Skoltech

Skolkovo Institute of Science and Technology

Tracking and forecasting in conditions of measurements gaps

Experimental Data Processing

Assignment №7

Team №10

Submitted by:
Yunseok Park
Ilya Novikov
Ruslan Kalimullin

Instructor:
Tatiana Podladchikova

MA060238 (Term 1B, 2022-2023)

October 5th, 2022

Contents

1 Introduction

The objective of this laboratory work is to develop the estimation and tracking algorithm in conditions of measurement gaps, that is of prime importance for many practical control and forecasting problems. This will bring about a deeper understanding of main difficulties of practical Kalman filter implementation and skills to overcome these difficulties to get optimal assimilation output.

2 Work progress

Question 1 - 2

Generation of a true trajectory X_i of an object motion disturbed by normally distributed unbiased random acceleration a_i with variance $\sigma_\alpha^2 = 0.2^2$ (Equation 1). Measurements of coordinate x_i are performed every second with variance of measurement noise $\sigma_\eta^2 = 20^2$. Observation interval is 200 seconds with probability gap $P = 0.2$. Develop Kalman filter for tracking moving object.

$$\begin{aligned} x_i &= x_{i-1} + V_{i-1}T + \frac{a_{i-1}T^2}{2} \\ V_i &= V_{i-1} + a_{i-1}T \end{aligned} \tag{1}$$

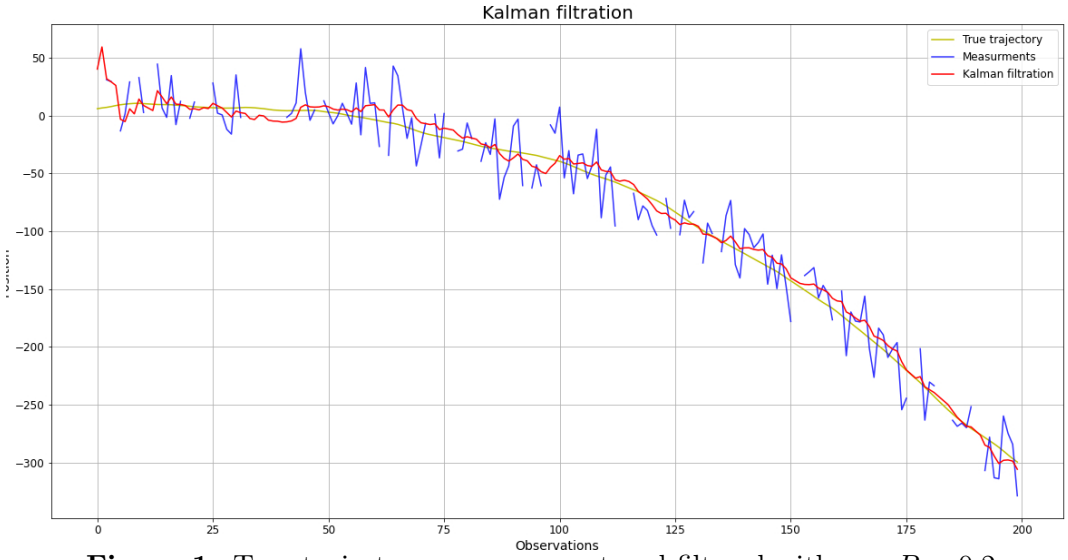


Figure 1: True trajectory, measurement and filtered with gap $P = 0.2$

In this part, first measurements with gaps are created with probability of measurement gap P is 0.2. The measurement parts where there is no data is introduced as "NaN" and is shown as a blank in the graph. After that, Kalman filtered is developed accordingly and the parts where there is no data is introduced as the value does not change. In Figure 1 true trajectory, measurement and Kalman filter with gap P=0.2 is shown. It can be seen that filtered estimates gets closer to the true trajectory whenever there is a data and if we look at the whole trajectory, we can say that it is a good estimate even though we have some unknown data. Even now, it can be seen that if the probability gap P increased, we would have less accurate results and we will see that in part 4 with different P values.

Question 3

In this part, comparison of Kalman filtered, 1 and 7 step extrapolated errors is done. It can be seen from Figure 2 that all 3 errors have the same trend; however, it can be seen both visually that Kalman filtered measurement gives less error and it is closer to the true value.

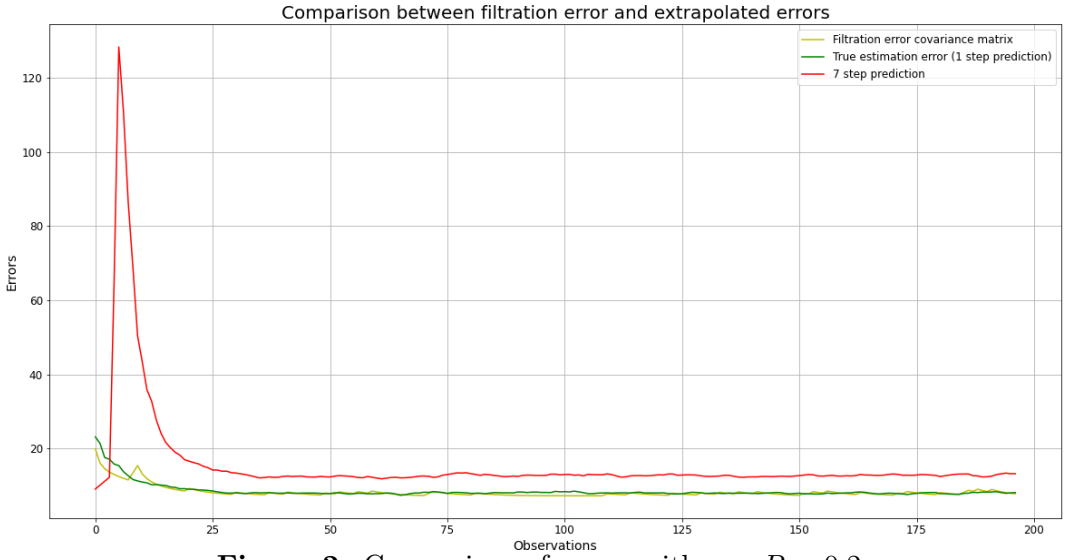


Figure 2: Comaprison of errors with gap $P = 0.2$

Question 4

Here, we analyzed the decrease of estimation accuracy in conditions of measurement gaps. We compared results when probability of measurement gap is:

