Skoltech

Skolkovo Institute of Science and Technology

Development of a tracking filter of a moving object when measurements and motion models are in different coordinate systems

Experimental Data Processing

Assignment №8

Team №10

Submitted by:
Yunseok Park
Ilya Novikov
Ruslan Kalimullin

Instructor:
Tatiana Podladchikova

MA060238 (Term 1B, 2022-2023)

October 20th, 2022

Contents

1 Introduction

The objective of this laboratory work is to develop a tracking filter of a moving object when measurements and motion model are in different coordinate systems. This problem is typical for radio navigation systems. Important outcome of this exercise is to detect main difficulties of practical Kalman filter implementation related with instability zone of a tracking filter, and to analyze conditions under which navigation system may become blind and filter diverges. This is important to prevent collisions and for other safety issues.

2 Work progress

Question 1 - 5

First, we generated the true trajectory X_i of an object that moves uniformly (deterministic) using Cartesian coordinates x_i, y_i and components of velocity V_i^x and V_i^y:

$$\begin{aligned} x_i &= x_{i-1} + V_{i-1}^x T \\ V_i^x &= V_{i-1}^x \\ y_i &= y_{i-1} + V_{i-1}^y T \\ V_i^y &= V_{i-1}^y \end{aligned} \tag{1}$$

Following initial conditions are given to generate true trajectory:

1. size of trajectory $N = 26$
2. Interval between measurements: $T = 2$
3. Initial components of velocity: $V_x = -50$; $V_y = -45$
4. Initial coordinates: $x_0 = \frac{13500}{\sqrt{2}}$; $y_0 = \frac{13500}{\sqrt{2}}$

Initial coordinates indicates that an object starts its motion **at a quite great distance from an observer**.

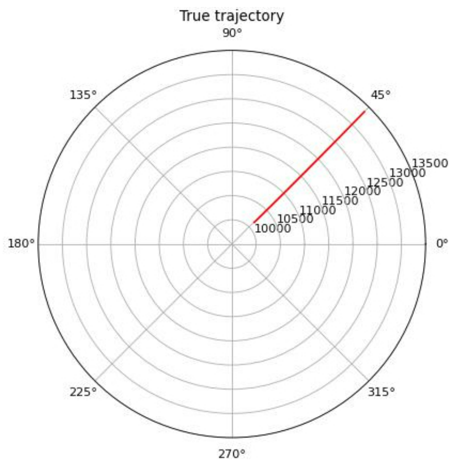


Figure 1: Motion of a quite great distance from an observer

Second, generate true values of range D and azimuth β with initial values $D_0 = \sqrt{x_0^2 + y_0^2}$

and $\beta_0 = \arctan \frac{x_0}{y_0}$:

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

You can see the visualized motion at a quite great distance from an observer in Figure 1.

Third, generate measurements D^m and β^m of range D and azimuth β with given values of variances $\eta_i^D = 20$ and $\eta_i^\beta = 0.02$:

$$\begin{aligned} D_i^2 &= D_i + \eta_i^D \\ \beta_i^2 &= \beta_i + \eta_i^\beta \end{aligned} \tag{3}$$

Fourth, transformed polar coordinates D^m and β^m to Cartesian ones and got pseudo-measurements of coordinates x and y:

$$\begin{aligned} x_i^m &= D_i^m \sin \beta_i^m \\ y_i^m &= D_i^m \cos \beta_i^m \end{aligned} \tag{4}$$

Then measurement vector z from pseudo-measurements of coordinates x and y was made:

$$z_i^c = \begin{vmatrix} x_i^m \\ y_i^m \end{vmatrix} \tag{5}$$

Question 5 - 10

Developed Kalman filter algorithm to estimate state vector X_i (extrapolation and filtration) and ran it over $M = 500$.

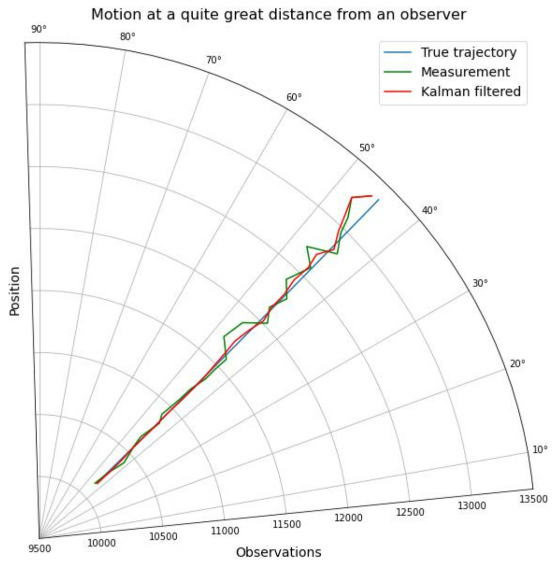


Figure 2: True trajectory, measurement and Kalman filter for quite great distance

In Figure 2 true trajectory, measurement and Kalman filter for a motion at a quite great distance from an observer is presented.

WE have plotted in Figure 3 Errors of extrapolation and filtration estimates of range D and azimuth β relative to σ_D and σ_β, respectively. By increasing steps, errors of range (D) and azimuth (β) have a decreasing manner. For range D, errors tend to decrease under 10, while errors of azimuth are much less which are around 0.01. In short, we could expect to have good filtration and prediction due to decreasing errors.

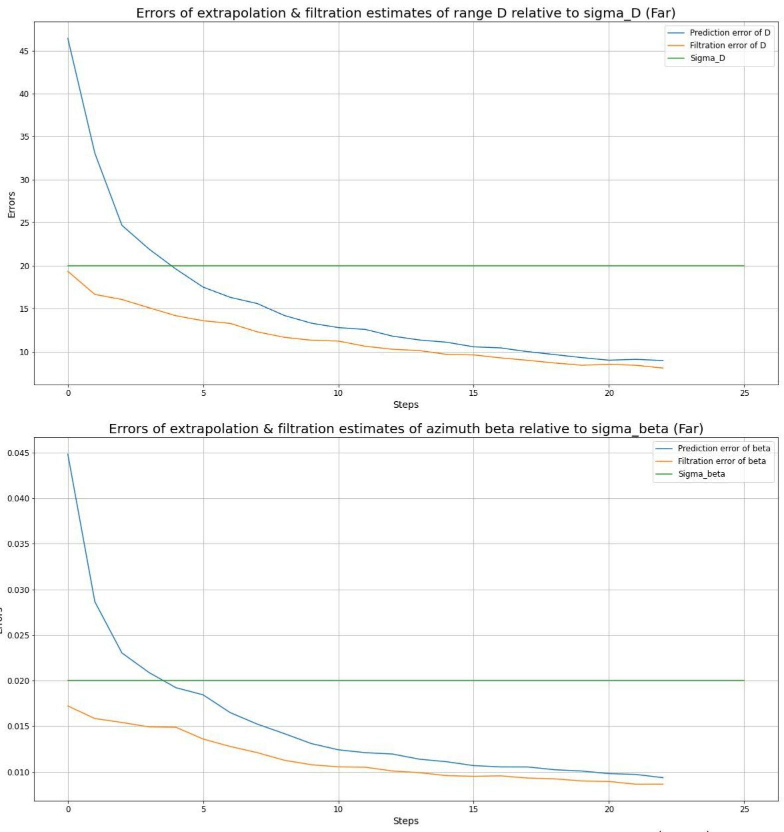


Figure 3: Errors of range D and azimuth β (Far)

Question 11

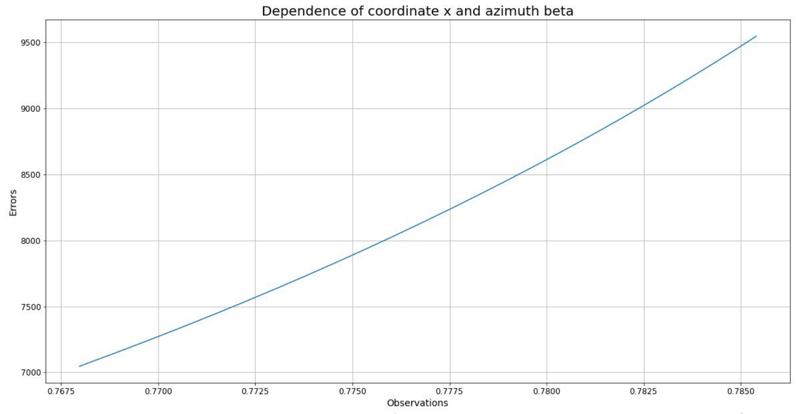


Figure 4: Dependence of coordinate x on azimuth β

In this question we analyzed dependence of coordinate x on azimuth β using the nonlinear relation $x = D \sin \beta$. When the dependence of coordinate x on β is checked, it can be seen from Figure 4 that the dependence is close to linear which means that the linearization errors are insignificant.

Question 12

For this part we calculated the condition number of covariance matrix R over the observation interval.

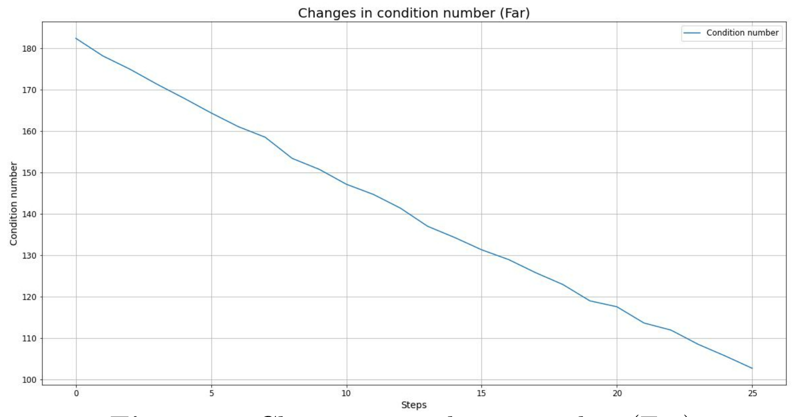


Figure 5: Change in condition number (Far)

If we check change in condition number for quite far case, in time, condition number will continuously decrease and get closer to 1. If condition number close to 1, then matrix R is well-conditioned.

Question 13

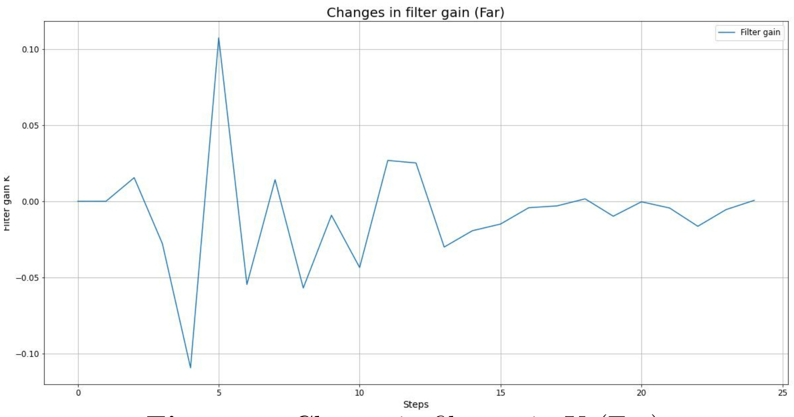


Figure 6: Change in filter gain K (Far)

In Figure 6 we can see the filter gain for this case. Value of $K(1,1)$ over observation interval don't always belong to interval (0,1). This is related to the fact that matrix R depends on polar measurements that have errors.

Question 14 - 15

We ran a second case of our task, where we used other initial conditions to generate a trajectory $x_0 = \frac{3500}{\sqrt{2}}$ and $y_0 = \frac{3500}{\sqrt{2}}$. This means that an object starts its motion at a quite close distance from an observer.

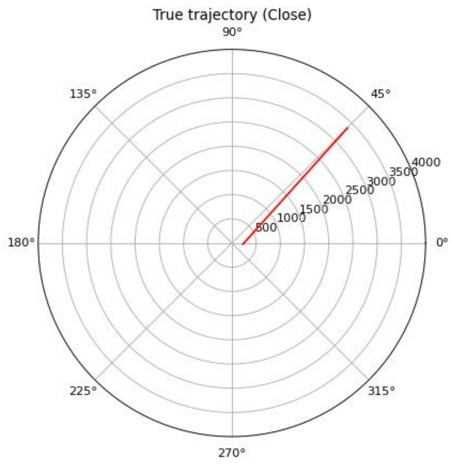


Figure 7: Motion of a quite great distance from an observer

Question 15 - 18

In Figure 8 we illustrated the error of range (D) and azimuth (β) for close case. As can be seen, after step 20, error starts to increase.

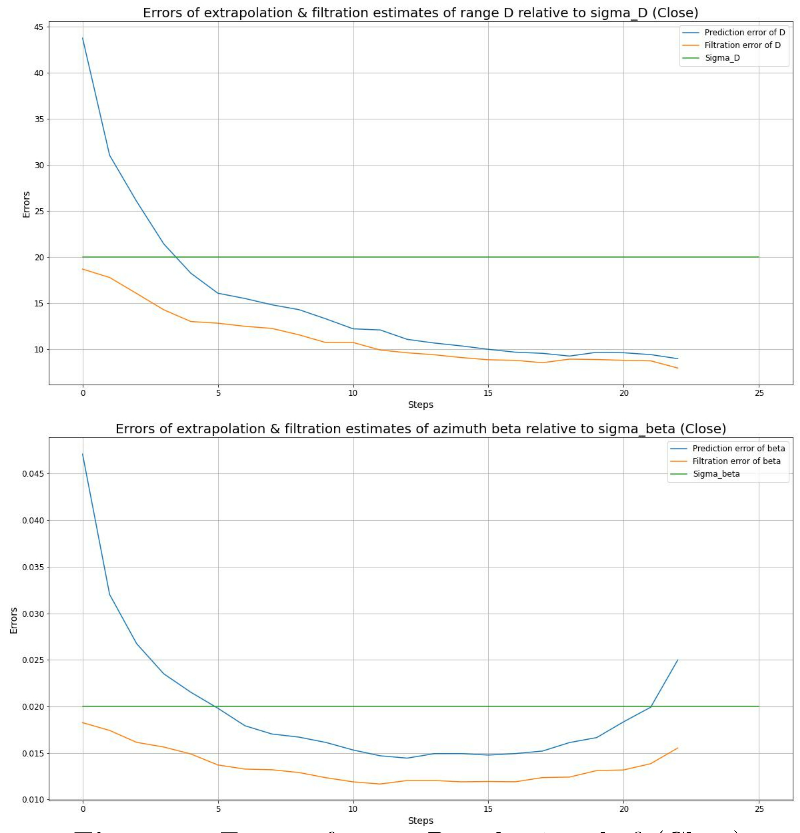


Figure 8: Errors of range D and azimuth β (Close)

Also, if we look at the Figure 10, after step 20, the condition number faces an increase and starts to get far from the condition number 1. This leads linearization errors to increase, i.e. linearization breaks. Expectedly, the filter diverges. At the end, the navigation system becomes blind. After getting too much relative to the observer, the filter stops working and cannot precisely determine the location of the object. That's why GPS has a limited accuracy when the object's position is so close to the observer.

