Skoltech

Skolkovo Institute of Science and Technology

Development of optimal smoothing to increase the estimation accuracy

Experimental Data Processing

Assignment №6

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 algorithms to improve Kalman filter estimates, 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 - 3

For this question we developed backward smoothing algorithm to get improved estimates of state vector X_i using Equations Every condition is same as in Lab 5. Backward smoothing is applied using following formulas:

$$X_{i,N} = X_{i,i} + A_i(X_{i+1,N} - \phi_{i+1,i}X_{i,i}) \tag{1}$$

$$A_i = P_{i,i}\phi^T_{i+1,i}P^{-1}_{i+1,i} \tag{2}$$

$$P_{i,N} = P_{i,i} + A_i(P_{i+1,N} - P_{i+1,i})A^T_i \tag{3}$$

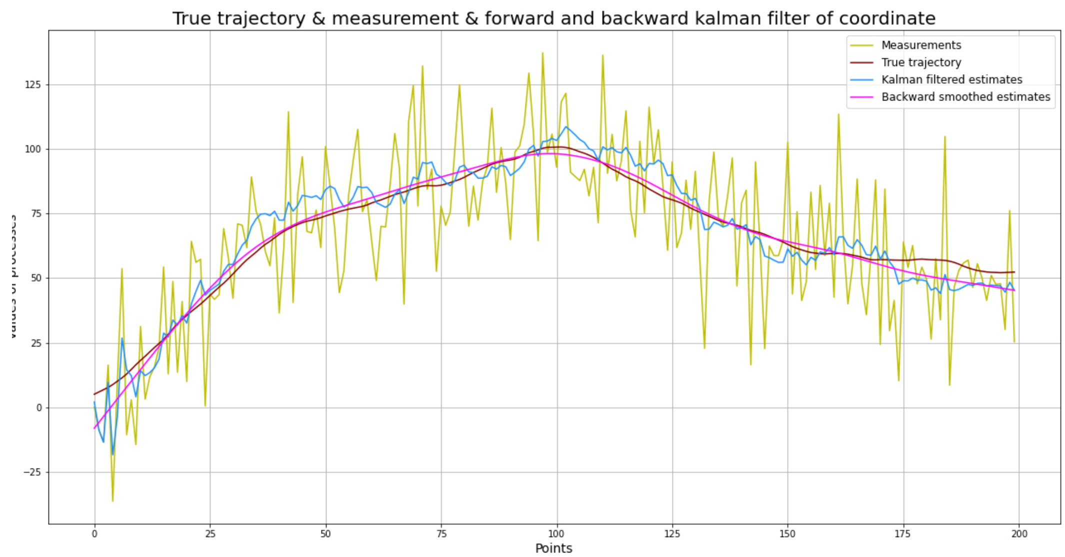


Figure 1: Comparison of data

In this part, backward smoothing algorithm is applied that is basically the Kalman filter in the previous assignment starting backwards. In Figure 1, true trajectory, measurements, filtered data and backward smoothed data are presented over the observation interval. We can see that smoothed data is much closer to the true data and it doesn't have sharp edges just like in the filtered data. Therefore, backward smoothing is much more smoother than the filtered data.

Question 4

For this part, we made $M = 500$ runs of smoothing and compared true estimation error with errors of smoothing $P_{i,N}$ provided by smoothing algorithm (error of smoothed estimates and coordinate x_i and velocity V_i).