1. $P = 0.3$
2. $P = 0.5$
3. $P = 0.7$

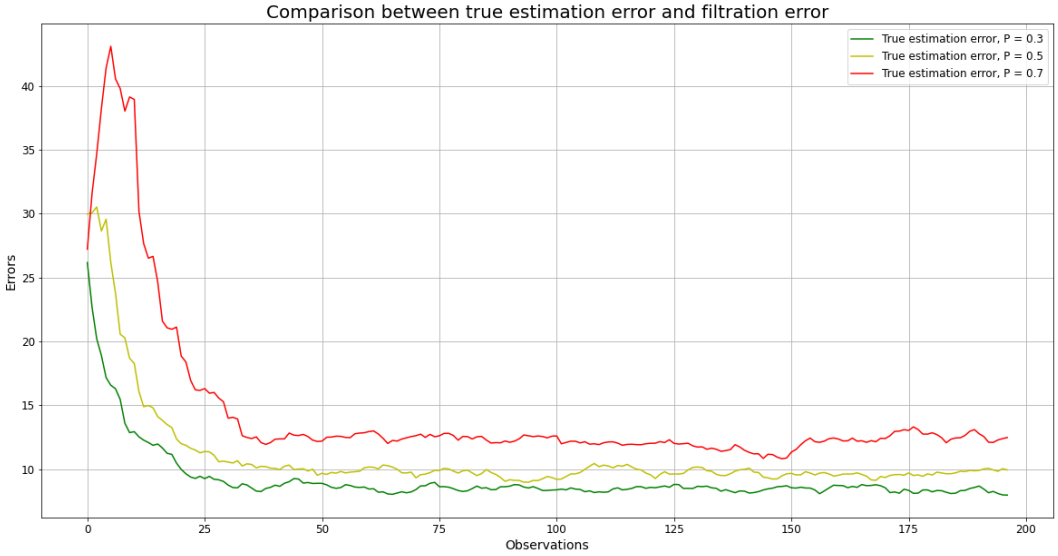


Figure 3: Comparison of true errors with different P

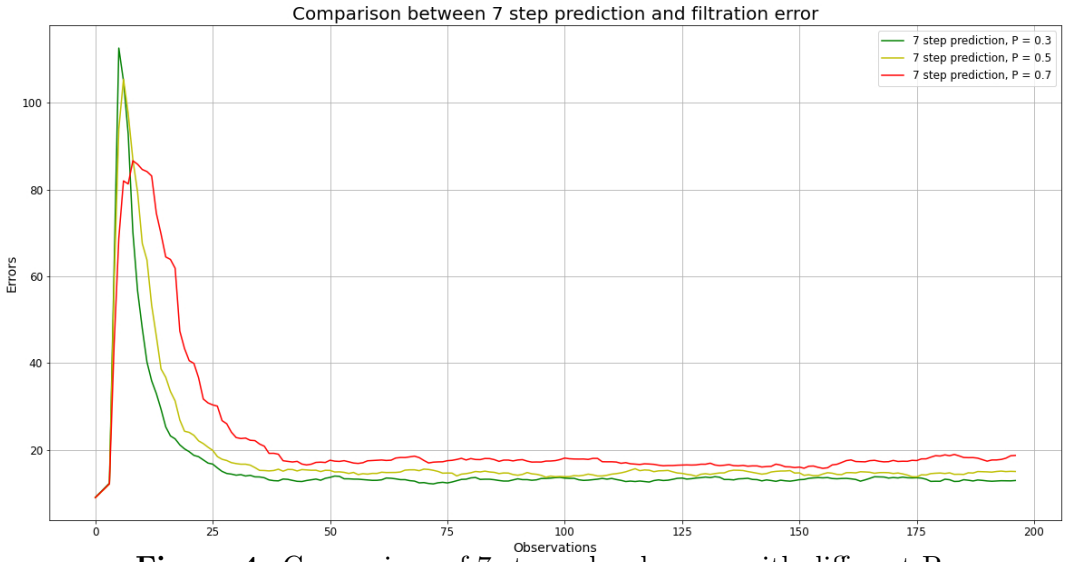


Figure 4: Comparison of 7 steps ahead errors with different P

In this part different probability of measurement gaps are tried and they are 0.3, 0.5 and 0.7 respectively. We checked the final errors between these 3 estimation at different P values. Firstly, All the filtration errors at different P values are shown below in Figure. They stabilize at a smaller value in the case of lower P values. At All P values, they also follow the same trend; however, when P value increase, there is a bigger jump at the start and they stabilize at a larger value of error. So among all P values, P=0.3 is the one that gives the least errors.

3 Conclusion

What we have learnt and tried:

1. How to generate measurements with the probability of measurements gap P.
2. How to develop a Kalman filter estimation even when we have some unknown data.
3. Importance of the probability of measurement gap P
4. Comparison of filtered and extrapolated errors

What we have reflected upon:

1. Tried different measurement gaps and saw that when P increases, there is a decrease of estimation accuracy
2. Implemented a new algorithm for the Kalman filter with measurement gaps
3. Compared the errors and decided that Kalman filter is better than the extrapolated estimations in this case.