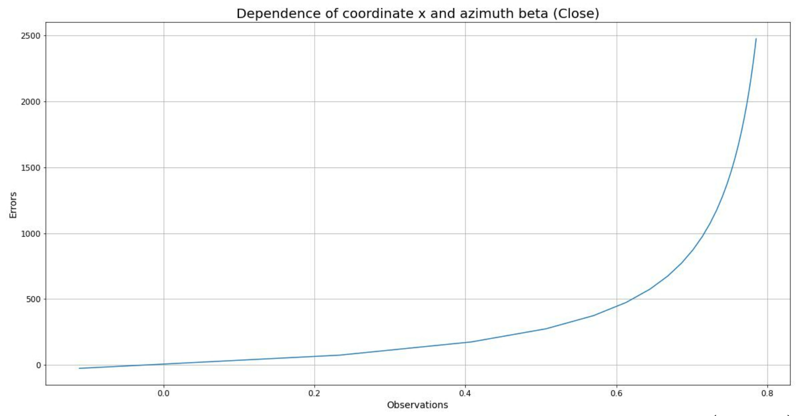


Figure 9: Dependence of coordinate x on azimuth β (Close)

As a second proof, in Figure 9 we can check to understand the deterioration in linear dependence.

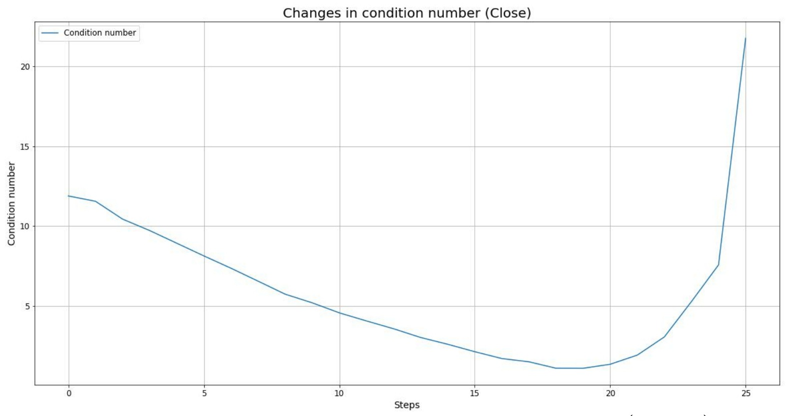


Figure 10: Change in condition number (Close)

Question 18

Conclusion on how linearization errors affect tracking accuracy and how important for tracking accuracy is starting position of a moving object (close or far from an observer):
The filter stops operating once we reach a specific step. As the object approached a certain position in relation to the observer, the linear approximation deteriorated. The better the linearization, the further the object is from the observer, because, as we can see in the close and far cases, the dependence of beta and x was non-linear as the object starts from a closer position. As a result, there are more estimation errors. The dependence is almost linear in the far case, and the errors can be ignored. The effect of non-linearity can be observed from condition number besides estimation errors. When the condition number starts to increase, we realized that the linearity spoiled.

Question 19 - 20

In the third case we use the same initial condition as 2nd one but with different values of variances of measurement noises $\eta_i^D = 50$ and $\eta_i^\beta = 0.0015$

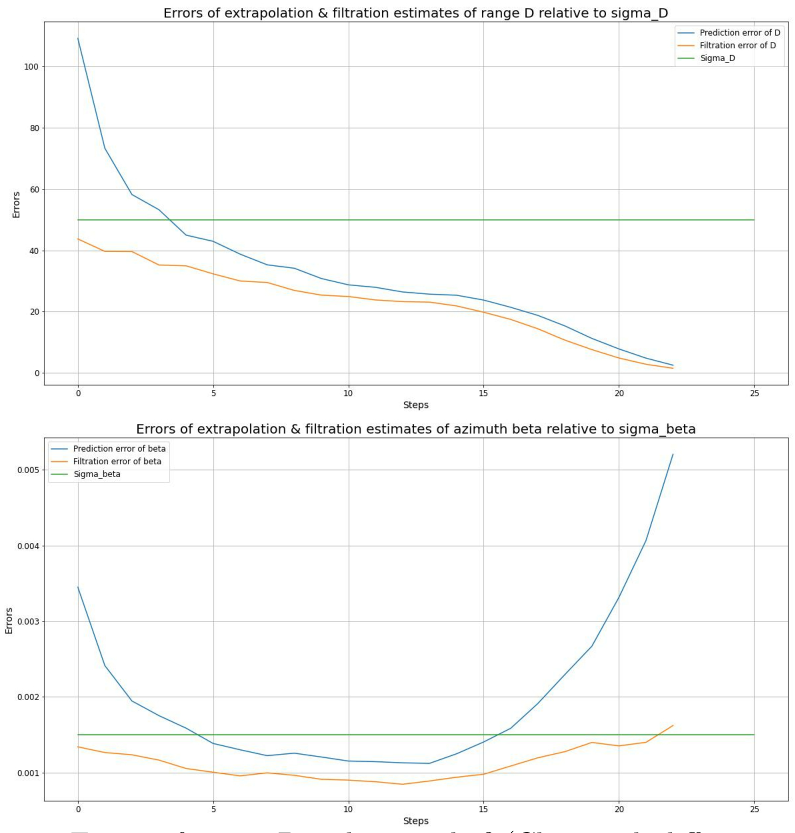


Figure 11: Errors of range D and azimuth β (Close with different variances)

At this part, we analyzed the motion which starts at the same (close) starting point with different variances. As can be seen, σ_D is higher and σ_β is less than the first case. As we increased the variance, the errors of range (D) decreases accordingly. On the other hand, decreasing the variance makes the errors of azimuth (β) higher.

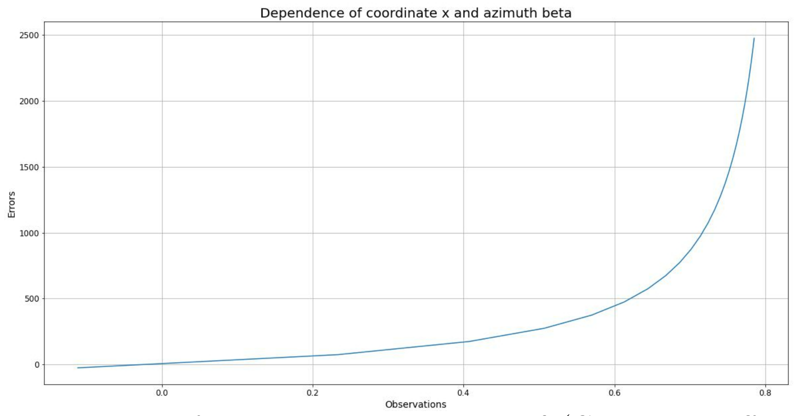


Figure 12: Dependence of coordinate x on azimuth β (Close with different variances)

After observing the increase in errors of β, non-linearity in the dependence of coordinate and azimuth was an expected result. Also, from Figure 13, condition number has a sharp increase and keeps getting far away from the value 1 which makes it ill-conditioned.

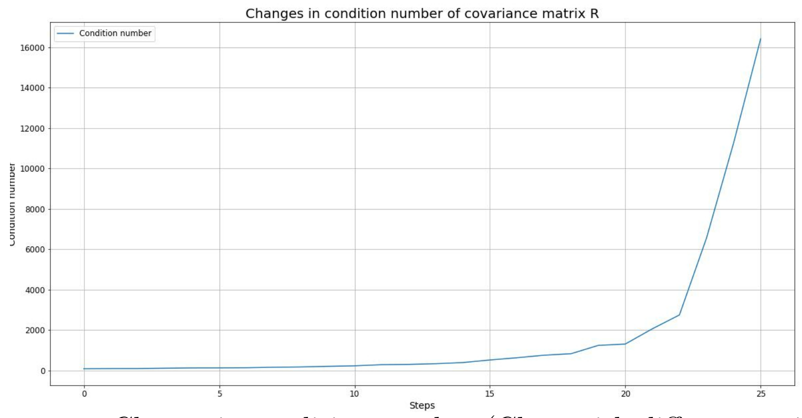


Figure 13: Change in condition number (Close with different variances)

Question 21

At the close case and the far case with different variances, the navigation system becomes blind and might diverge. When the linearization errors are huge, the condition number starts to rise from that point very rapidly. If we compare two diverged cases; at the close case, non-linearity is broken and the condition number starts to increase from step 20, similarly, the far case with the different variances has almost the same manner. Yet, the errors in total are less in the last case (far with different variances). Thus, ill-conditioned problem has the greatest influence on blindness/divergence of the filter. So, we can conclude that the linearization errors and ill-conditioned problems have influence on the divergence/blindness in the filters. By changing the position of observer or variances, ill-conditioned problem might be fixed.

3 Conclusion

What we have learnt and tried:

1. How to work with close and far trajectories (deterministic) rather than random motion.

2. How initial position affects the estimation accuracy.

3. Importance of transformation of coordinates.

4. Effect of changing variances on estimation errors of range and azimuth.

5. How to decide the matrix R is well or ill-conditioned

6. When filter gain (K) diverges, there should be a distortion in linearity(due to linear dependence) or ill-conditioned(due to condition number) problem might be addressed.

7. How to understand significance of linearization errors.

8. How to connect linear/non-linear dependence of x and b with estimation errors while the observer is far away or quite close.

What we have reflected upon:

1. In what conditions filter stops working properly and diverges eventually.

2. How linearity affects the estimation errors and proper working of the filter.

3. Closer proximity to a moving object causes bigger estimation errors, hence the farther away the object from the observer makes better the linearization

Contribution of each members:

1. Ilya: wrote the code for the first case

2. Ruslan: wrote the code for the third case

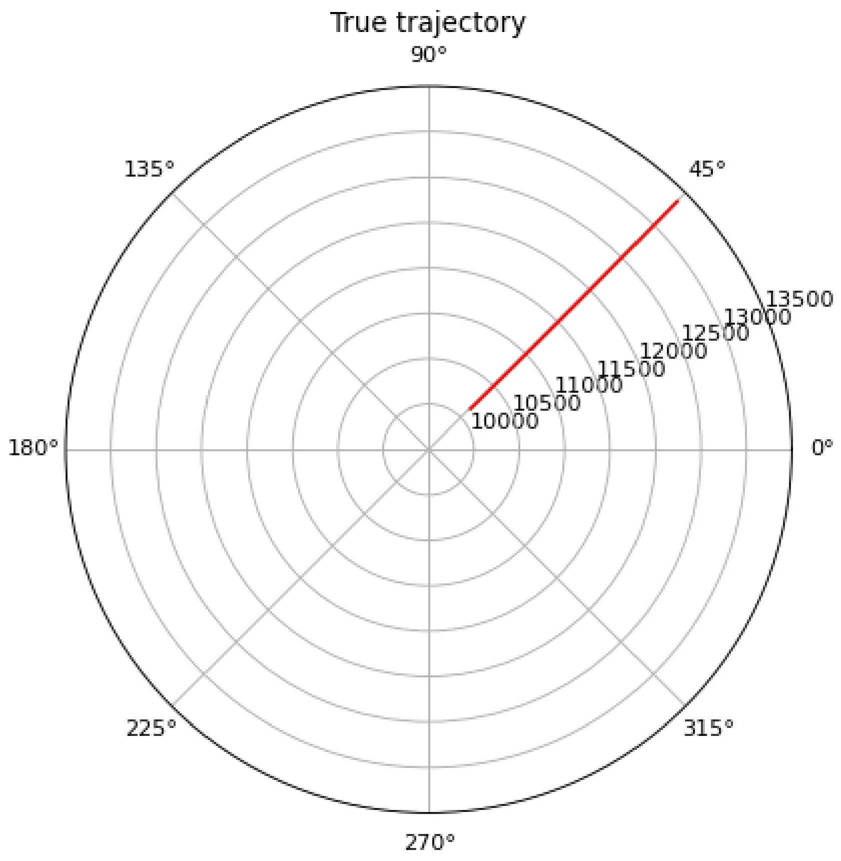
3. Yunseok: wrote the code for the second case and report

In [1]:
```
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
import random
```

In [2]:
```
#Question 1
#Generate a true trajectory of an object that moves uniformly. Trajectory is deterministic,
#as no random distrubance affect a motion

#size of trajectory
n = 26
#Interval between measuerments
T = 2

def true_c_trajectory(xy0 = np.ones(2) * 13500 / np.sqrt(2), v0 = np.array([-50, -45])):
    Xc = np.zeros((2, n))
    Xc[:, 0] = xy0
    for i in range(1, n):
        Xc[:, i] = Xc[:, i - 1] + v0 * T
    return Xc

c_traj = true_c_trajectory()
```

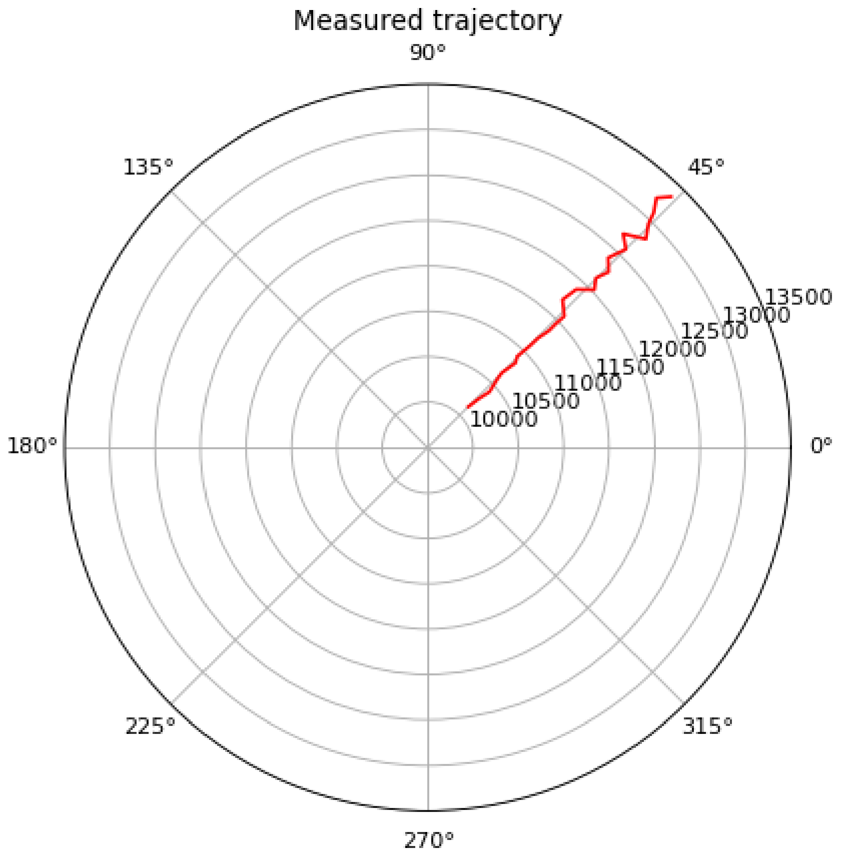
In [3]:
```
def true_p_trajectory(c_traj):
    Xp = np.zeros((2, n))
    Xp[0, :] = np.sqrt(c_traj[0, :] ** 2 + c_traj[1, :] ** 2) #D
    Xp[1, :] = np.arctan2(c_traj[0, :], c_traj[1, :]) #beta
    return Xp

p_traj = true_p_trajectory(c_traj)
```

In [4]:
```
# Plot
plt.figure(figsize=(6, 6), dpi=80)
plt.polar(p_traj[1, 1:], p_traj[0, 1:], c='red')
plt.title('True trajectory')
plt.ylim((9500,13500))
plt.savefig('True_trajectory(Far).jpg')
```