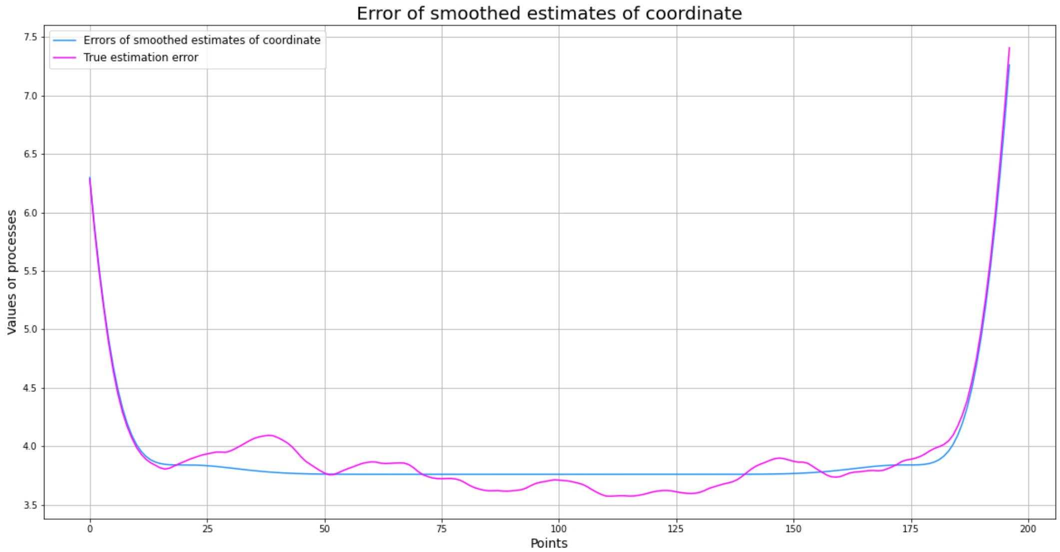


Figure 2: Error of smoothed estimates of coordinate x_i

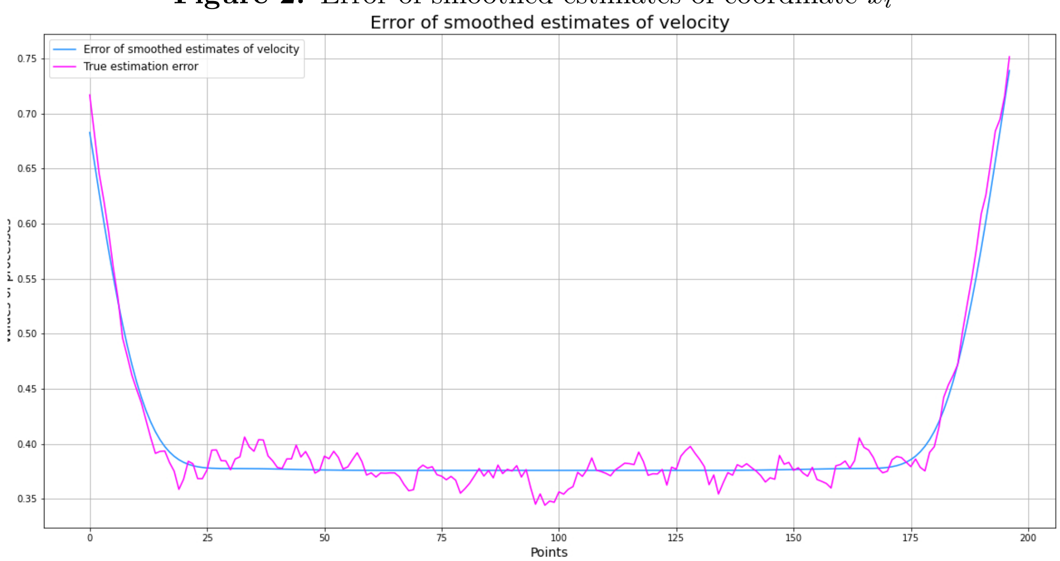


Figure 3: Error of smoothed estimates of velocity V_i

Both errors of smoothed estimates of coordinate and velocity are shown in Figure 2 and 3. In both figures, smoothness of the smoothed data stands out and it can be seen that is is sort of in the middle of the true estimation errors. Also, we can visually see that both of them complies with the true estimation error.

Question 5

In this part of the assignment, we compared the smoothing errors of estimation with filtration errors of estimation. In Figure 1, it was shown that backward smoothing is closer to the true data and much smoother than the filtered data. In order to see the difference better, the errors of backward smoothing and filtration are shown for both coordinate and velocity. It is seen in both cases that backward smoothing algorithm has less error and gives more accurate results.