Contribution of each members:

1. Ilya: wrote the main code for Kalman filter.
2. Ruslan: wrote the code, made plots.
3. Yunseok: wrote the code and report

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

In [2]:
```
#Question 1-2
T = 1

sigma_a = 0.2 ** 2
size = 200
#Initial state vector
X_true_0 = np.array([[5], [1]])
#State vector
X_0 = np.zeros((2, size))
#Measurments
z = np.zeros((size, 1))
sigma_eta = 20 ** 2
#Transition matrix
Phi = np.array([[1, T], [0, 1]])
#Input matrix
G = np.array([[T ** 2 / 2], [T]])
#Observation matrix
H = np.array([[1, 0]])

P = 0.2

def true_trajectory_(X_0_1):
    a_i = np.random.normal(0, np.sqrt(sigma_a))
    X_0_ = Phi.dot(X_0_1) + G.dot(a_i)
    return X_0_

#True trajectory and measurments
def measurments (P, size):
    X_0_ = np.zeros((2, size))
    z_ = np.zeros((size, 1))


    for n in range(size):
        if n == 0:
            X_0_[:, n] = true_trajectory_(X_true_0).reshape(-1)
            if random.random() >= P:
                z_[n] = H.dot(X_0_[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
```

```
            else:
                z_[n] = np.nan
        else:
            X_0_[:, n] = true_trajectory_(X_0_[:, n-1].reshape(2, 1)).reshape(-1)
            if random.random() >= P:
                z_[n] = H.dot(X_0_[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
            else:
                z_[n] = np.nan
    return X_0_, z_

X_0, z = measurments(P, size)
```

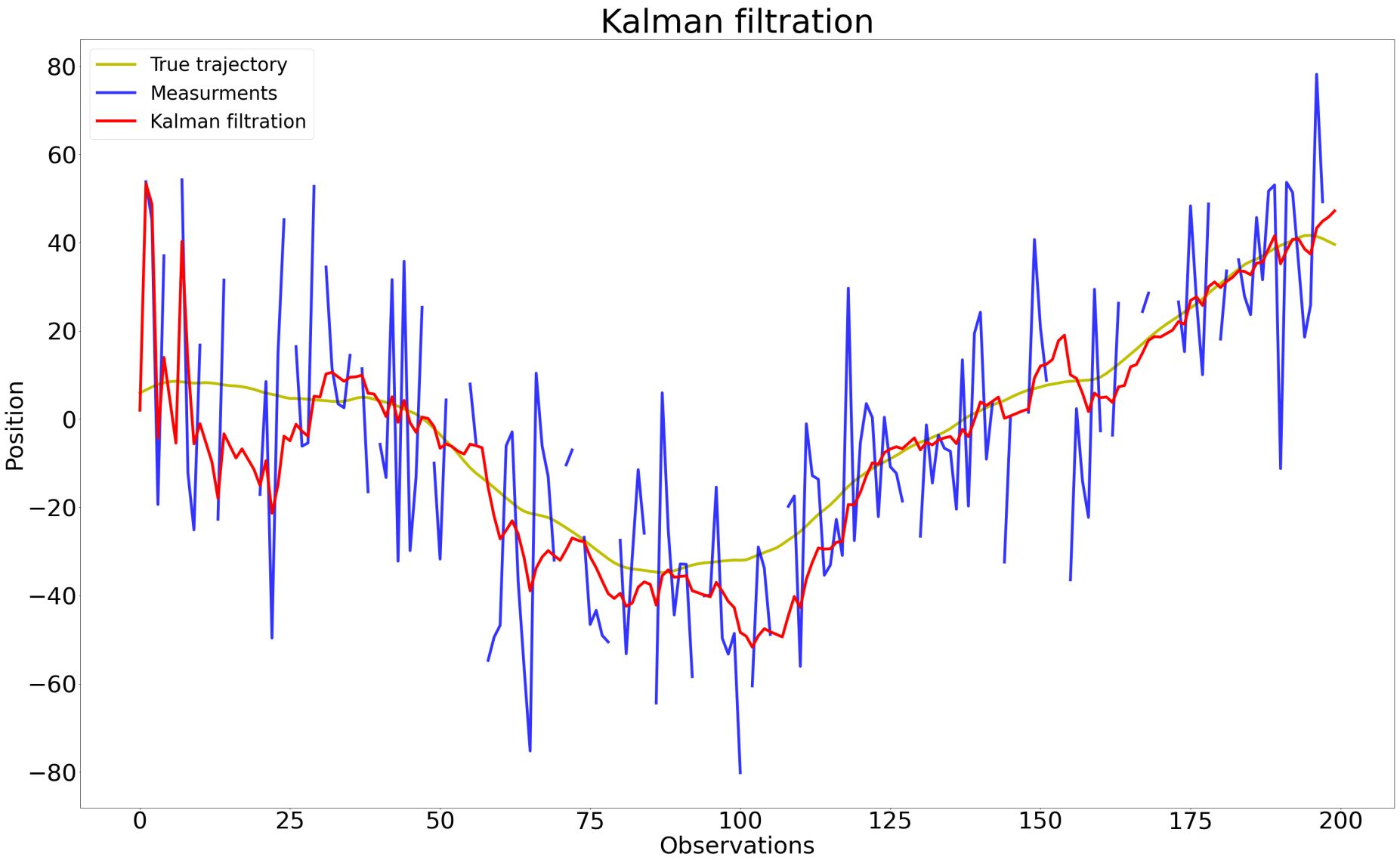
In [3]:

```
def Kalman(X_init, P_init, sigma_a_, sigma_eta_, size_, z_):
    #Filterd position
    X_ = np.zeros((2, size_))
    P_ = np.zeros((2, 2, size_))
    K_ = np.zeros((2, size_))

    Q_ = G.dot(G.T).dot(sigma_a_)
    R_ = sigma_eta_
    I = np.eye(2)

    for n in range(size_):
        if n == 0:
            if np.isnan(z[n]) == True:
                X_i_i_1 = Phi.dot(X_init.reshape(2, 1)).reshape(-1)
                P_i_i_1 = (Phi.dot(P_init)).dot(Phi.T) + Q_

                X_[:, n] = X_i_i_1
                P_[:, :, n] = P_i_i_1
                K_[:, n] = np.array([np.nan, np.nan])
            else:
                X_i_i_1 = Phi.dot(X_init.reshape(2, 1)).reshape(-1)
                P_i_i_1 = (Phi.dot(P_init)).dot(Phi.T) + Q_

                K_i = (P_i_i_1.dot(H.T) / (H.dot(P_i_i_1).dot(H.T) + R_)).reshape(-1)
                P_i_i = (I - K_i.reshape(2, 1).dot(H.reshape(1, 2))).dot(P_i_i_1)
                X_i_i = X_i_i_1 + K_i * (z_[n] - H.dot(X_i_i_1))
                X_[:, n] = X_i_i
                P_[:, :, n] = P_i_i
                K_[:, n] = K_i
        else:
            if np.isnan(z[n]) == True:
```

```
                X_i_i_1 = Phi.dot(X_[:, n-1].reshape(2, 1)).reshape(-1)
                P_i_i_1 = (Phi.dot(P_[:, :, n-1])).dot(Phi.T) + Q_

                X_[:, n] = X_i_i_1
                P_[:, :, n] = P_i_i_1
                K_[:, n] = np.array([np.nan, np.nan])
            else:
                X_i_i_1 = Phi.dot(X_[:, n-1].reshape(2, 1)).reshape(-1)
                P_i_i_1 = (Phi.dot(P_[:, :, n-1])).dot(Phi.T) + Q_

                K_i = (P_i_i_1.dot(H.T) / (H.dot(P_i_i_1).dot(H.T) + R_)).reshape(-1)
                P_i_i = (I - K_i.reshape(2, 1).dot(H.reshape(1, 2))).dot(P_i_i_1)
                X_i_i = X_i_i_1 + K_i * (z_[n] - H.dot(X_i_i_1))
                X_[:, n] = X_i_i
                P_[:, :, n] = P_i_i
                K_[:, n] = K_i

    return X_, P_, K_
```

