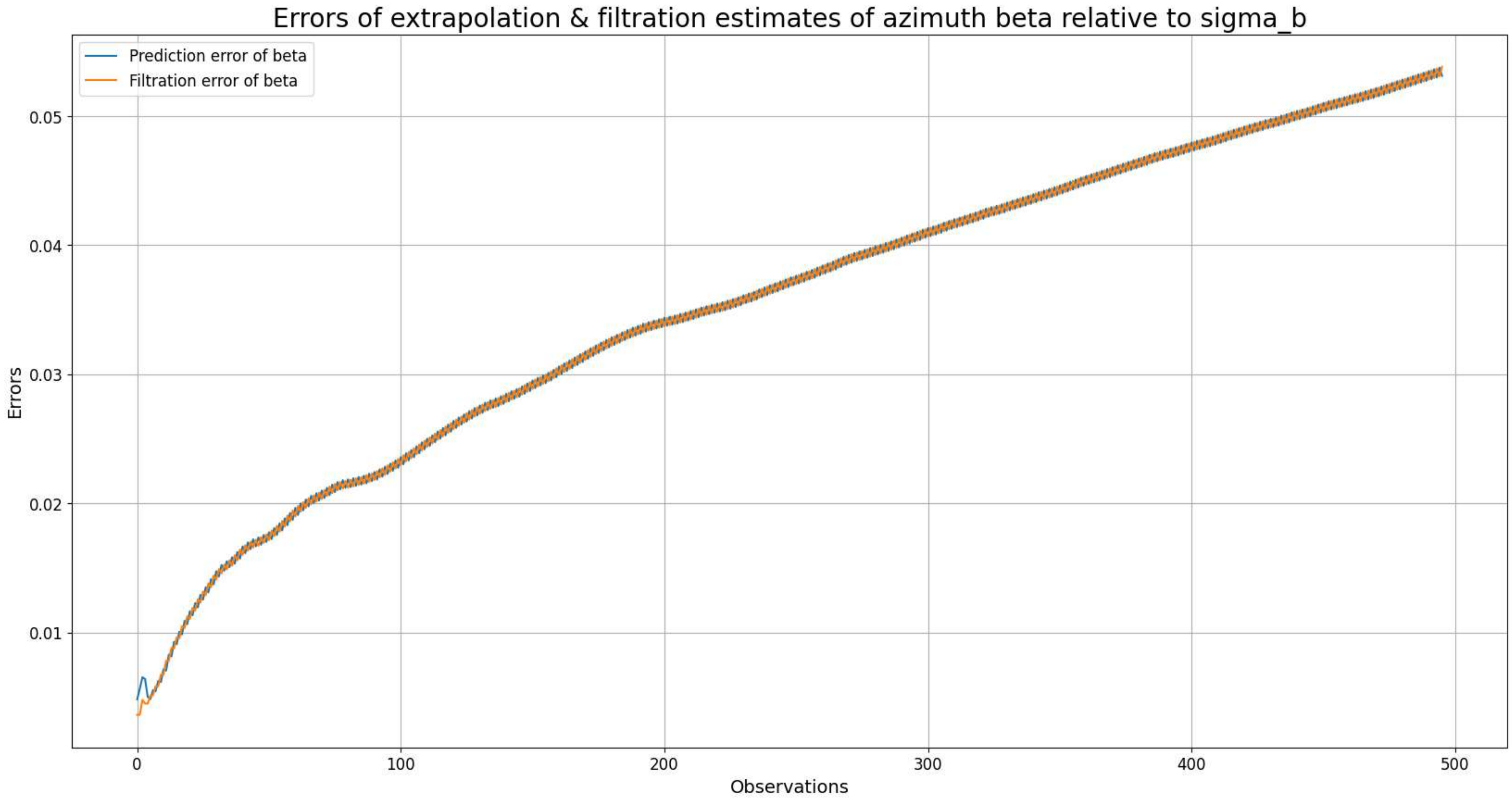
```
plt.xlim(100, 120)
plt.ylim(0, 100)

x.grid()
```



In [142…

```
fig, b = plt.subplots(figsize=(20,10))
b.set_title("Errors of extrapolation & filtration estimates of azimuth beta relative to sigma_b", fontsize = 20)
b.set_xlabel("Observations", fontsize = 14)
b.set_ylabel("Errors", fontsize = 14)
b.plot(Final_err_pred[1, :], label = "Prediction error of beta")
b.plot(Final_err_filt[1, :], label = "Filtration error of beta")
```

```
b.tick_params(labelsize = 12)
b.legend(fontsize = 12)
b.grid()
```



In [132…

```
fig, x = plt.subplots(figsize=(20,10))
x.set_title("Range D", fontsize = 20)
x.set_xlabel("Observations", fontsize = 14)
x.set_ylabel("Errors", fontsize = 14)
x.plot(Final_err_meas[2, 6:], label = "Measurement 1 error of D")
x.plot(Final_err_filt[1, 6:], label = "Estimation 2 error of D")
x.tick_params(labelsize = 12)
```

```
x.legend(fontsize = 12)
x.grid()
```