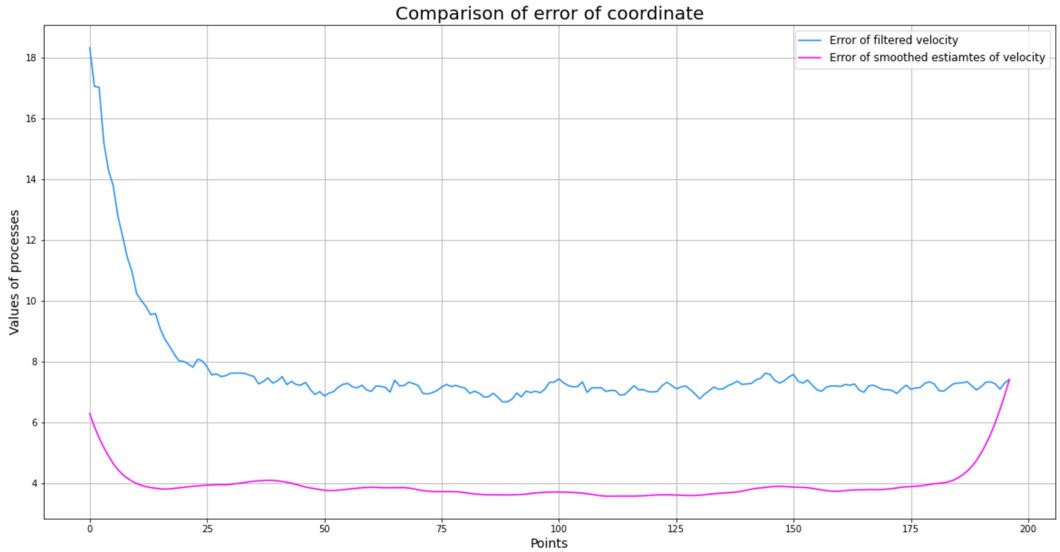


Figure 4: Comparison of errors x_i

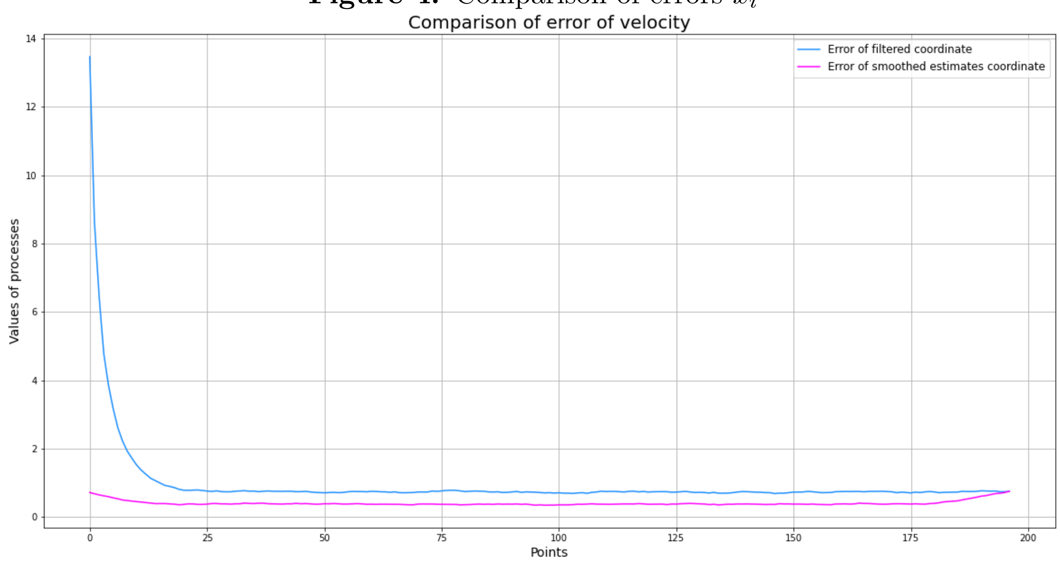


Figure 5: Comparison of errors V_i

3 Conclusion

What we have learnt and tried:

1. How to develop a backward smoothing algorithm.

2. Backward smoothing data is more accurate and smoother than the filtration.

3. To both coordinate and velocity values in order to see the results more clearly.

4. By smoothing we take into account both current and future measurements which provides better results than Kalman filter.

What we have reflected upon:

1. Used backward smoothing algorithm to have smoother data

2. Compared smoothing errors of estimation with filtration errors of estimation to see which one is useful in this case

3. How to develop algorithms to improve Kalman filter

4. We proved that backward smoothed data is much closer to the true data and it doesn't have sharp edges just like in the filtered data.