In [4]:

```
#Initial filtred state vector
X_0_4 = np.array(([2], [0]))
P_0_4 = np.array([[10000, 0], [0, 10000]])
M = 500

X_2 = Kalman(X_0_4, P_0_4, sigma_a, sigma_eta, size, z)[0]

fig, ax = plt.subplots(figsize=(50,30))
ax.set_title("Kalman filtration", fontsize = 70)
ax.set_ylabel("Position", fontsize = 50)
ax.set_xlabel("Observations", fontsize = 50)
ax.plot(X_0[0, :], linewidth=6, label = "True trajectory", c = 'y')
ax.plot(z, 'b', alpha = 0.8, linewidth = 6, label = "Measurments")
ax.plot(X_2[0, :], 'r', linewidth = 6, label = "Kalman filtration")

ax.tick_params(labelsize = 50)
ax.legend(fontsize = 40)

plt.show()
```

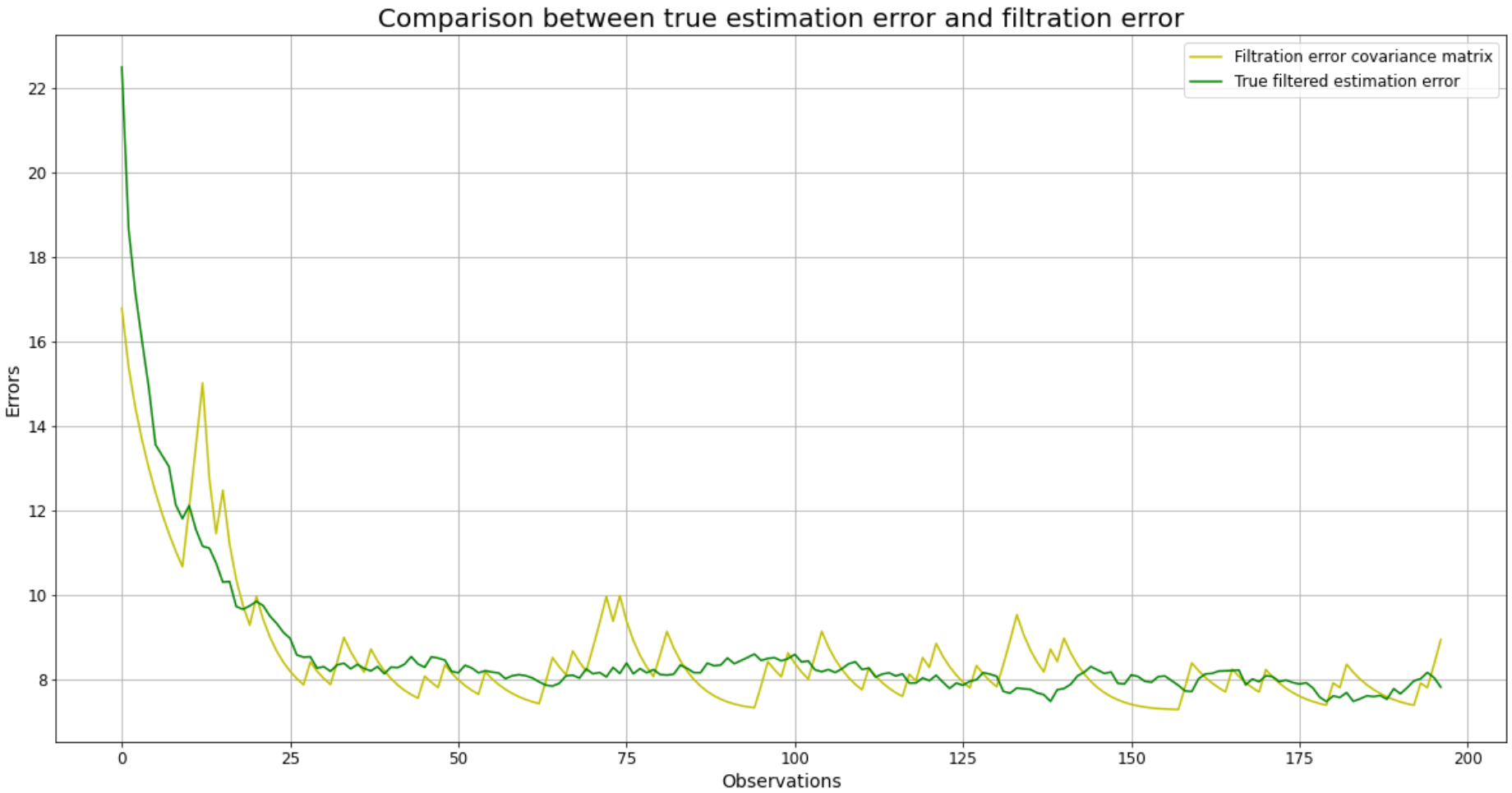


In [5]:
```
#Question 3. Kalman errors (500 iterations)
mistake_ = np.zeros((2, size, M))
mistake_7_ = np.zeros((2, size, M))

for m in range (M):
    P_ = np.zeros((2, 2, size))
```

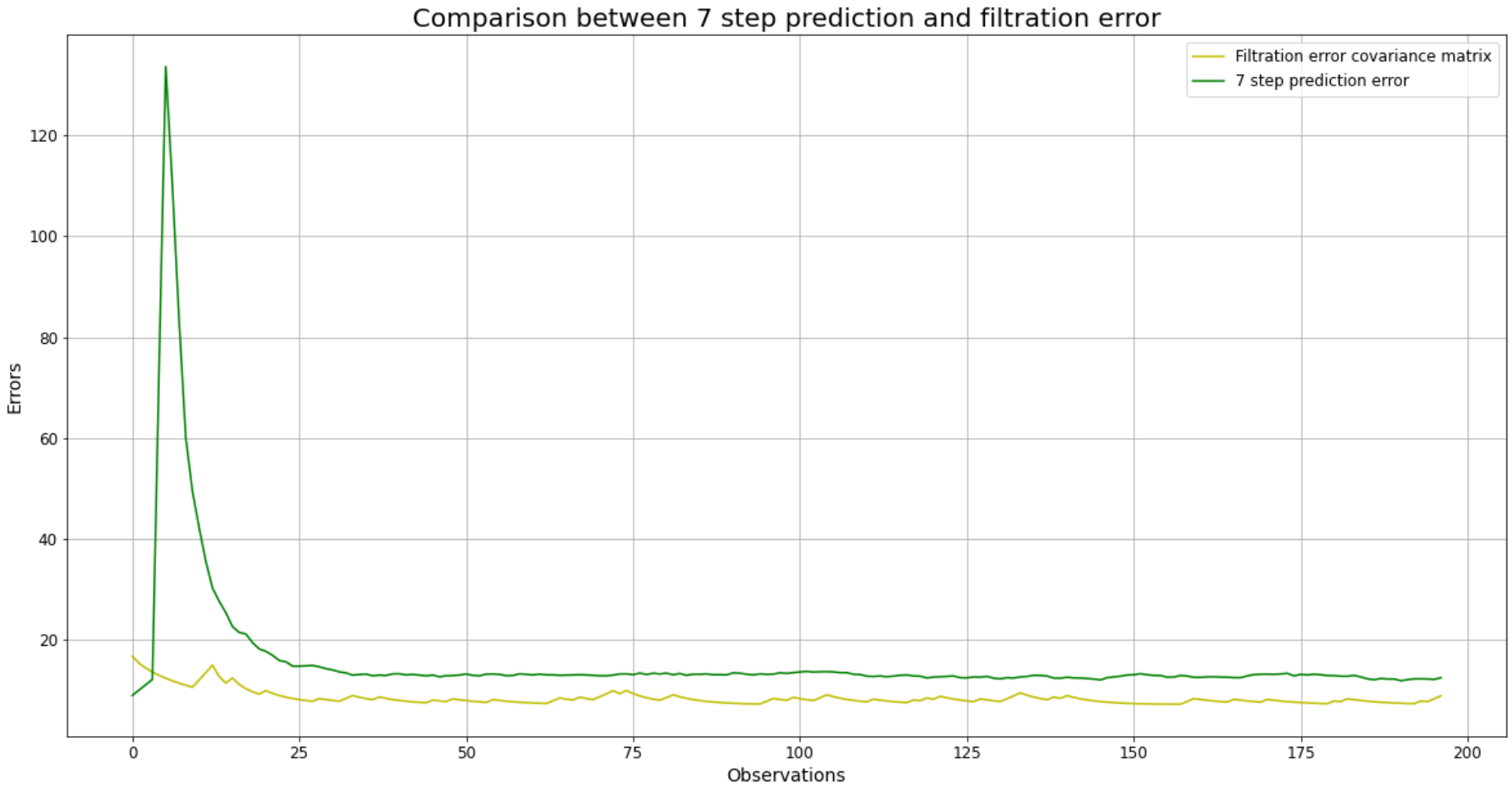
```
    for n in range(size):
        if n == 0:
            X_0[:, n] = true_trajectory_(X_true_0).reshape(-1)
            if random.random() >= P:
                z[n] = H.dot(X_0[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
            else:
                z[n] = np.nan
        else:
            X_0[:, n] = true_trajectory_(X_0[:, n-1].reshape(2, 1)).reshape(-1)
            if random.random() >= P:
                z[n] = H.dot(X_0[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
            else:
                z[n] = np.nan
    X_1 = np.zeros((2, size))

    X_1 = Kalman(X_0_4, P_0_4, sigma_a, sigma_eta, size, z)[0]
    P_ = Kalman(X_0_4, P_0_4, sigma_a, sigma_eta, size, z)[1]
    X_7_3 = np.zeros((2, size + 7))

    for n in range(size):
        X_7_3[:, n + 7] = Phi.dot(Phi).dot(Phi).dot(Phi).dot(Phi).dot(Phi).dot(X_1[:, n]).reshape(-1)

    mistake_[:, :, m] = (X_1 - X_0) ** 2
    mistake_7_[:, :, m] = (X_7_3[:, :size] - X_0) ** 2
Final_err_1step = np.sqrt(np.sum(mistake_, axis = 2) / (M - 1))
Final_err_7steps = np.sqrt(np.sum(mistake_7_, axis = 2) / (M - 1))
```