In [5]:
```python
#Question 3
#Generate measurements Dm and betam of range D and azimuth beta
def measurements(p_traj, sigma_D, sigma_b):
    Zm = np.zeros((2, n))
    Zm[0, :] = p_traj[0, :] + np.random.normal(0, sigma_D, n) #Dm
    Zm[1, :] = p_traj[1, :] + np.random.normal(0, sigma_b, n) #betam
    return Zm

zm = measurements(p_traj, 20, 0.02)
```

In [6]:
```python
# Plot
plt.figure(figsize=(6, 6), dpi=80)
plt.polar(zm[1, 1:], zm[0, 1:], c='red')
plt.title('Measured trajectory')
plt.ylim((9500,13500))
plt.savefig('Measured_trajectory(Far).jpg')
```



In [7]:

```
#Question 4 - 5
#Transform polar coordinates to cartesian ones and get pseudo-measurements of coordinates
def pseudo_measurements(Zm):
    Zc = np.zeros((2, n))
    Zc[0, :] = Zm[0, :] * np.sin(Zm[1, :]) #xm
    Zc[1, :] = Zm[0, :] * np.cos(Zm[1, :]) #ym
    return Zc

zc = pseudo_measurements(zm)
```

In [8]:

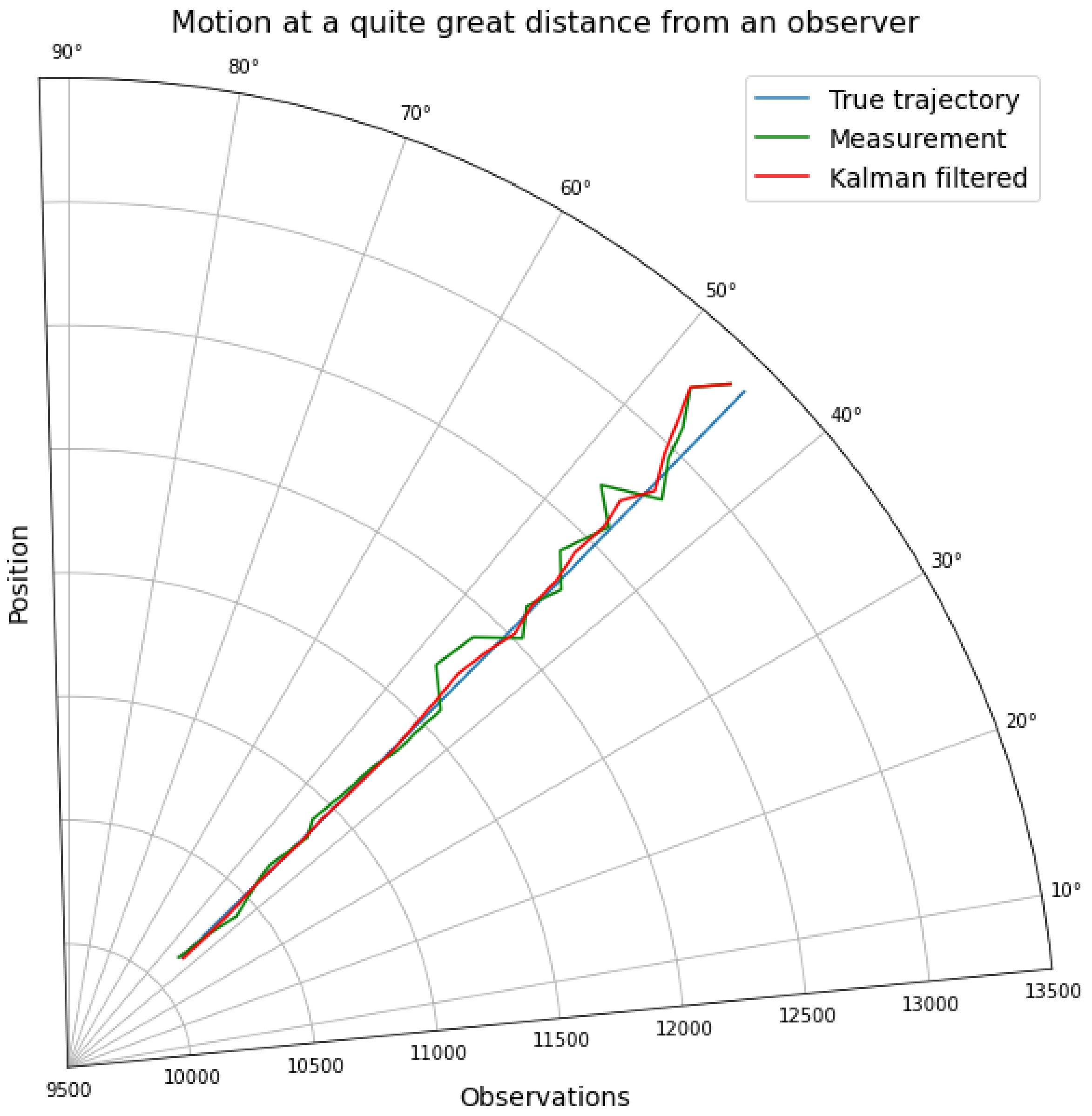
```
#Question 6 - 9
#Develop Kalman filter algorithm to estimate state vector (extrapolation and filtration)
```

```python
#At every extrapolation and filtration step we need to caculate range D and azimuth beta from extrapolated and filtered estimates
def create_R(zm, sigma_D, sigma_b):
    R = np.zeros((2, 2, n))
    for i in range(n):
        R[0, 0, i] = (np.sin(zm[1, i]) * sigma_D) ** 2 + (zm[0, i] ** 2) * ((np.cos(zm[1, i]) * sigma_b) ** 2)
        R[0, 1, i] = np.sin(zm[1, i]) * np.cos(zm[1, i]) * (sigma_D ** 2 - (zm[0, i] ** 2) * (sigma_b ** 2))
        R[1, 0, i] = np.sin(zm[1, i]) * np.cos(zm[1, i]) * (sigma_D ** 2 - (zm[0, i] ** 2) * (sigma_b ** 2))
        R[1, 1, i] = (np.cos(zm[1, i]) * sigma_D) ** 2 + (zm[0, i] ** 2) * ((np.sin(zm[1, i]) * sigma_b) ** 2)
    return R

R = create_R(zm, 20, 0.02)
```

In [9]:
```python
def Kalman_filter(meas, R, X0 = np.array([40000, -20, 40000, -20]), P0 = np.eye(4) * (10 ** 10)):
    #Observation matrix
    phi = np.array([[1, T, 0, 0], [0, 1, 0, 0], [0, 0, 1, T], [0, 0, 0, 1]])
    #Transition matrix
    H = np.array([[1, 0, 0, 0], [0, 0, 1, 0]])

    #Initialization of arrays
    X_pred = np.zeros((4, n))
    P_pred = np.zeros((4, 4, n))
    X_filt = np.zeros((4, n))
    P_filt = np.zeros((4, 4, n))
    K = np.zeros((4, 2, n))
    pf = np.zeros((4, n))
    I = np.eye(4)

    # intial conditions
    X_filt[:, 0] = X0
    P_filt[:, :, 0] = P0

    for k in range(1, n):
        #Prediction
        X_pred[:, k] = phi.dot(X_filt[:, k - 1].reshape(4, 1)).reshape(4)
        P_pred[:, :, k] = (phi.dot(P_filt[:, :, k - 1])).dot(phi.T)

        #Filtration
        K[:, :, k] = np.dot(P_pred[:, :, k].dot(H.T), np.linalg.inv((H.dot(P_pred[:, :, k])).dot(H.T) + R[:, :, k]))
        X_filt[:, k] = (X_pred[:, k].reshape(4, 1) + K[:, :, k].dot(meas[:, k].reshape(2, 1) - H.dot(X_pred[:, k].reshape(4, 1))
        P_filt[:, :, k] = (I - K[:, :, k].dot(H)).dot(P_pred[:, :, k])

    K = np.delete(K, 0, axis = 2)
    #Predicted
    pf[0, :] = np.sqrt(X_pred[0, :] ** 2 + X_pred[2, :] ** 2)
    pf[1, :] = np.arctan2(X_pred[0, :], X_pred[2, :])
    #Filtered
    pf[2, :] = np.sqrt(X_filt[0, :] ** 2 + X_filt[2, :] ** 2) #D
    pf[3, :] = np.arctan2(X_filt[0, :], X_filt[2, :]) #beta

    return pf, K
```

In [10]:
```
pf, K = Kalman_filter(zc, R)
```

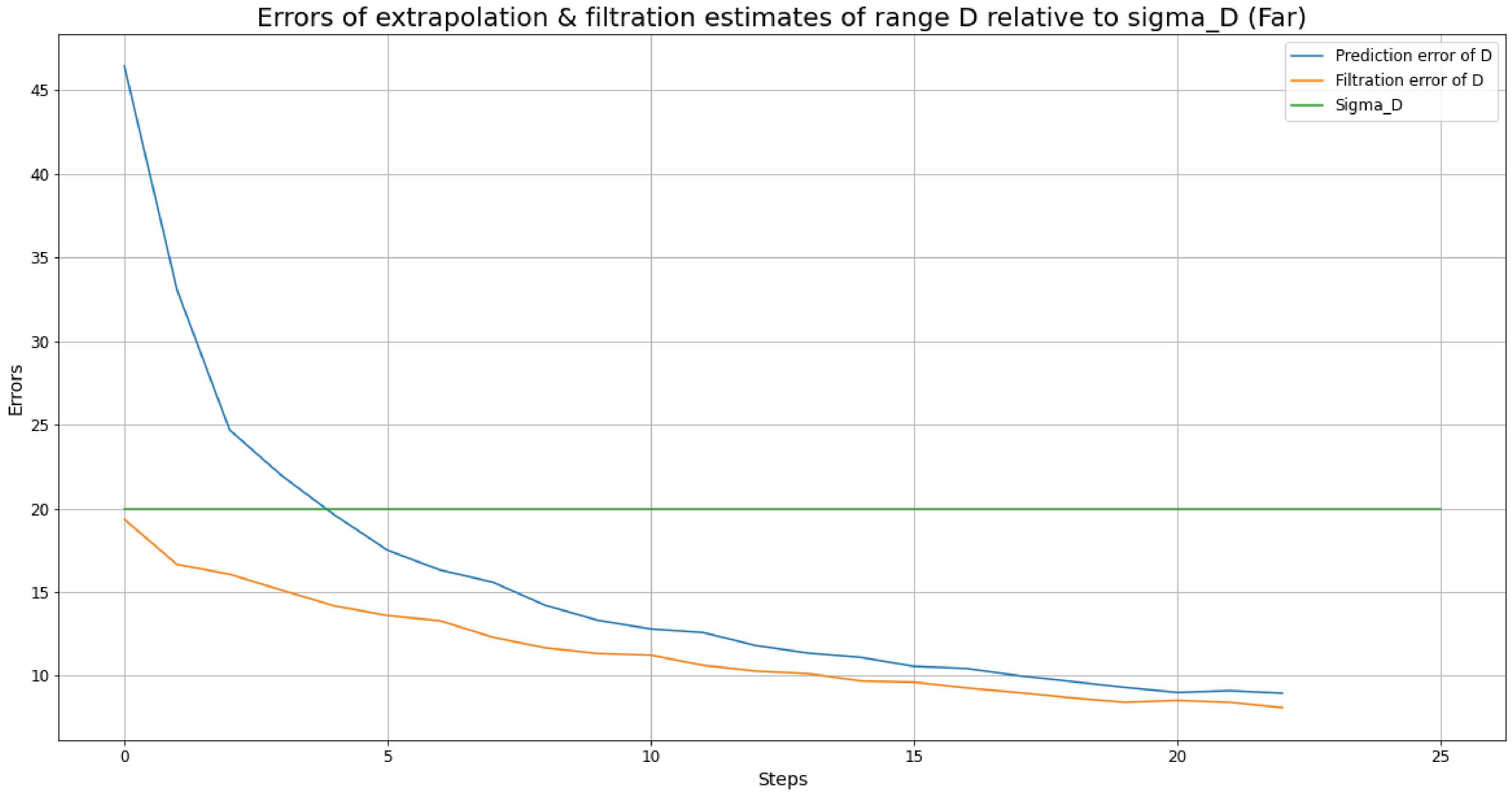
In [11]:
```
# Plot
fig, ax = plt.subplots(figsize=(10, 10), subplot_kw={'projection': 'polar'})
ax.set_title("Motion at a quite great distance from an observer", fontsize = 16)
ax.set_ylabel("Position", fontsize = 14)
ax.set_xlabel("Observations", fontsize = 14)
ax.plot(p_traj[1, 1:], p_traj[0, 1:], label='True trajectory')
ax.plot(zm[1, 1:], zm[0, 1:], label='Measurement', c='green')
ax.plot(pf[3, 1:], pf[2, 1:], label='Kalman filtered', c='red')
plt.xlim(0.1, 1.6)
plt.ylim((9500,13500))
plt.legend(fontsize = 14,loc = 'best')
plt.savefig('Kalman(far).jpg')
```