5. Backward smoothing has less error than the filtered estimates.

Contribution of each members:

1. Ilya: wrote the code and report.

2. Ruslan: wrote the code, made plots and wrote report.

3. Yunseok: wrote the code for backward smoothing.

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

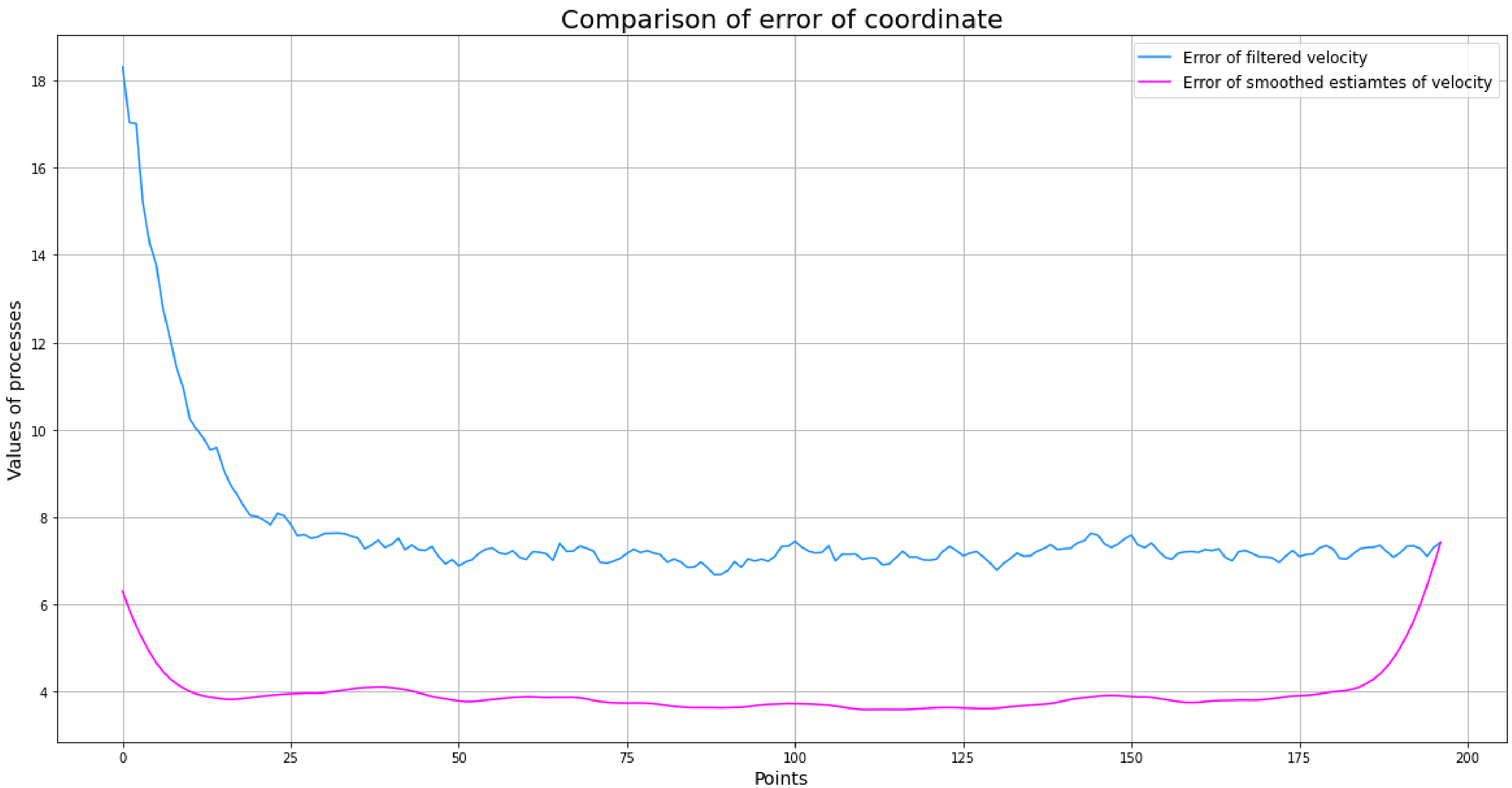
In [2]:
```
# Size of the trajectory
n = 200
M = 500

# Time step
T = 1

# Variances
sigma_a2 = 0.2 ** 2
sigma_et2 = 20 ** 2

# Errors initialization
Error_filt = np.zeros((2, n, M))
Error_smoothed = np.zeros((2, n, M))

for k in range(M):
    # Initialization of arrays
    x = np.zeros((n, 1))
    V = np.zeros((n, 1))
    z = np.zeros((n, 1))

    x[0] = 5
    V[0] = 1

    # Generation of normally distributed random noises with zero mathematical expectation and corresponding variances
    a = np.random.normal(0, np.sqrt(sigma_a2), n - 1)
    et = np.random.normal(0, np.sqrt(sigma_et2), n)

    for i in range(1, len(V)):
        V[i] = V[i-1] + a[i - 1] * T
        x[i] = x[i - 1] + V[i - 1] * T + (a[i - 1] ** T) / 2
        z[i] = x[i] + et[i]

    #Transition matrix
    phi = np.array([[1, T], [0, 1]])
    #Input matrix
    G = np.array([[(T ** 2) / 2.0], [T]])
    #Observation matrix
```

```python
H = np.array([1, 0])
#Measurement of coordinate
Z = np.zeros((2, n))
#State vector
X = np.array([[5], [1]])

X_ = X
# Generation of true trajectory X
for i in range(1,len(a)+1):
    X = np.hstack((X, phi.dot(X_) + G * a[i - 1]))
    X_ = phi.dot(X_) + G * a[i - 1]

#Covariance matrix Q of state noise
Q = G * G.T * sigma_a2