In [6]:

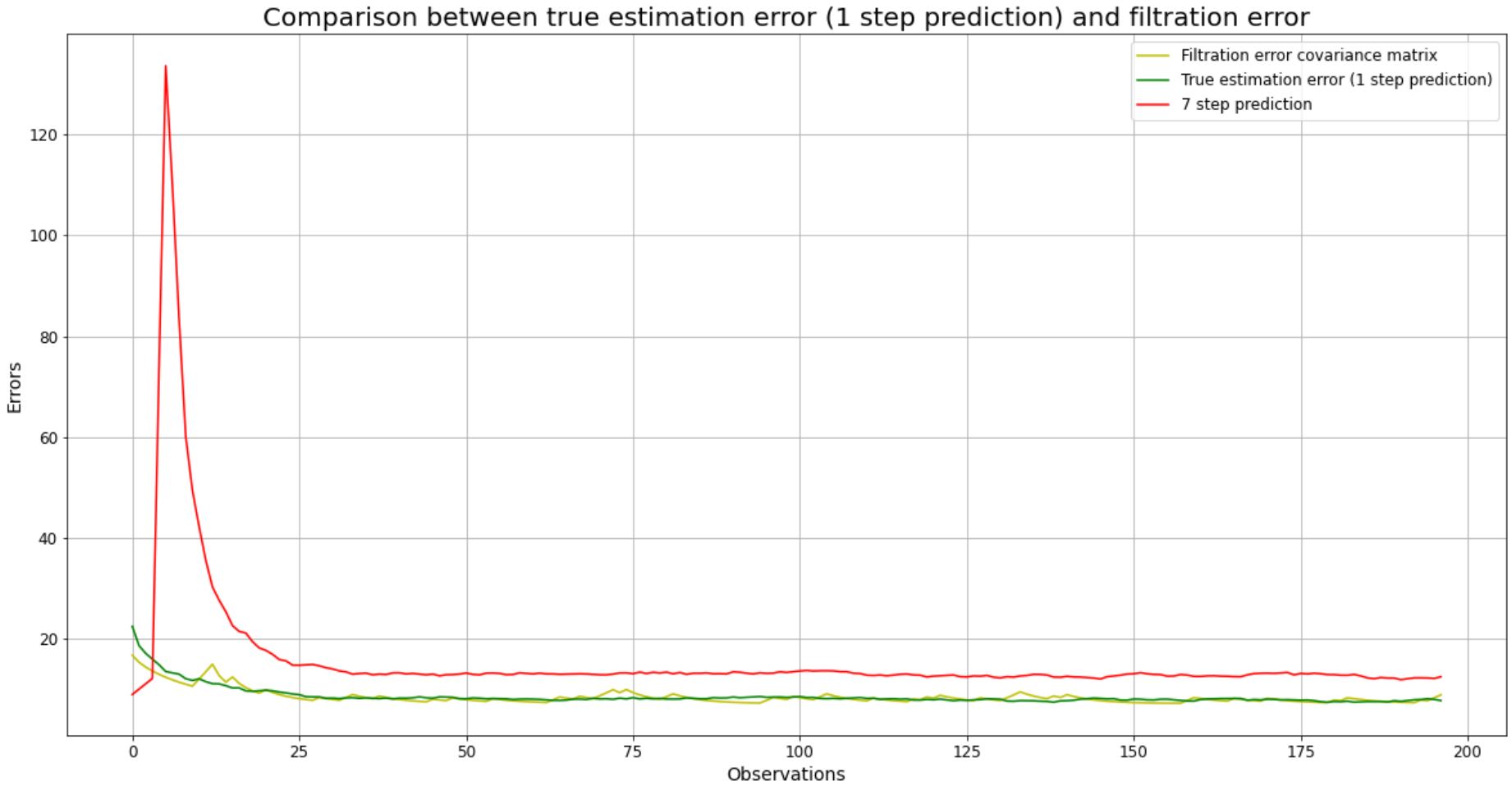
```
# Plotting filtered and true errors (extrapolated 1step and 7step ahead)
fig, e = plt.subplots(figsize=(20,10))
e.set_title("Comparison between true estimation error and filtration error", fontsize = 20)
e.set_xlabel("Observations", fontsize = 14)
e.set_ylabel("Errors", fontsize = 14)
e.plot(np.sqrt(P_[0, 0, 3 :]), label = "Filtration error covariance matrix", color = "y")
e.plot(Final_err_1step[0, 3:], label = "True filtered estimation error", color = "g")
e.tick_params(labelsize = 12)
e.legend(fontsize = 12)
e.grid()
```



In [7]:
```
fig, e = plt.subplots(figsize=(20,10))
e.set_title("Comparison between 7 step prediction and filtration error", fontsize = 20)
e.set_xlabel("Observations", fontsize = 14)
e.set_ylabel("Errors", fontsize = 14)
e.plot(np.sqrt(P_[0, 0, 3:]), label = "Filtration error covariance matrix", color = "y")
e.plot(Final_err_7steps[0, 3:], label = "7 step prediction error", color = "g")
e.tick_params(labelsize = 12)
e.legend(fontsize = 12)
e.grid()
```