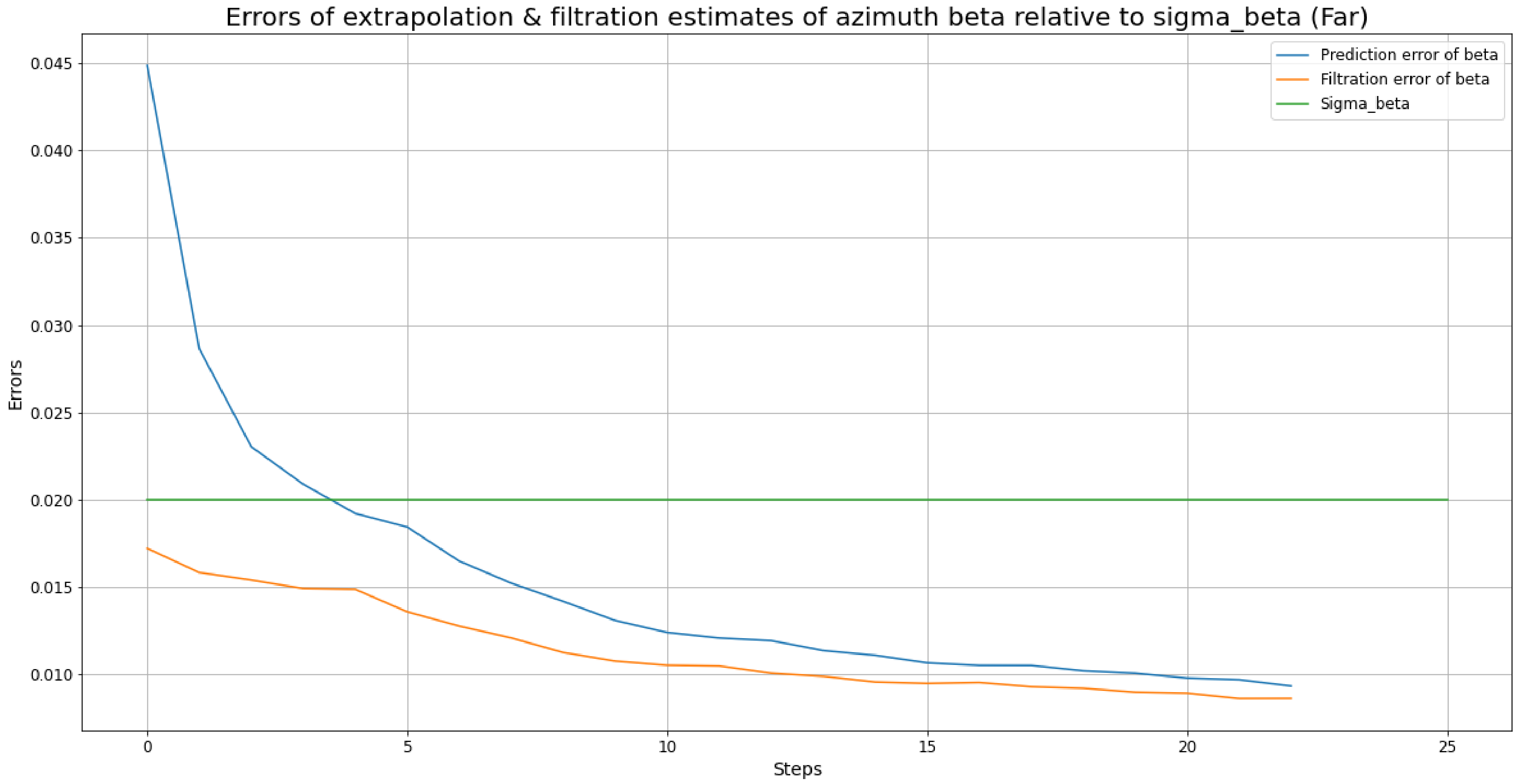
In [12]:
```
#Question 10
#Run Kalman filter algorithm over M=500. Calculate true errors of estimations:
#a) errors of extrapolation and filtration estimates of range D relative to sigma_D
#b) errors of extrapolation and filtration estimates of azimuth beta relative to sigma_beta
def Errors(M):
    error_pred = np.zeros((2, n, M))
    error_filt = np.zeros((2, n, M))

    for i in range(M):
        c_traj = true_c_trajectory()
        p_traj = true_p_trajectory(c_traj)
        zm = measurements(p_traj, 20, 0.02)
        zc = pseudo_measurements(zm)
        R = create_R(zm, 20, 0.02)
        pf = Kalman_filter(zc, R)[0]

        error_pred[:, :, i] = (p_traj - pf[:2, :]) ** 2
        error_filt[:, :, i] = (p_traj - pf[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))

    return Final_err_pred, Final_err_filt
```

In [13]:
```
Final_err_pred, Final_err_filt = Errors(500)
```

In [14]:
```
sg_D = np.full((n), 20)
fig, a = plt.subplots(figsize=(20,10))
a.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D (Far)", fontsize = 20)
a.set_xlabel("Steps", fontsize = 14)
a.set_ylabel("Errors", fontsize = 14)
a.plot(Final_err_pred[0, 3:], label = "Prediction error of D")
a.plot(Final_err_filt[0, 3:], label = "Filtration error of D")
a.plot(sg_D, label = "Sigma_D")
a.tick_params(labelsize = 12)
a.legend(fontsize = 12)
a.grid()
plt.savefig('D_error(far).jpg')
```

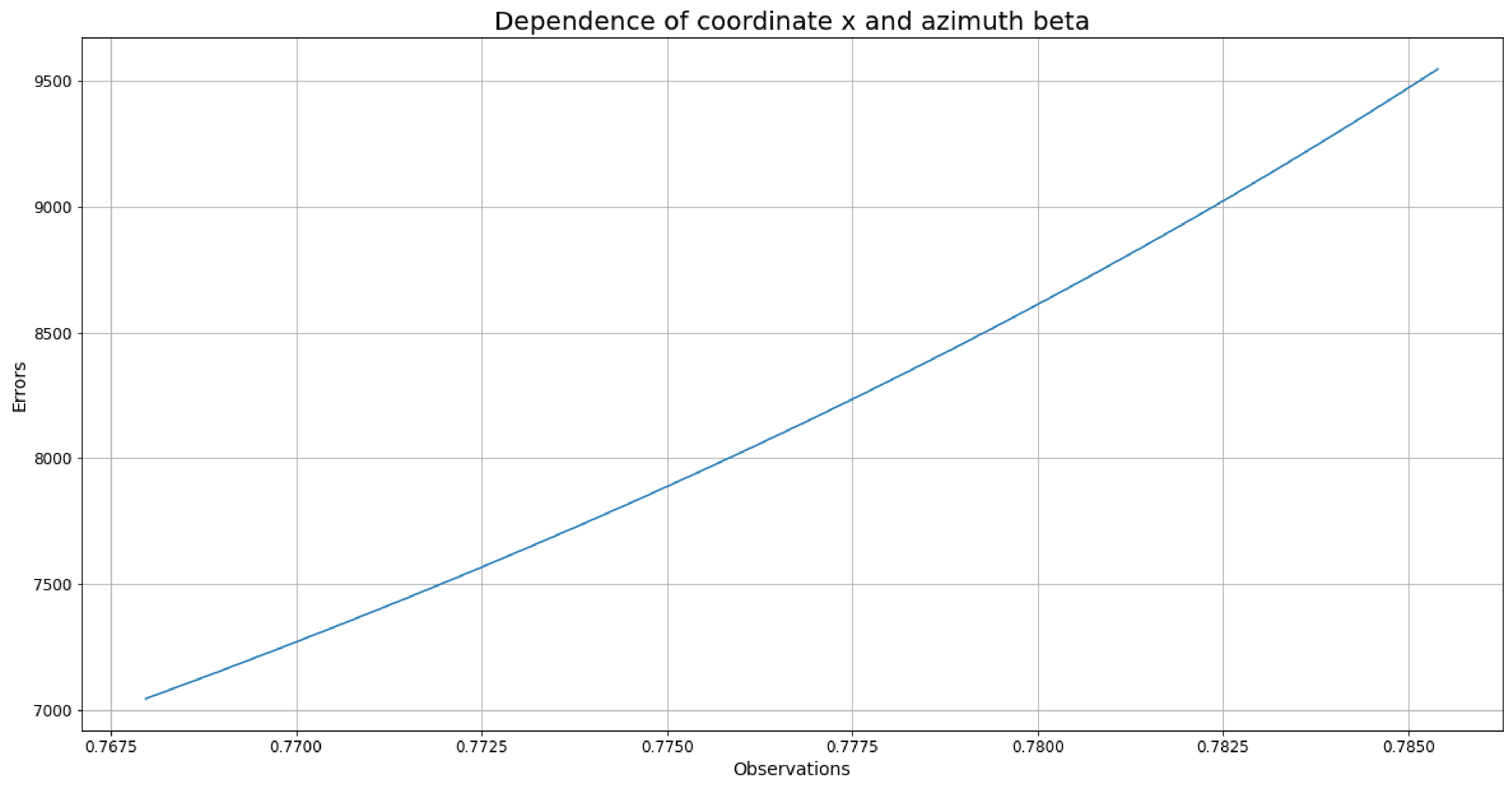


```
In [15]: sg_b = np.full((n), 0.02)
         fig, b = plt.subplots(figsize=(20,10))
         b.set_title("Errors of extrapolation & filtration estimates of azimuth beta relative to sigma_beta (Far)", fontsize = 20)
         b.set_xlabel("Steps", fontsize = 14)
         b.set_ylabel("Errors", fontsize = 14)
         b.plot(Final_err_pred[1, 3:], label = "Prediction error of beta")
         b.plot(Final_err_filt[1, 3:], label = "Filtration error of beta")
         b.plot(sg_b, label = "Sigma_beta")
         b.tick_params(labelsize = 12)
         b.legend(fontsize = 12)
         b.grid()
         plt.savefig('b_error(far).jpg')
```



In [16]:

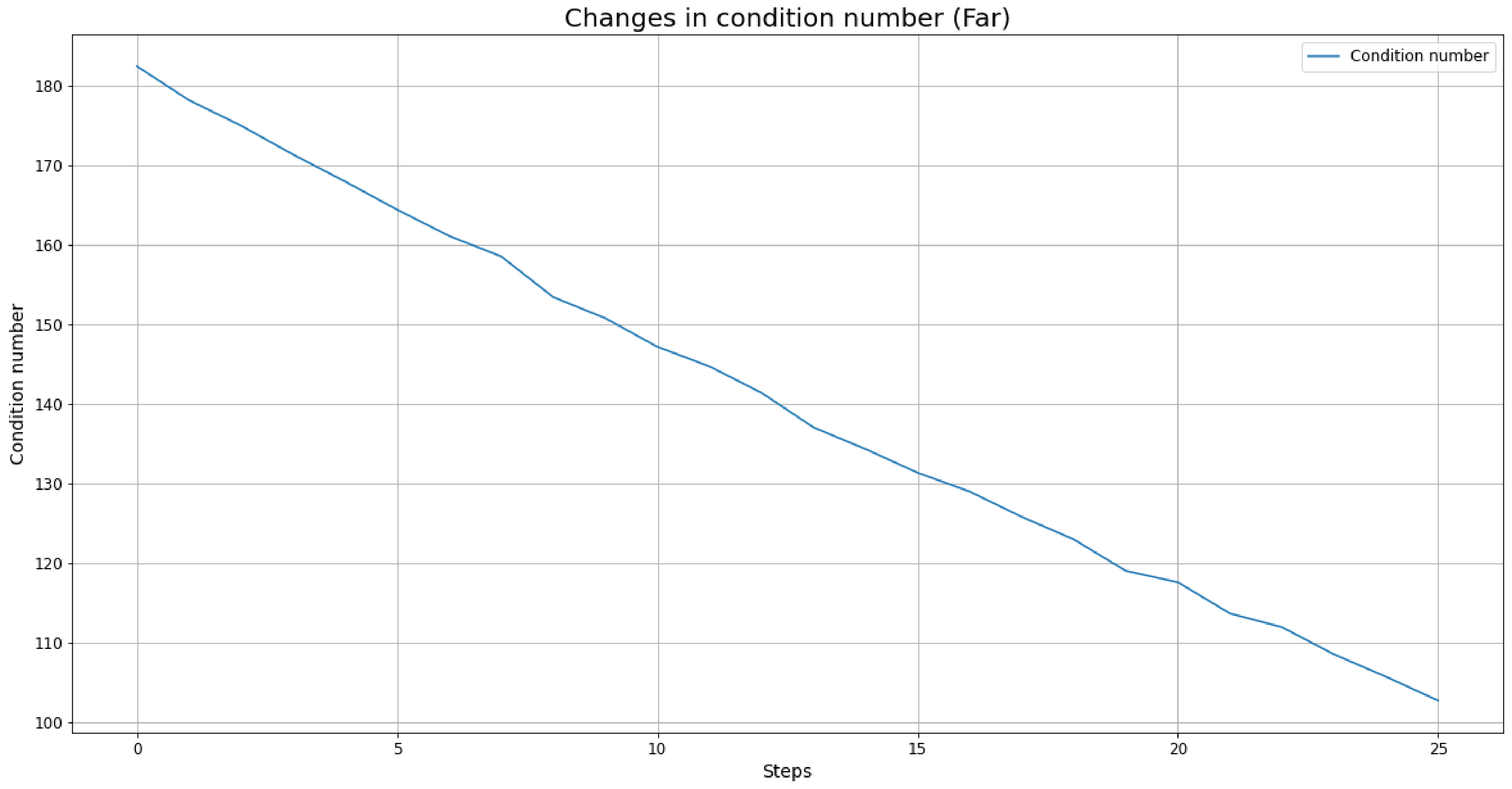
```python
#Question 11
#Analyze the dependence of coordinate x on aximuth beta
fig, c = plt.subplots(figsize=(20,10))
c.set_title("Dependence of coordinate x and azimuth beta", fontsize = 20)
c.set_xlabel("Observations", fontsize = 14)
c.set_ylabel("Errors", fontsize = 14)
c.plot(p_traj[1, :], c_traj[0, :], label='Dependence of coordinate x on azimuth beta - true data')
#c.plot(pf[0][3, :], c_traj[0, :], label='Dependence of coordinate x on azimuth beta - filtered')
c.tick_params(labelsize = 12)
c.grid()
plt.savefig('x_b(far).jpg')
```



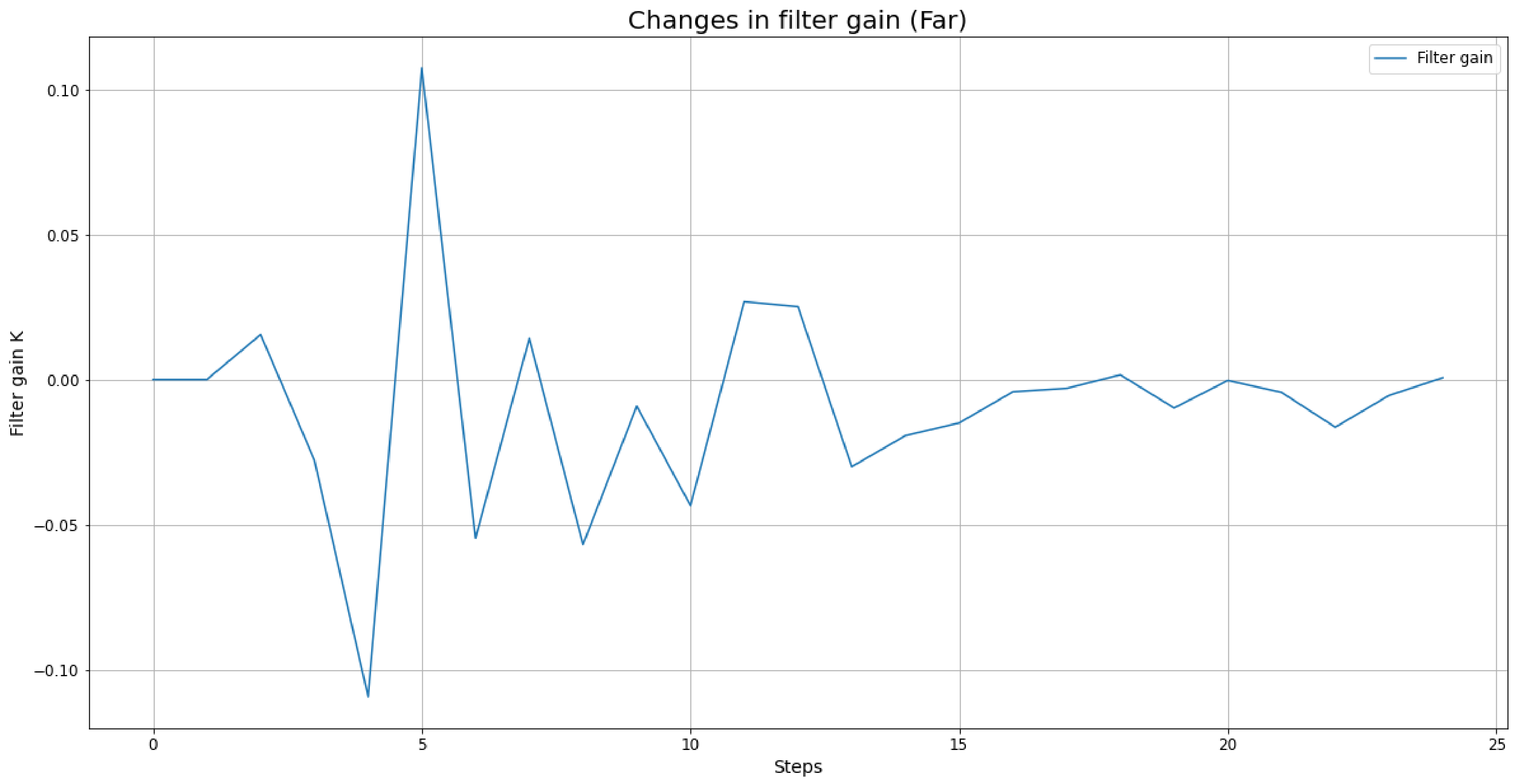
In [17]:
```
#Question 12
#Calculate condition number of covariance matrix R over the observation interval
def condition_number(sigma_D, sigma_b, zm):
    lambda1 = np.ones(n) * sigma_D ** 2
    lambda2 = (sigma_b * zm[0, :]) ** 2
    cond_num = np.zeros(n)
    for i in range(n):
        if lambda1[i] > lambda2[i]:
            cond_num[i] = lambda1[i] / lambda2[i]
        else:
            cond_num[i] = lambda2[i] / lambda1[i]
    return cond_num
```

```python
cond_num = condition_number(20, 0.02, zm)
```