#Covariance matrix R of measurements noise
R = sigma_et2

# Initialization of matrixes
P_pred = np.zeros((2, 2, n))
X_pred = np.zeros((2, n))
P_filt = np.zeros((2, 2, n))
X_filt = np.zeros((2, n))
K = np.zeros((2, n))
HT = H.T

#Initial P for filtering
P_filt[:, :, 0] = [[10000, 0],[0, 10000]]
#Initial X_filt for filtering
X_filt[:, 0] = [2, 0]

# Kalman filtering
for i in range(1, n):
    X_pred[:, i] = phi.dot(X_filt[:, i - 1].reshape(2, 1)).reshape(2)
    P_pred[:, :, i] = (phi.dot(P_filt[:, :, i - 1])).dot(phi.T) + Q

    K[:, i] = ((P_pred[:, :, i].dot(HT)) / ((H.dot(P_pred[:, :, i])).dot(HT) + R)).reshape(2)
    X_filt[:, i] = X_pred[:, i] + K[:, i] * (z[i] - H.dot(X_pred[:, i]))
    P_filt[:, :, i] = (np.eye(2) - K[:, i].reshape(2, 1) * H).dot(P_pred[:, :, i])

K = np.delete(K, 0, axis = 1)

#Backward smoothing
X_back = np.zeros((2, n))
X_back[:, -1] = X_filt[:, -1]
```

```
    P_back = np.zeros((2, 2, n))
    P_back[:, :, -1] = P_filt[:, :, -1]
    A_back = np.zeros((2, 2, n))

    for i in reversed(range(n - 1)):
        A_back[:, :, i] = P_filt[:, :, i].dot(phi.T).dot(np.linalg.inv(P_pred[:, :, i + 1]))
        X_back[:, i] = X_filt[:, i] + A_back[:, :, i].dot(X_back[:, i + 1] - phi.dot(X_filt[:, i]))
        P_back[:, :, i] = P_filt[:, :, i] + A_back[:, :, i].dot(P_back[:, :, i + 1] - P_pred[:, :, i + 1]).dot(A_back[:, :, i].T)

    Error_filt[:, :, k] = (X - X_filt) ** 2
    Error_smoothed[:, :, k] = (X - X_back) ** 2

Final_err_filt = np.sqrt(np.sum(Error_filt, axis = 2) / (M - 1))
Final_err_smoothed = np.sqrt(np.sum(Error_smoothed, axis = 2) / (M - 1))
```