In [8]:
```
# Plotting filtered, predicted and true errors
fig, e = plt.subplots(figsize=(20,10))
e.set_title("Comparison between true estimation error (1 step prediction) and filtration error", fontsize = 20)
e.set_xlabel("Observations", fontsize = 14)
e.set_ylabel("Errors", fontsize = 14)
e.plot(np.sqrt(P_[0, 0, 3 :]), label = "Filtration error covariance matrix", color = "y")
e.plot(Final_err_1step[0, 3:], label = "True estimation error (1 step prediction)", color = "g")
e.plot(Final_err_7steps[0, 3:], label = "7 step prediction", color = "r")

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



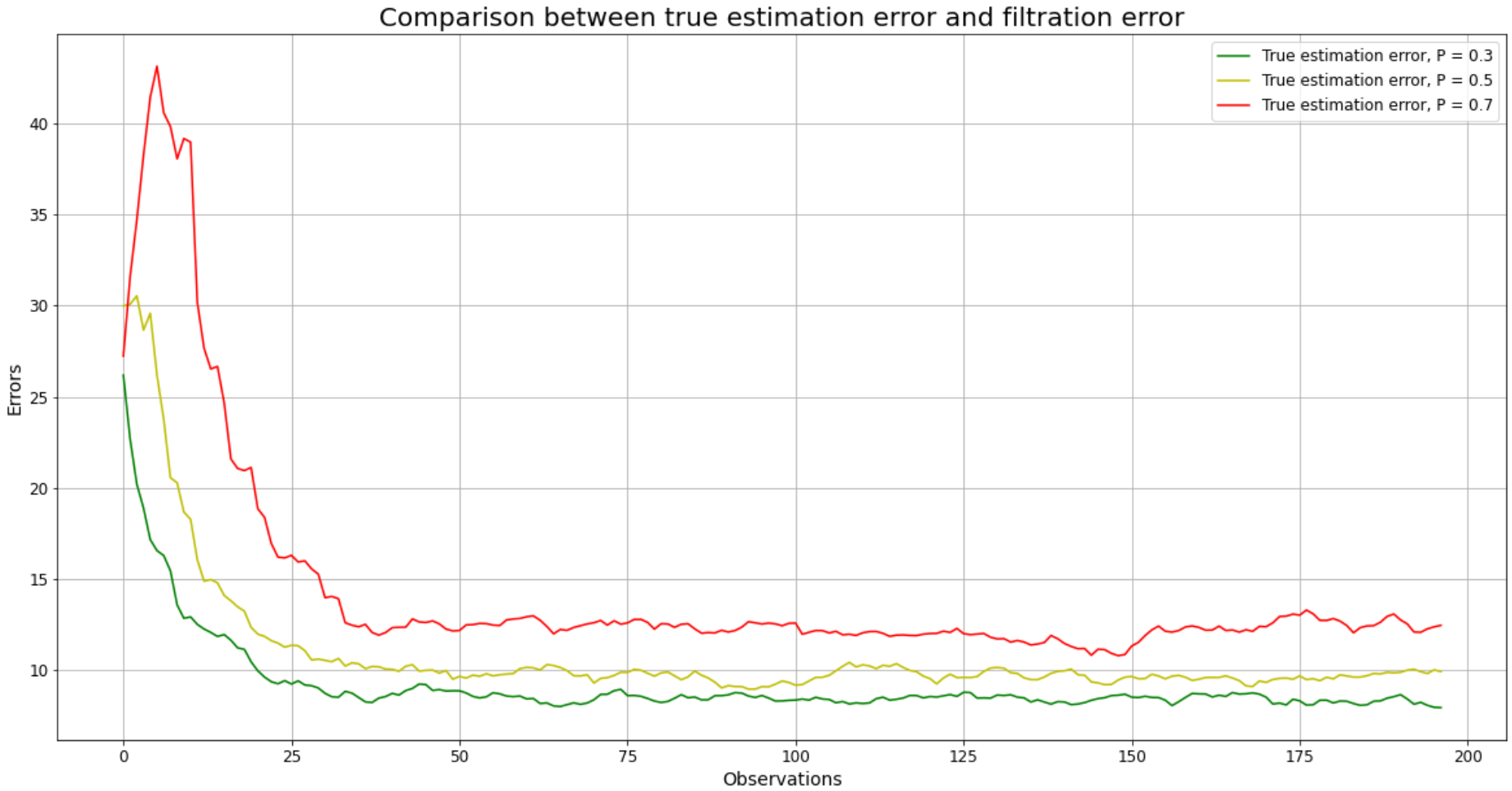
In [9]:

```
Probability = np.array([0.3, 0.5, 0.7])

Final_err_1step = np.zeros((2, len(Probability), size))
Final_err_7step = np.zeros((2, len(Probability), size))

for k in range (len(Probability)):
    mistake_ = np.zeros((2, size, M))
    mistake_7_ = np.zeros((2, size, M))

    for m in range (M):
        P_ = np.zeros((2, 2, size))