In [18]:
```python
fig, d = plt.subplots(figsize=(20,10))
d.set_title("Changes in condition number (Far)", fontsize = 20)
d.set_xlabel("Steps", fontsize = 14)
d.set_ylabel("Condition number", fontsize = 14)
d.plot(cond_num, label = "Condition number")
d.tick_params(labelsize = 12)
d.legend(fontsize = 12)
d.grid()
plt.savefig('cond_n(far).jpg')
```



In [19]:
```python
#Question 13
#Analyze filter gain K. Dimension of filter gain in this case is 4x2
fig, e = plt.subplots(figsize=(20,10))
e.set_title("Changes in filter gain (Far)", fontsize = 20)
e.set_xlabel("Steps", fontsize = 14)
e.set_ylabel("Filter gain K", fontsize = 14)
e.plot(K[1, 1, :], label = "Filter gain")
e.tick_params(labelsize = 12)
e.legend(fontsize = 12)
e.grid()
plt.savefig('K(far).jpg')
```

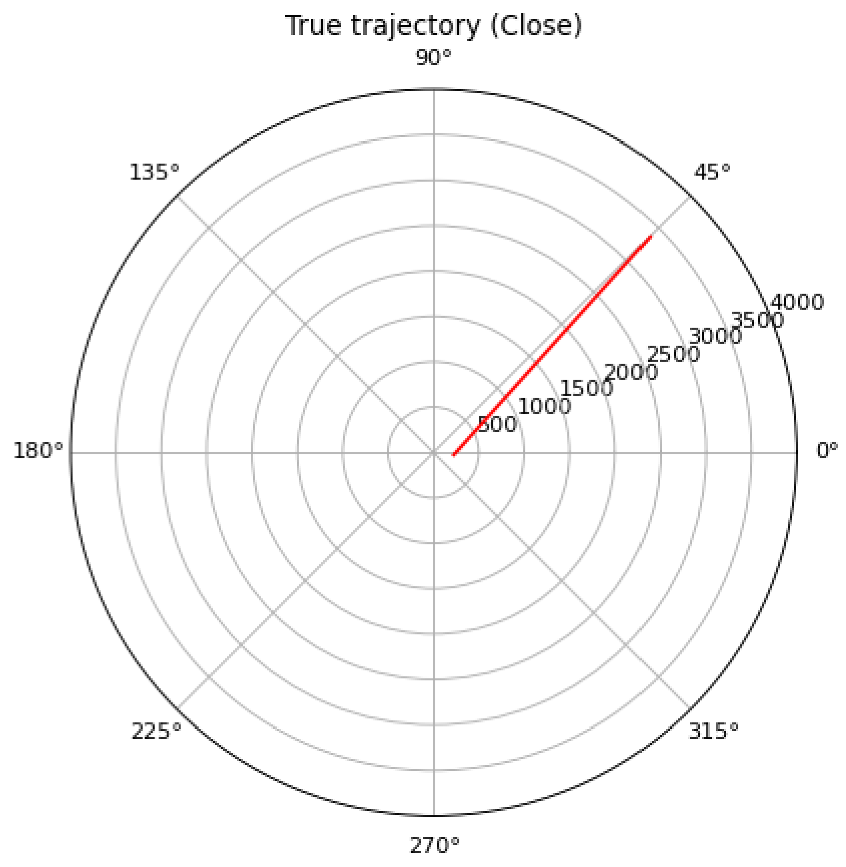


In [20]:
```python
#Question 14
```

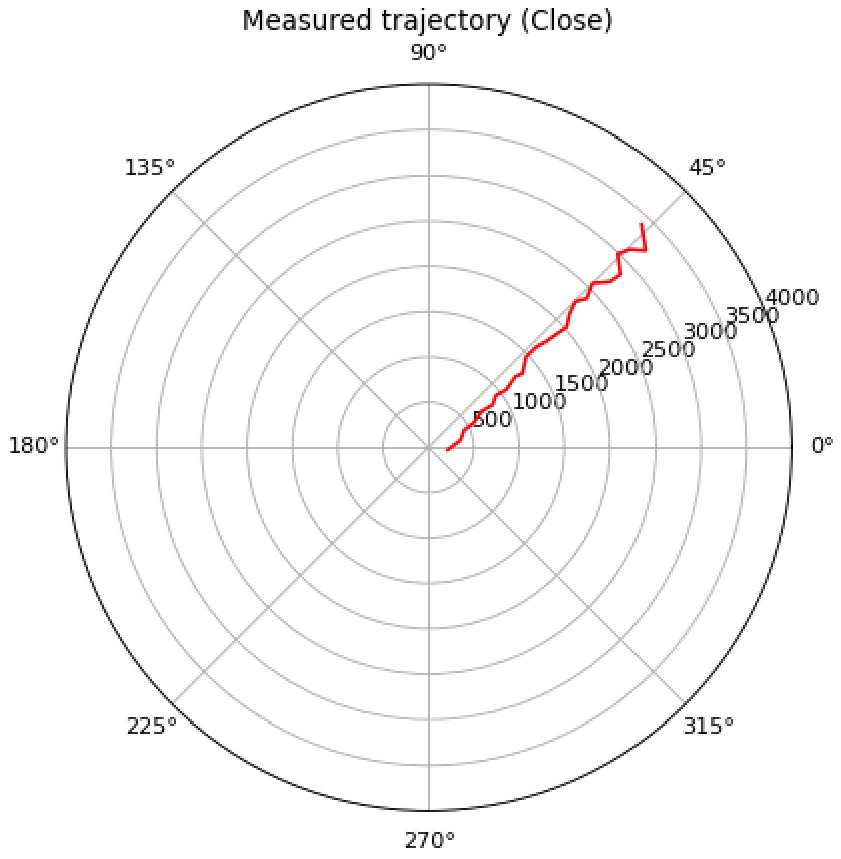
```
#Run filter again over M = 500, but use other initial conditions to generate a trajectory
#A quite close distance from an observer
c_traj2 = true_c_trajectory(xy0 = np.ones(2) * 3500 / np.sqrt(2), v0 = np.array([-50, -45]))
p_traj2 = true_p_trajectory(c_traj2)
zm2 = measurements(p_traj2, 20, 0.02)
zc2 = pseudo_measurements(zm2)
R2 = create_R(zm2, 20, 0.02)
pf2 = Kalman_filter(zc2, R2)[0]
cond_num2 = condition_number(20, 0.02, zm2)

def Errors2(M):
    error_pred2 = np.zeros((2, n, M))
    error_filt2 = np.zeros((2, n, M))

    for i in range(M):
        c_traj2 = true_c_trajectory(xy0 = np.ones(2) * 3500 / np.sqrt(2), v0 = np.array([-50, -45]))
        p_traj2 = true_p_trajectory(c_traj2)
        zm2 = measurements(p_traj2, 20, 0.02)
        zc2 = pseudo_measurements(zm2)
        cond_num2 = condition_number(20, 0.02, zm2)
        pf2 = Kalman_filter(zc2, R2)[0]

        error_pred2[:, :, i] = (p_traj2 - pf2[:2, :]) ** 2
        error_filt2[:, :, i] = (p_traj2 - pf2[2:, :]) ** 2

    Final_err_pred2 = np.sqrt(np.sum(error_pred2, axis = 2) / (M - 1))
    Final_err_filt2 = np.sqrt(np.sum(error_filt2, axis = 2) / (M - 1))

    return Final_err_pred2, Final_err_filt2

Final_err_pred2, Final_err_filt2 = Errors2(500)
```

In [21]:

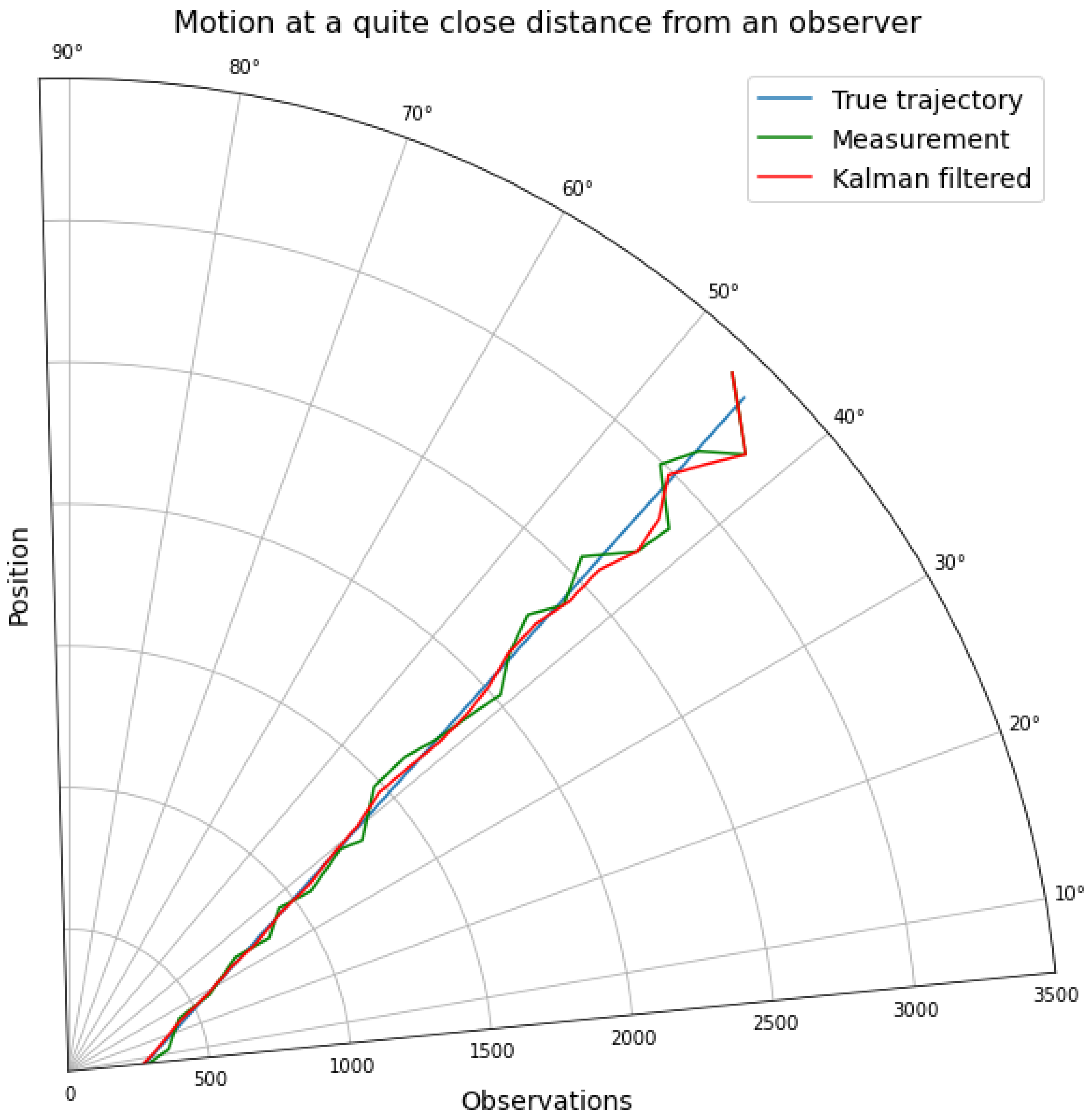
```
# Plot
plt.figure(figsize=(6, 6), dpi=80)
plt.polar(p_traj2[1, 1:], p_traj2[0, 1:], c='red')
plt.title('True trajectory (Close)')
plt.ylim((0, 4000))
plt.savefig('True_trajectory(close).jpg')
```



```
In [22]: # Plot
         plt.figure(figsize=(6, 6), dpi=80)
         plt.polar(zm2[1, 1:], zm2[0, 1:], c='red')
         plt.title('Measured trajectory (Close)')
         plt.ylim((0, 4000))
         plt.savefig('Measured_trajectory(close).jpg')
```