        for n in range(size):
```

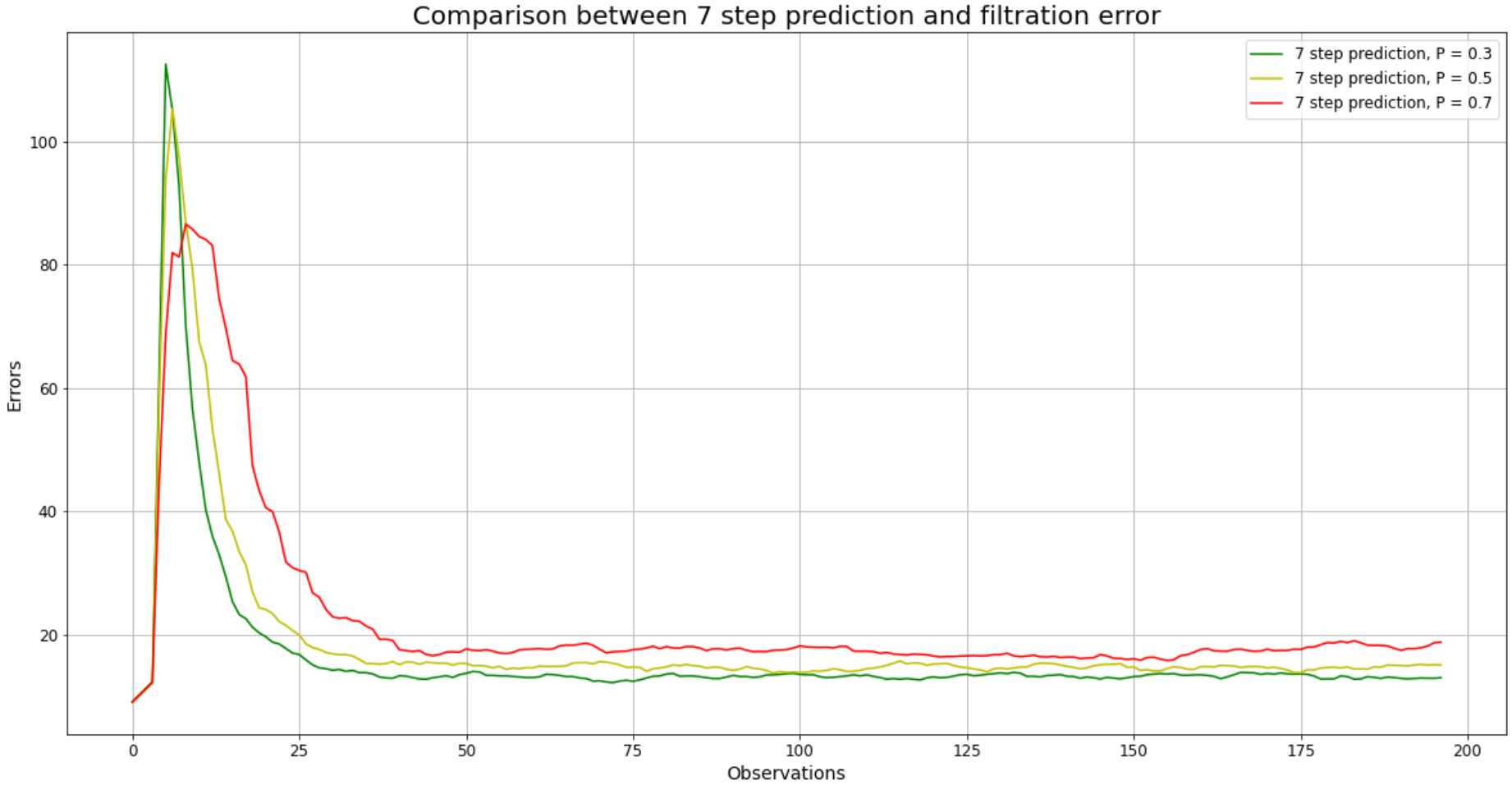
```
            if n == 0:
                X_0[:, n] = true_trajectory_(X_true_0).reshape(-1)
                if random.random() >= Probability[k]:
                    z[n] = H.dot(X_0[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
                else:
                    z[n] = np.nan
            else:
                X_0[:, n] = true_trajectory_(X_0[:, n-1].reshape(2, 1)).reshape(-1)
                if random.random() >= Probability[k]:
                    z[n] = H.dot(X_0[:, n].reshape(2, 1)) + np.random.normal(0, np.sqrt(sigma_eta))
                else:
                    z[n] = np.nan
        X_1 = np.zeros((2, size))

        X_1 = Kalman(X_0_4, P_0_4, sigma_a, sigma_eta, size, z)[0]
        P_ = Kalman(X_0_4, P_0_4, sigma_a, sigma_eta, size, z)[1]
        X_7_3 = np.zeros((2, size + 7))

        for n in range(size):
            X_7_3[:, n + 7] = Phi.dot(Phi).dot(Phi).dot(Phi).dot(Phi).dot(Phi).dot(X_1[:, n]).reshape(-1)

        mistake_[:, :, m] = (X_1 - X_0) ** 2
        mistake_7_[:, :, m] = (X_7_3[:, :size] - X_0) ** 2
    Final_err_1step[:, k, :] = np.sqrt(np.sum(mistake_, axis = 2) / (M - 1))
    Final_err_7step[:, k, :] = np.sqrt(np.sum(mistake_7_, axis = 2) / (M - 1))
```

In [10]:

```
# Plotting filtered and true estimation errors

fig, e = plt.subplots(figsize=(20,10))
e.set_title("Comparison between true estimation error and filtration error", fontsize = 20)
e.set_xlabel("Observations", fontsize = 14)
e.set_ylabel("Errors", fontsize = 14)
e.plot(Final_err_1step[0, 0, 3:], label = "True estimation error, P = 0.3", color = "g")
e.plot(Final_err_1step[0, 1, 3:], label = "True estimation error, P = 0.5", color = "y")
e.plot(Final_err_1step[0, 2, 3:], label = "True estimation error, P = 0.7", color = "r")
#e.plot(Err_1step_[0, 3:], label = "True estimation error, P = 0.7", color = "r")
#e.plot(X_1_K[0, :], label = "True estimation error, P = 0.7", color = "r")

e.tick_params(labelsize = 12)
e.legend(fontsize = 12)
e.grid()
plt.savefig('True_error')
```



In [11]:
```
    # Plotting filtered and extrapolated 7 step ahead errors
fig, e = plt.subplots(figsize=(20,10))
e.set_title("Comparison between 7 step prediction and filtration error", fontsize = 20)
e.set_xlabel("Observations", fontsize = 14)
e.set_ylabel("Errors", fontsize = 14)
e.plot(Final_err_7step[0, 0, 3:], label = "7 step prediction, P = 0.3", color = "g")
e.plot(Final_err_7step[0, 1, 3:], label = "7 step prediction, P = 0.5", color = "y")
e.plot(Final_err_7step[0, 2, 3:], label = "7 step prediction, P = 0.7", color = "r")
e.tick_params(labelsize = 12)
e.legend(fontsize = 12)
e.grid()
plt.savefig('7_error')
```


Comparison between 7 step prediction and filtration error
7 step prediction, P = 0.3
7 step prediction, P = 0.5
7 step prediction, P = 0.7
Errors
Observations
100
80
60
40
20
0
25
50
75
100
125
150
175
200