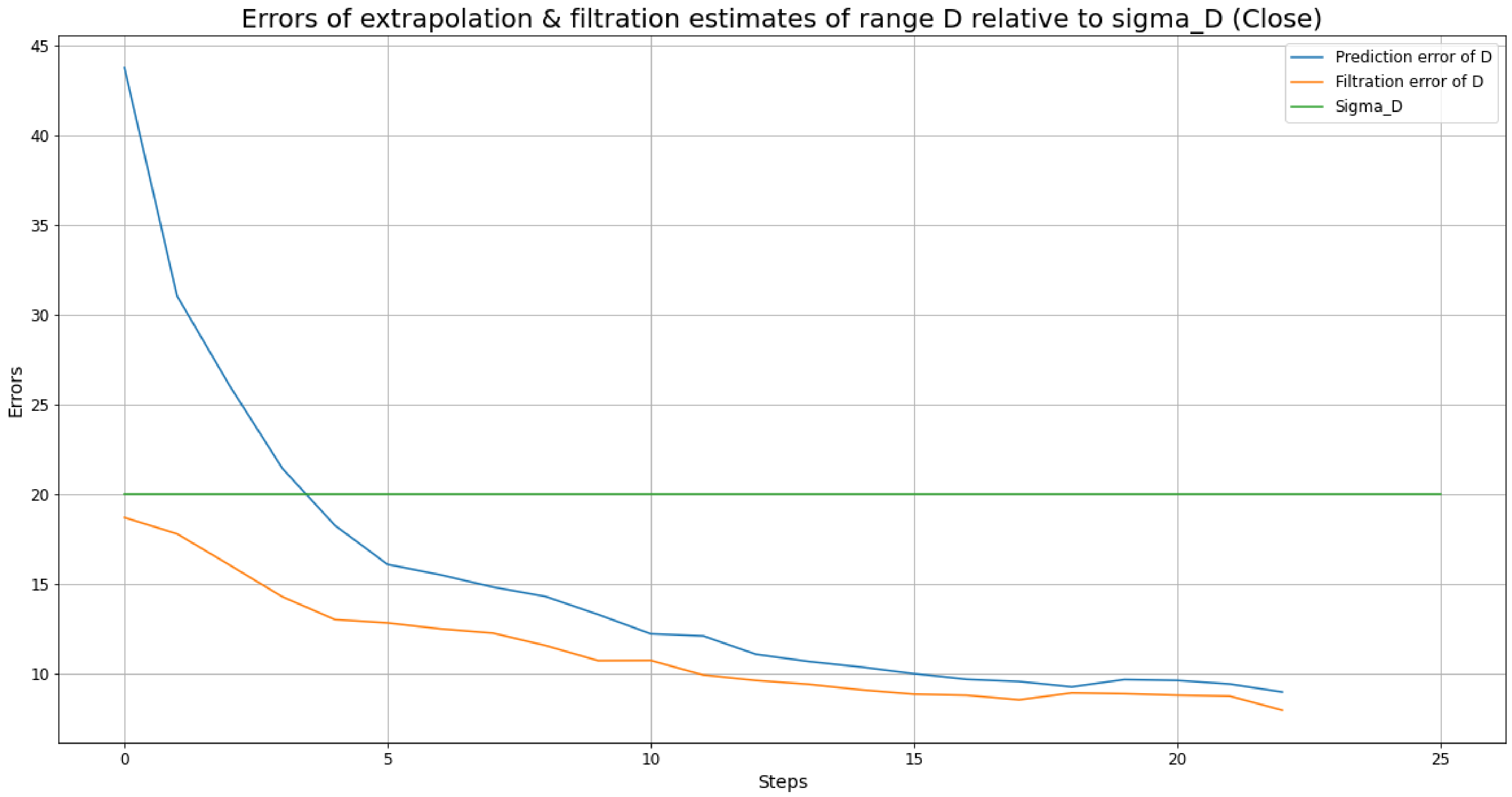


```
In [23]: # Plot
fig, ay = plt.subplots(figsize=(10, 10), subplot_kw={'projection': 'polar'})
ay.set_title("Motion at a quite close distance from an observer", fontsize = 16)
ay.set_ylabel("Position", fontsize = 14)
ay.set_xlabel("Observations", fontsize = 14)
ay.plot(p_traj2[1, 1:], p_traj2[0, 1:], label='True trajectory')
ay.plot(zm2[1, 1:], zm2[0, 1:], label='Measurement', c='green')
ay.plot(pf2[3, 1:], pf2[2, 1:], label='Kalman filtered', c='red')
plt.xlim(0.1, 1.6)
plt.ylim((0, 3500))
plt.legend(fontsize = 14,loc = 'best')
plt.savefig('Kalman(close).jpg')
```



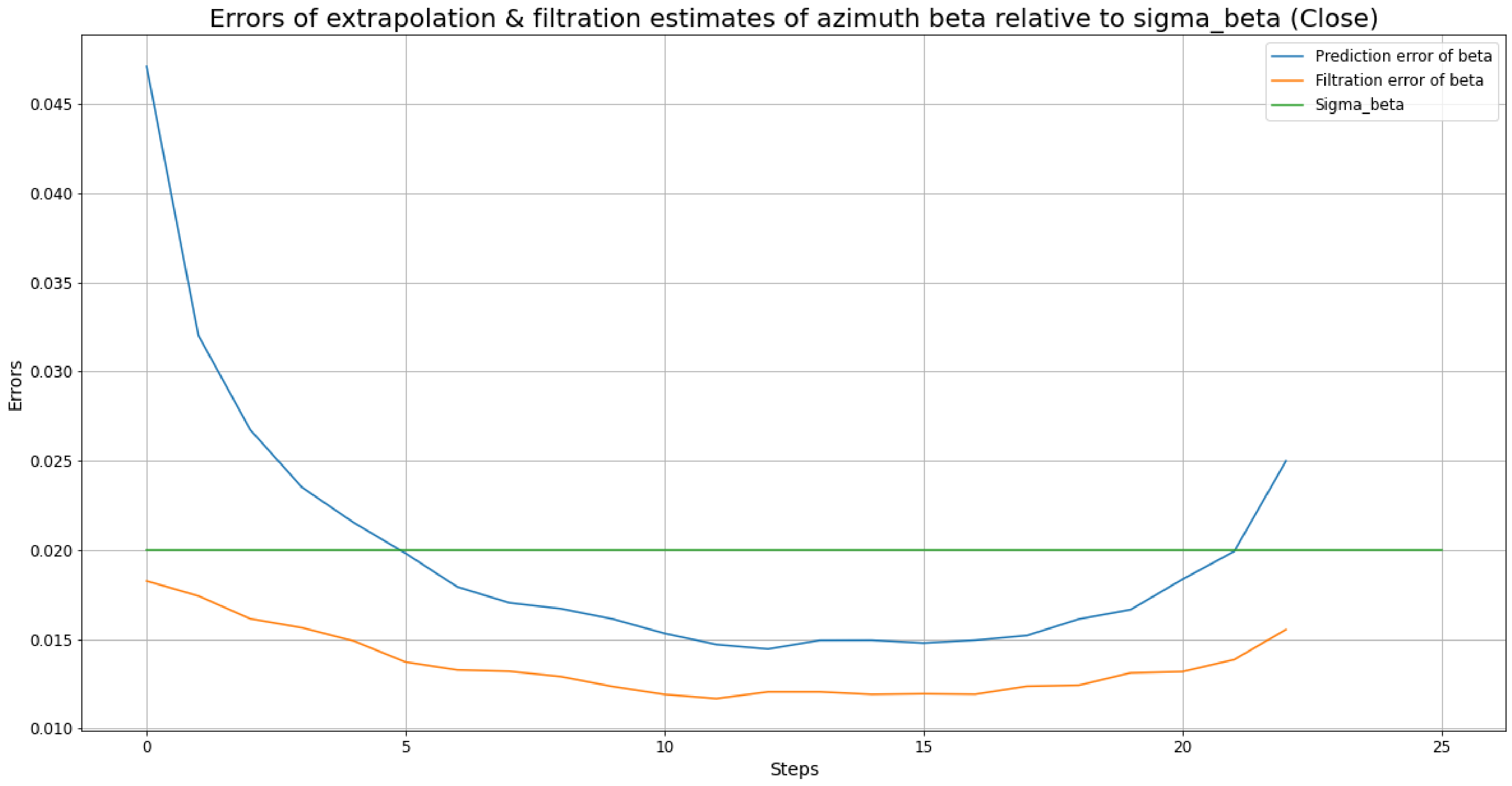
In [24]:
```
#Question 15
#Calculate true errors of estimation
sg_D = np.full((n), 20)
fig, f = plt.subplots(figsize=(20,10))
f.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D (Close)", fontsize = 20)
```

```
f.set_xlabel("Steps", fontsize = 14)
f.set_ylabel("Errors", fontsize = 14)
f.plot(Final_err_pred2[0, 3:], label = "Prediction error of D")
f.plot(Final_err_filt2[0, 3:], label = "Filtration error of D")
f.plot(sg_D, label = "Sigma_D")
f.tick_params(labelsize = 12)
f.legend(fontsize = 12)
f.grid()
plt.savefig('D_error(close).jpg')
```



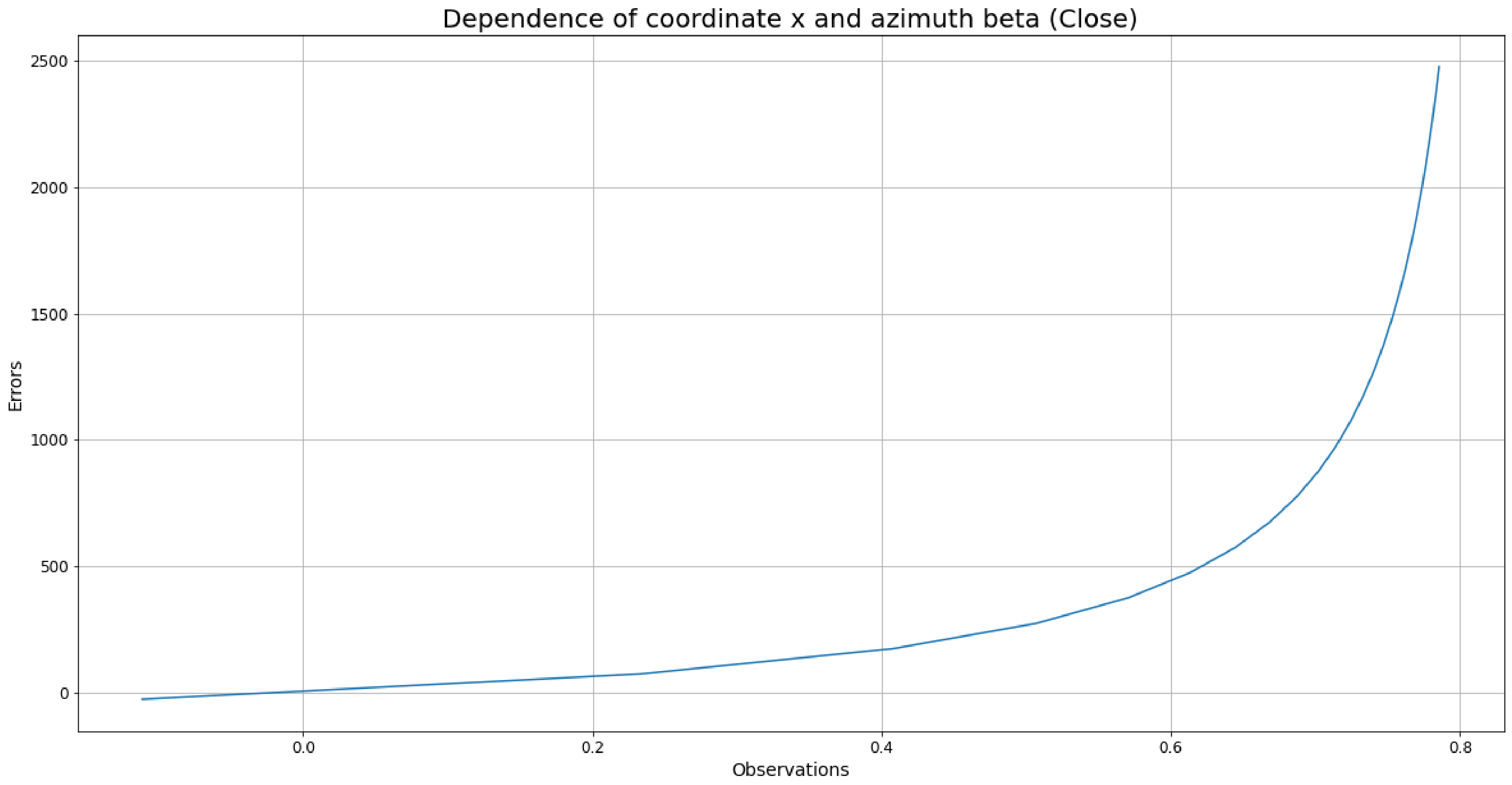
In [25]:
```
sg_b = np.full((n), 0.02)
fig, g = plt.subplots(figsize=(20,10))
g.set_title("Errors of extrapolation & filtration estimates of azimuth beta relative to sigma_beta (Close)", fontsize = 20)
```

```
g.set_xlabel("Steps", fontsize = 14)
g.set_ylabel("Errors", fontsize = 14)
g.plot(Final_err_pred2[1, 3:], label = "Prediction error of beta")
g.plot(Final_err_filt2[1, 3:], label = "Filtration error of beta")
g.plot(sg_b, label = "Sigma_beta")
g.tick_params(labelsize = 12)
g.legend(fontsize = 12)
g.grid()
plt.savefig('b_error(close).jpg')
```



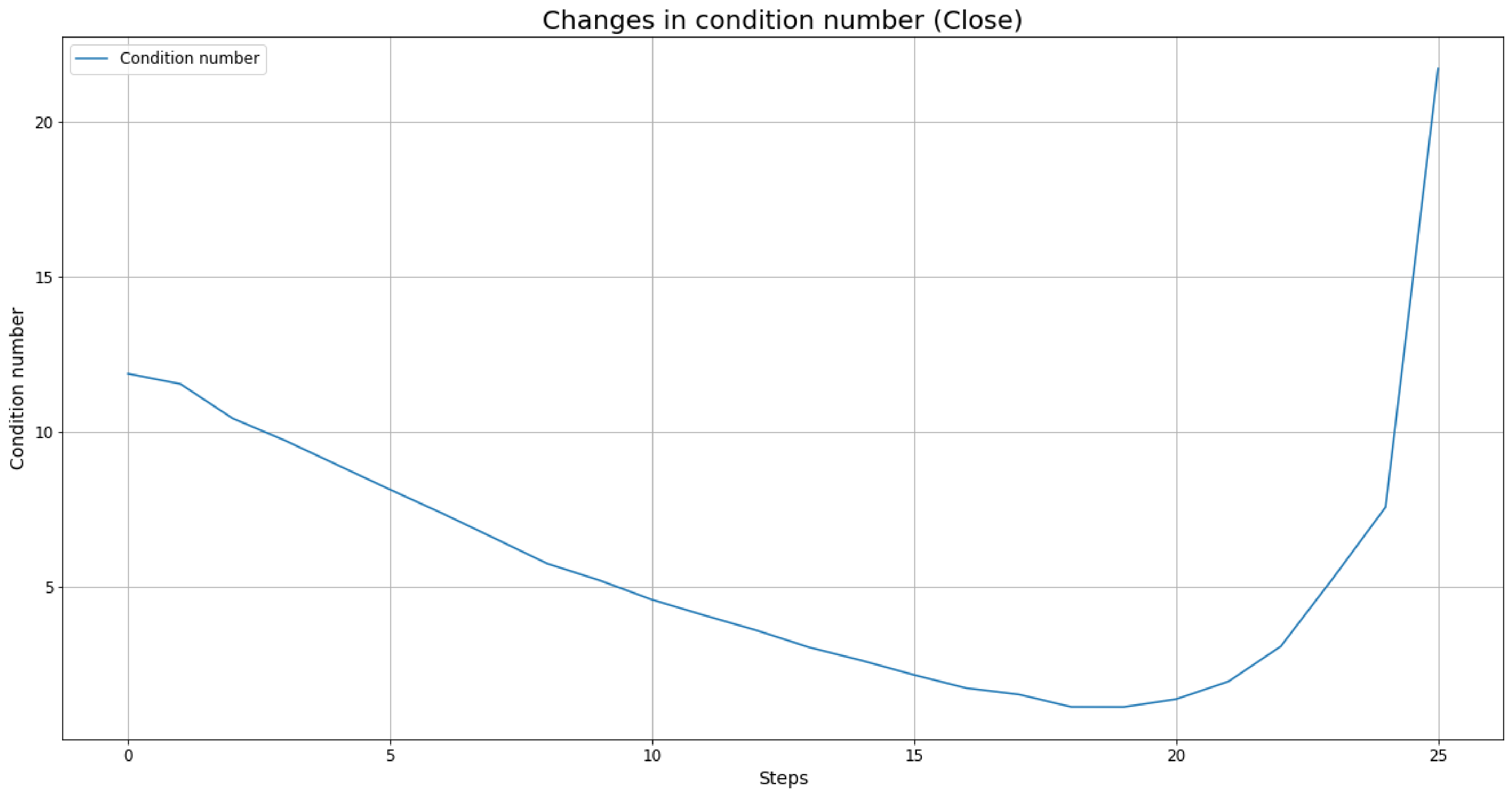
In [26]:
```
#Question 16
#Analyze dependence of coordinate x on azimuth beta by plotting
fig, h = plt.subplots(figsize=(20,10))
```

```
h.set_title("Dependence of coordinate x and azimuth beta (Close)", fontsize = 20)
h.set_xlabel("Observations", fontsize = 14)
h.set_ylabel("Errors", fontsize = 14)
h.plot(p_traj2[1, :], c_traj2[0, :], label='Dependence of coordinate x on azimuth beta - true data')
#h.plot(pf[0][3, :], c_traj[0, :], label='Dependence of coordinate x on azimuth beta - filtered')
h.tick_params(labelsize = 12)
h.grid()
plt.savefig('x_b(close).jpg')
```



In [27]:
```
#Question 17
fig, i = plt.subplots(figsize=(20,10))
i.set_title("Changes in condition number (Close)", fontsize = 20)
i.set_xlabel("Steps", fontsize = 14)
```

```
i.set_ylabel("Condition number", fontsize = 14)
i.plot(cond_num2, label = "Condition number")
i.tick_params(labelsize = 12)
i.legend(fontsize = 12)
i.grid()
plt.savefig('cond_n(close).jpg')
```

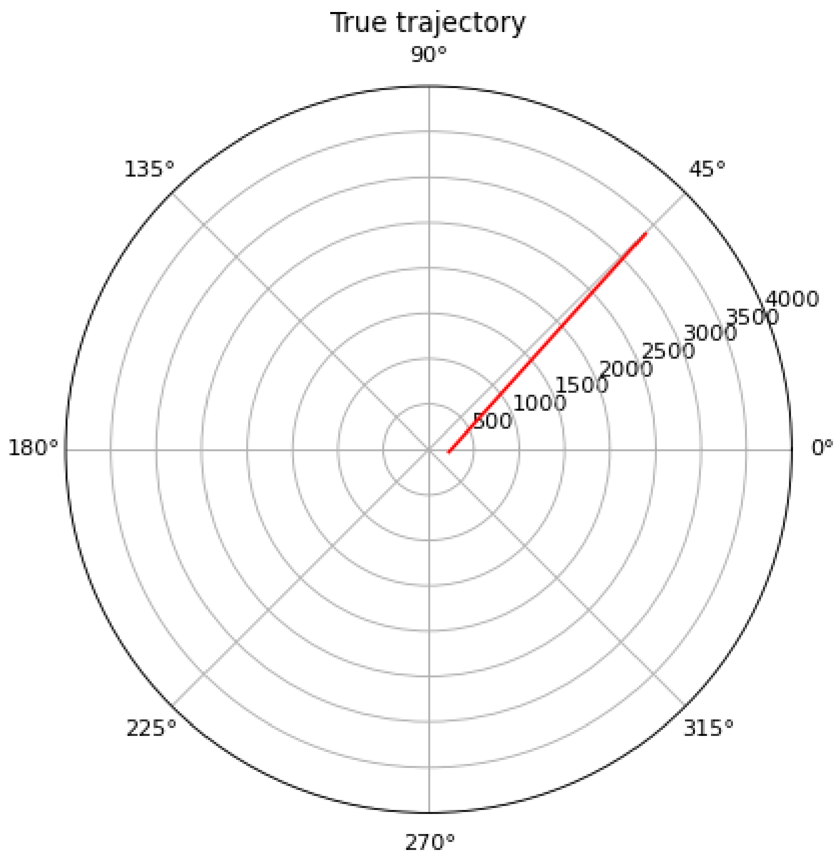


In [28]:
```
#Question 19 - 20
sigma_D3 = 50
sigma_b3 = 0.0015
c_traj3 = true_c_trajectory(xy0 = np.ones(2) * 3500 / np.sqrt(2), v0 = np.array([-50, -45]))
p_traj3 = true_p_trajectory(c_traj3)
zm3 = measurements(p_traj3, sigma_D3, sigma_b3)
```

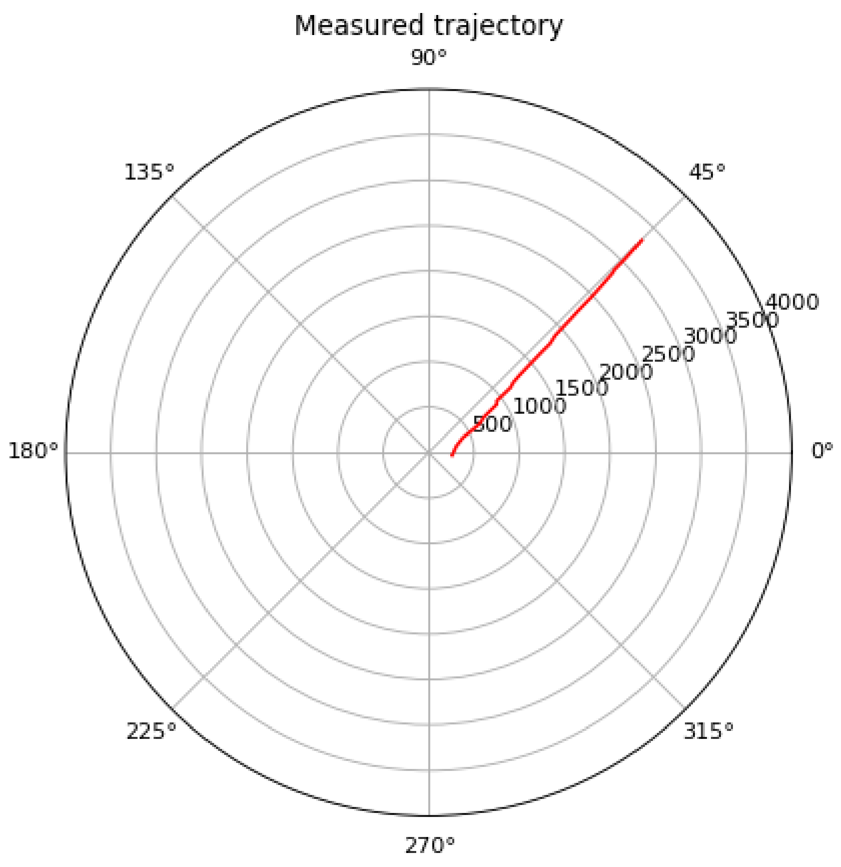
```
zc3 = pseudo_measurements(zm3)
R3 = create_R(zm3, sigma_D3, sigma_b3)
pf3 = Kalman_filter(zc3, R3)[0]
cond_num3 = condition_number(sigma_D3, sigma_b3, zm3)

def Errors3(M):
    error_pred3 = np.zeros((2, n, M))
    error_filt3 = np.zeros((2, n, M))

    for i in range(M):
        c_traj3 = true_c_trajectory(xy0 = np.ones(2) * 3500 / np.sqrt(2), v0 = np.array([-50, -45]))
        p_traj3 = true_p_trajectory(c_traj3)
        zm3 = measurements(p_traj3, sigma_D3, sigma_b3)
        zc3 = pseudo_measurements(zm3)
        cond_num3 = condition_number(sigma_D3, sigma_b3, zm3)
        pf3 = Kalman_filter(zc3, R3)[0]

        error_pred3[:, :, i] = (p_traj3 - pf3[:2, :]) ** 2
        error_filt3[:, :, i] = (p_traj3 - pf3[2:, :]) ** 2

    Final_err_pred3 = np.sqrt(np.sum(error_pred3, axis = 2) / (M - 1))
    Final_err_filt3 = np.sqrt(np.sum(error_filt3, axis = 2) / (M - 1))

    return Final_err_pred3, Final_err_filt3

Final_err_pred3, Final_err_filt3 = Errors3(500)
```

In [29]:

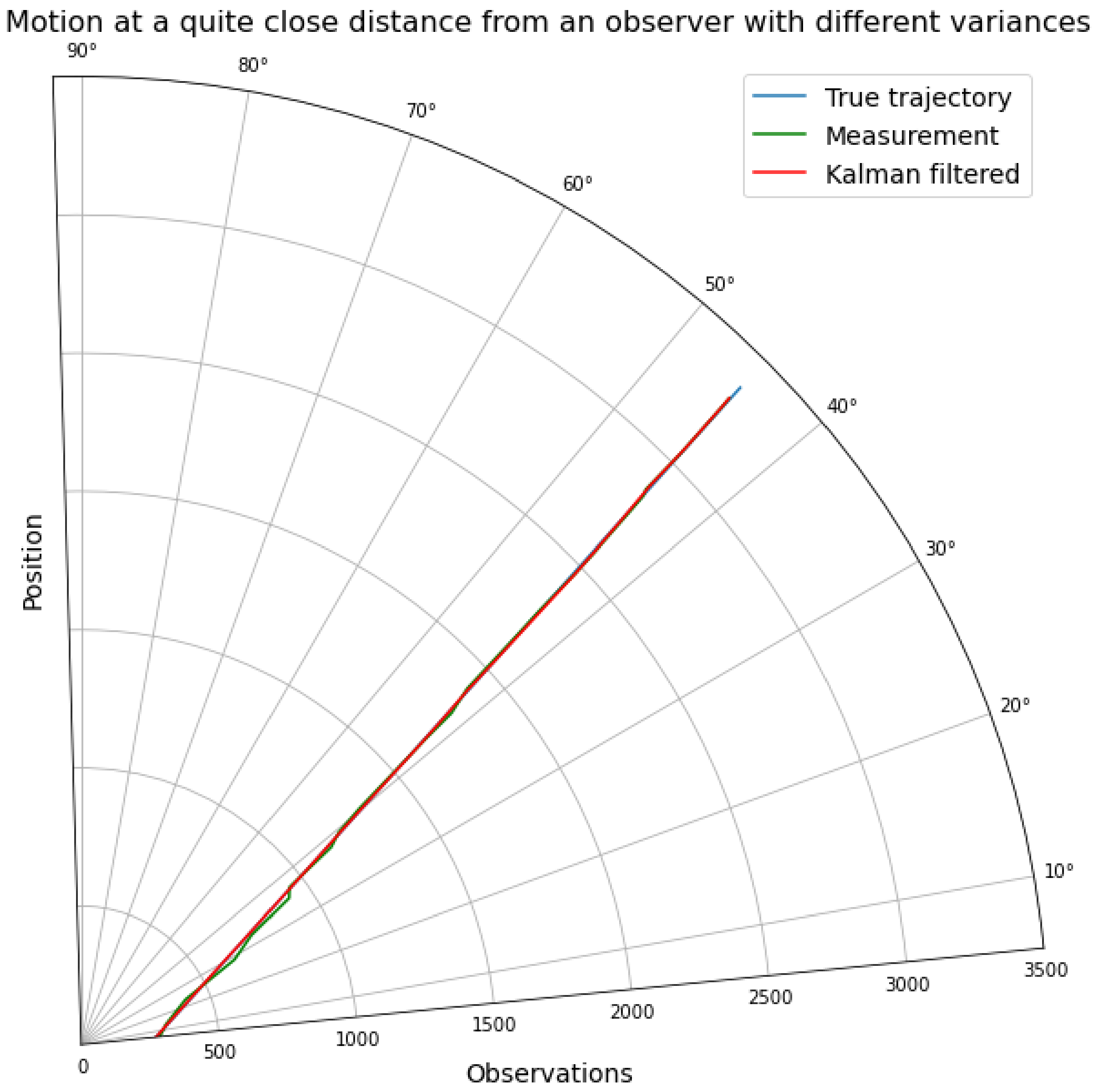
```
# Plot
plt.figure(figsize=(6, 6), dpi=80)
plt.polar(p_traj3[1, 1:], p_traj3[0, 1:], c='red')
plt.title('True trajectory')
plt.ylim((0, 4000))
plt.savefig('True_trajectory(close_diff).jpg')
```



```
In [30]: # Plot
         plt.figure(figsize=(6, 6), dpi=80)
         plt.polar(zm3[1, 1:], zm3[0, 1:], c='red')
         plt.title('Measured trajectory')
         plt.ylim((0, 4000))
         plt.savefig('Measured_trajectory(close_diff).jpg')
```

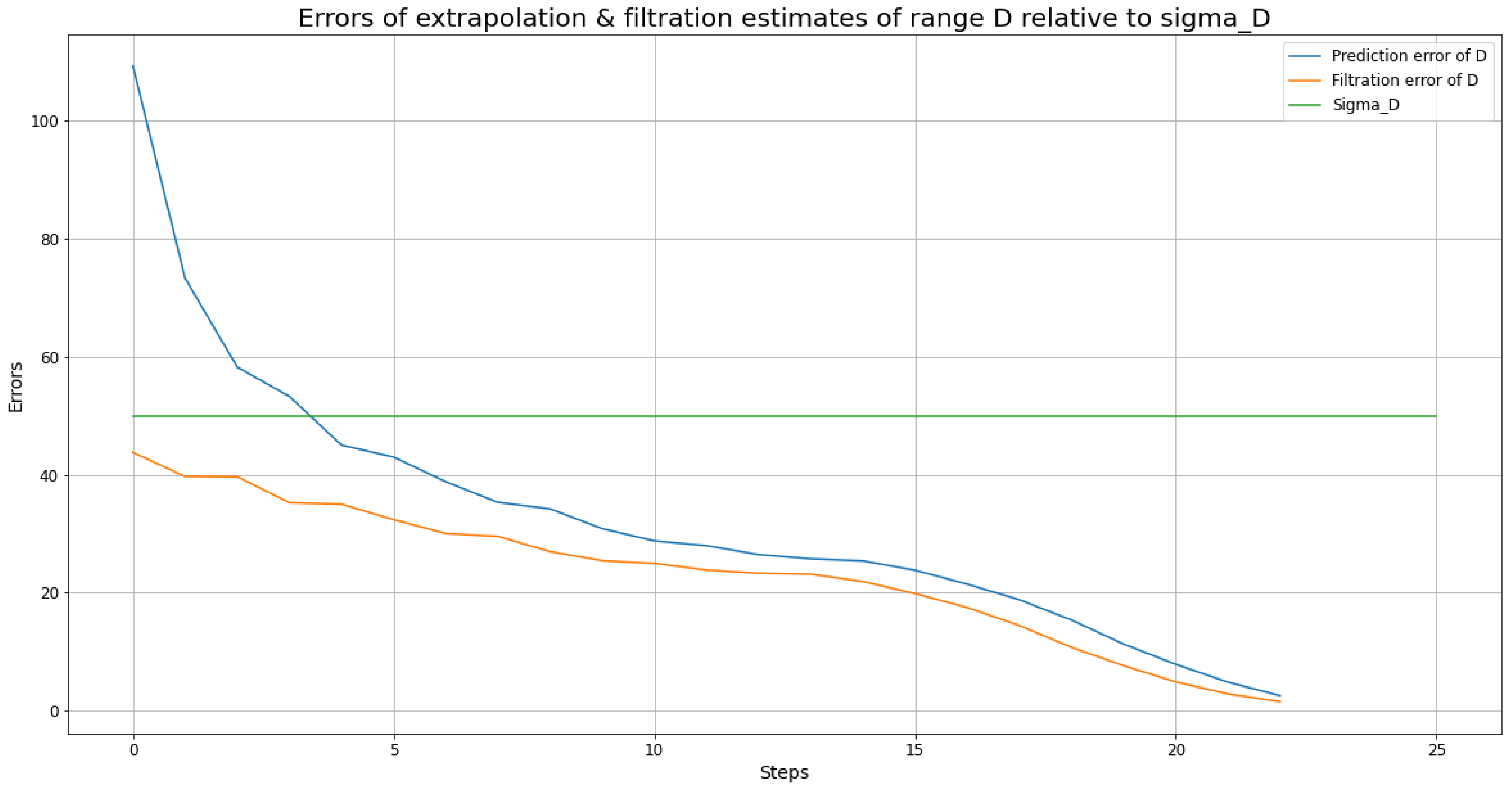


In [31]:
```
# Plot
fig, az = plt.subplots(figsize=(10, 10), subplot_kw={'projection': 'polar'})
az.set_title("Motion at a quite close distance from an observer with different variances", fontsize = 16)
az.set_ylabel("Position", fontsize = 14)
az.set_xlabel("Observations", fontsize = 14)
az.plot(p_traj3[1, 1:], p_traj3[0, 1:], label='True trajectory')
az.plot(zm3[1, 1:], zm3[0, 1:], label='Measurement', c='green')
az.plot(pf3[3, 1:], pf3[2, 1:], label='Kalman filtered', c='red')
plt.xlim(0.1, 1.6)
plt.ylim((0, 3500))
plt.legend(fontsize = 14,loc = 'best')
plt.savefig('Kalman(close_diff).jpg')
```



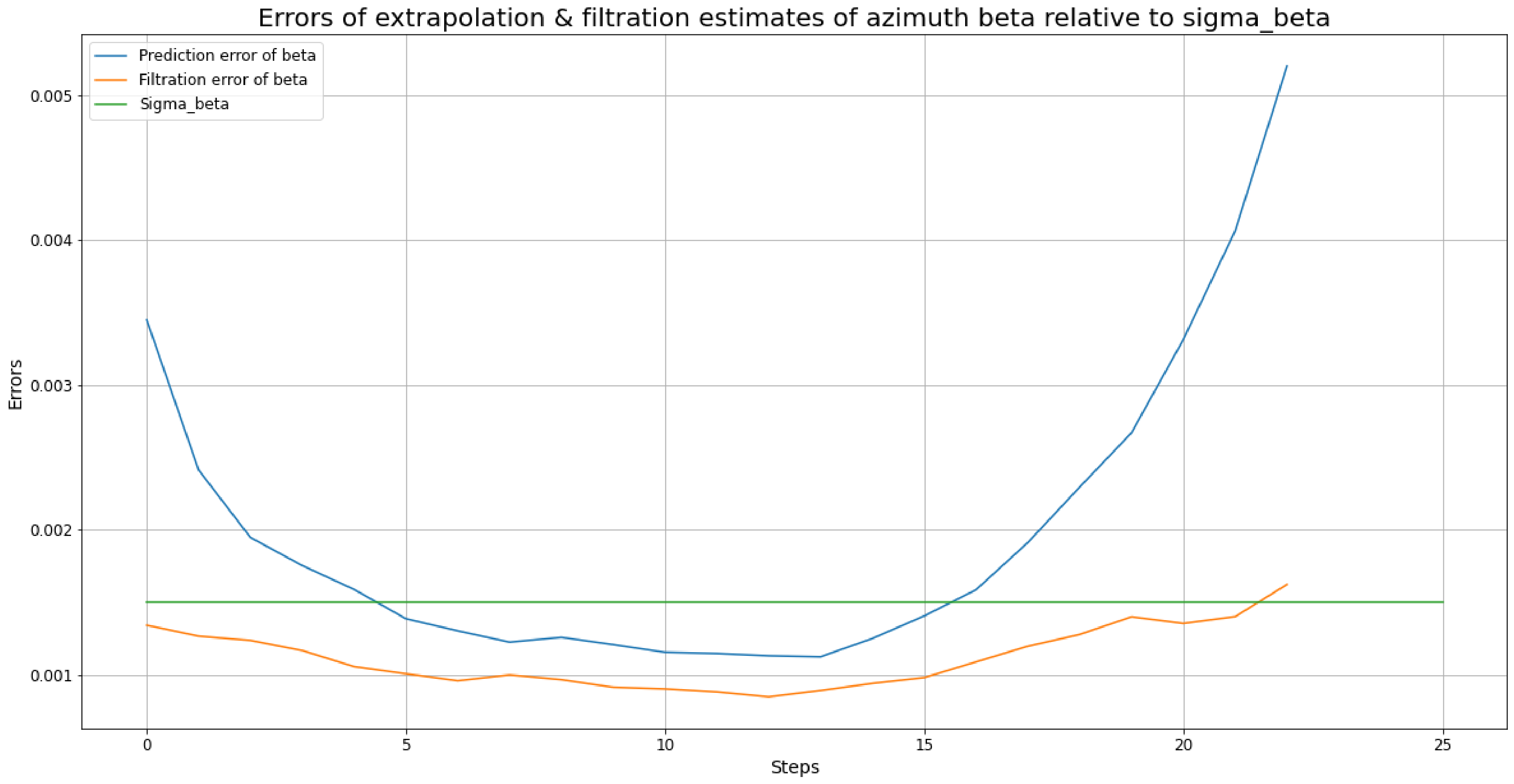
In [32]:
```
sg_D = np.full((n), 50)
fig, k = plt.subplots(figsize=(20,10))
k.set_title("Errors of extrapolation & filtration estimates of range D relative to sigma_D", fontsize = 20)
k.set_xlabel("Steps", fontsize = 14)
k.set_ylabel("Errors", fontsize = 14)
```

```
k.plot(Final_err_pred3[0, 3:], label = "Prediction error of D")
k.plot(Final_err_filt3[0, 3:], label = "Filtration error of D")
k.plot(sg_D, label = "Sigma_D")
k.tick_params(labelsize = 12)
k.legend(fontsize = 12)
k.grid()
plt.savefig('D_error(close_diff).jpg')
```



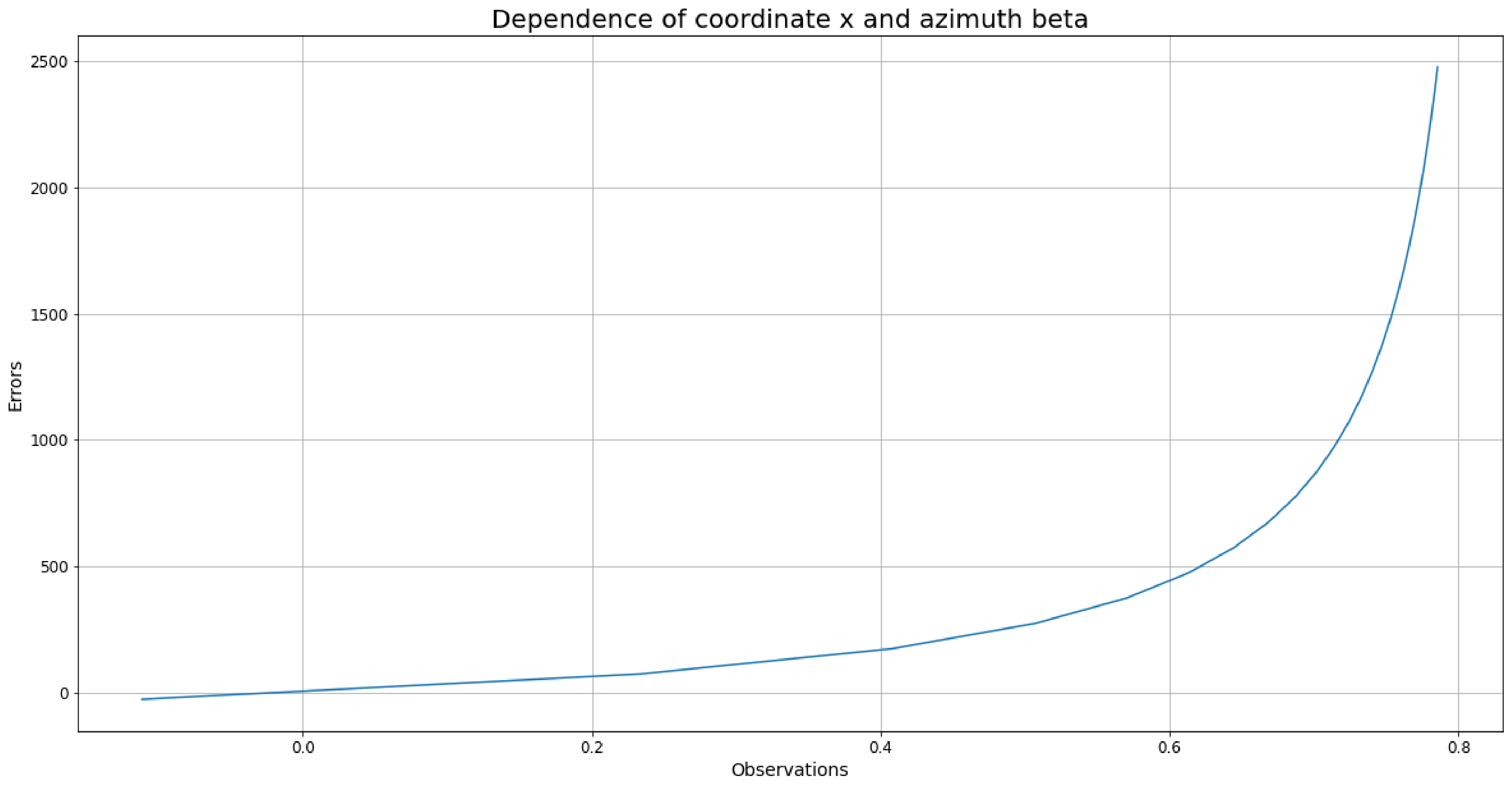
In [33]:
```
sg_b = np.full((n), 0.0015)
fig, l = plt.subplots(figsize=(20,10))
l.set_title("Errors of extrapolation & filtration estimates of azimuth beta relative to sigma_beta", fontsize = 20)
l.set_xlabel("Steps", fontsize = 14)
l.set_ylabel("Errors", fontsize = 14)
```

```
l.plot(Final_err_pred3[1, 3:], label = "Prediction error of beta")
l.plot(Final_err_filt3[1, 3:], label = "Filtration error of beta")
l.plot(sg_b, label = "Sigma_beta")
l.tick_params(labelsize = 12)
l.legend(fontsize = 12)
l.grid()
plt.savefig('b_error(close_diff).jpg')
```



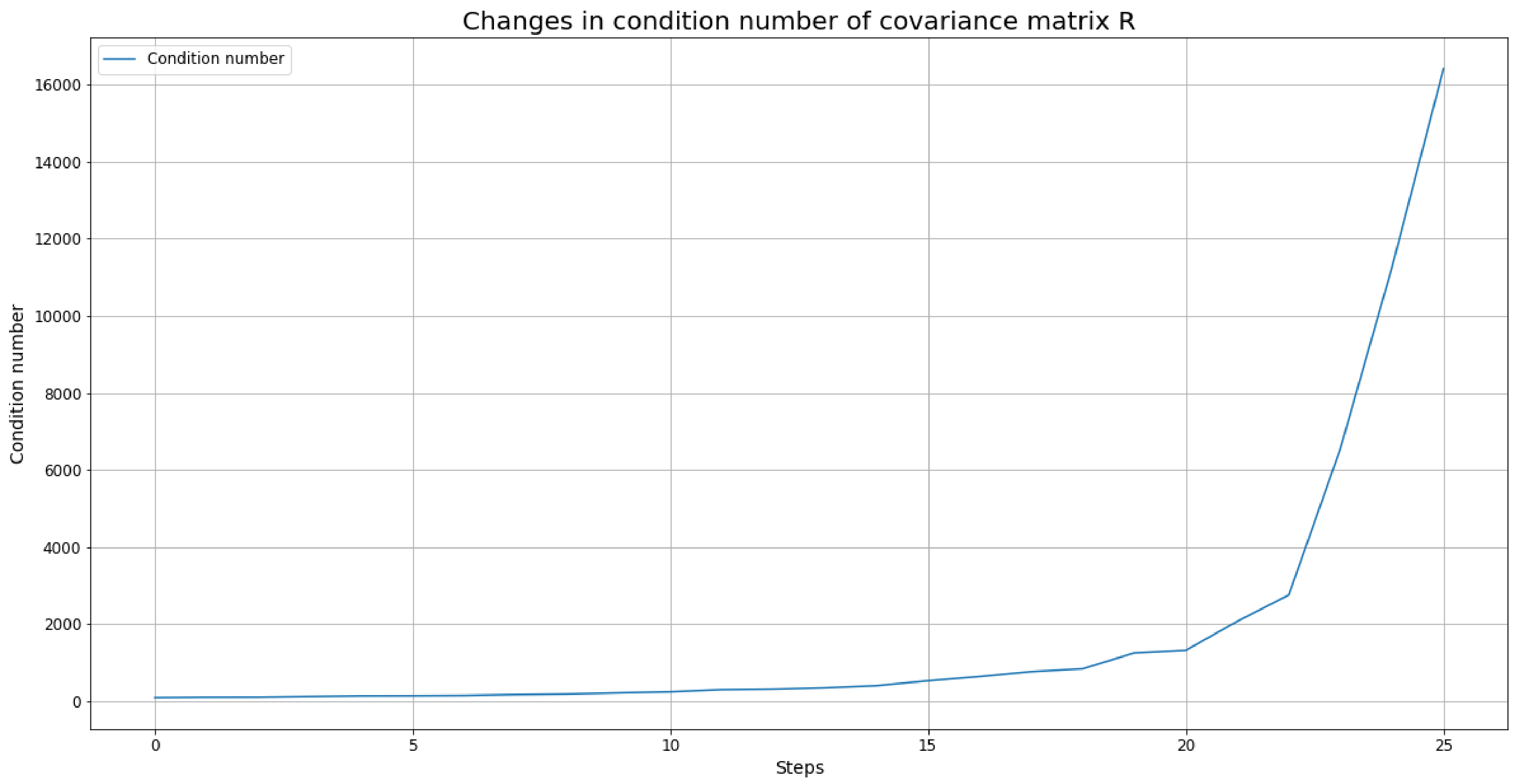
In [34]:
```
fig, m = plt.subplots(figsize=(20,10))
m.set_title("Dependence of coordinate x and azimuth beta", fontsize = 20)
m.set_xlabel("Observations", fontsize = 14)
m.set_ylabel("Errors", fontsize = 14)
m.plot(p_traj3[1, :], c_traj3[0, :], label='Dependence of coordinate x on azimuth beta - true data')
```

```
#m.plot(pf[0][3, :], c_traj[0, :], label='Dependence of coordinate x on azimuth beta - filtered')
m.tick_params(labelsize = 12)
m.grid()
plt.savefig('x_b(close_diff).jpg')
```



In [35]:
```
fig, n = plt.subplots(figsize=(20,10))
n.set_title("Changes in condition number of covariance matrix R", fontsize = 20)
n.set_xlabel("Steps", fontsize = 14)
n.set_ylabel("Condition number", fontsize = 14)
n.plot(cond_num3, label = "Condition number")
n.tick_params(labelsize = 12)
n.legend(fontsize = 12)
```

```
n.grid()
plt.savefig('cond_num(close_diff).jpg')
```


Changes in condition number of covariance matrix R
Condition number
Condition number
Steps