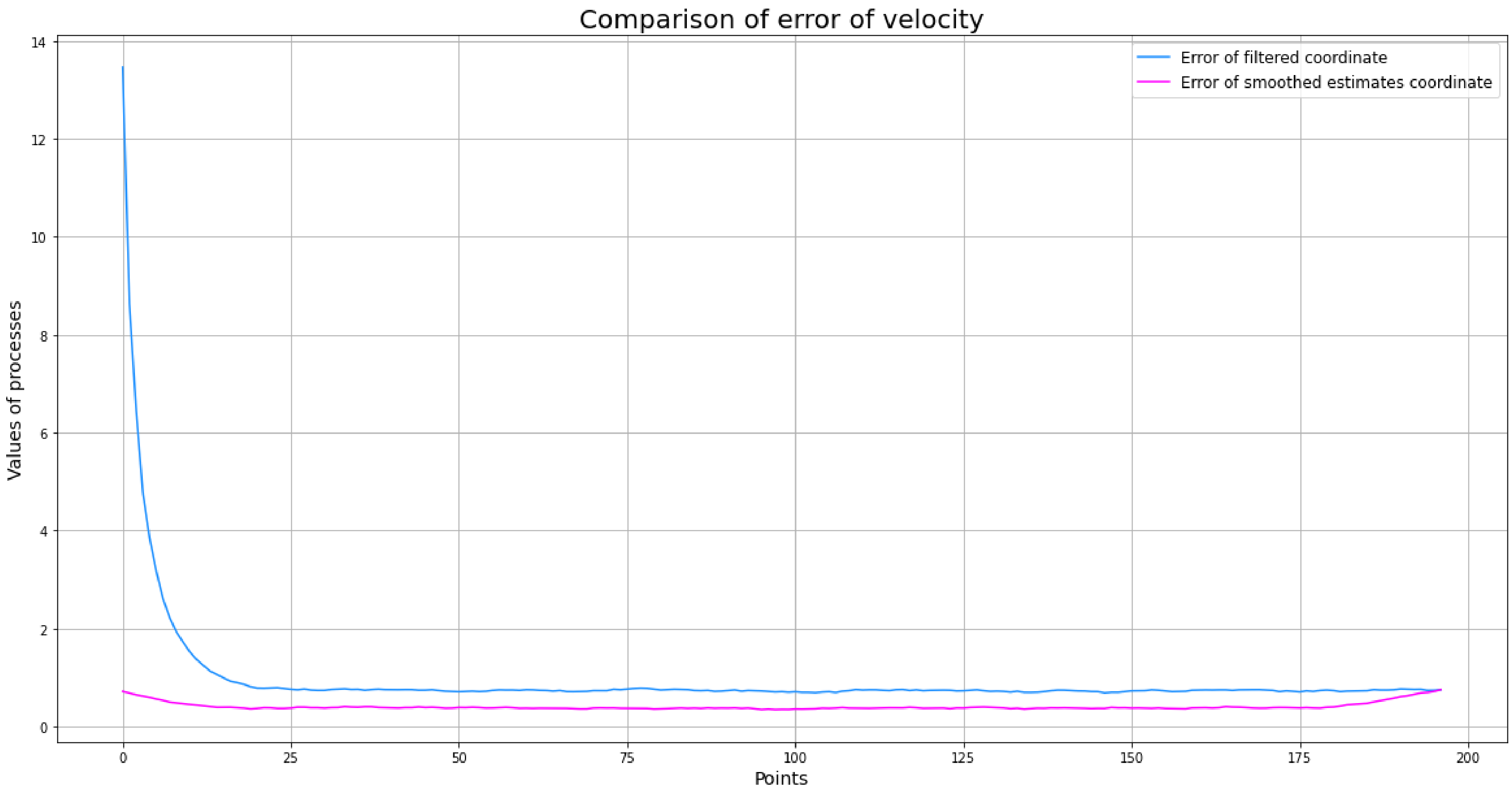
In [3]:

```
#Compare of true smoothing and filtration errors of estimation of coordinates
fig, a = plt.subplots(figsize=(20,10))
a.set_title("Comparison of error of coordinate", fontsize = 20)
a.set_xlabel("Points", fontsize = 14)
a.set_ylabel("Values of processes", fontsize = 14)
a.plot(Final_err_filt[0,3:], label = "Error of filtered velocity", c='dodgerblue')
a.plot(Final_err_smoothed[0,3:], label = "Error of smoothed estiamtes of velocity", c='magenta')
a.legend(fontsize = 12)
a.grid()
plt.savefig('compare coordinate')
```



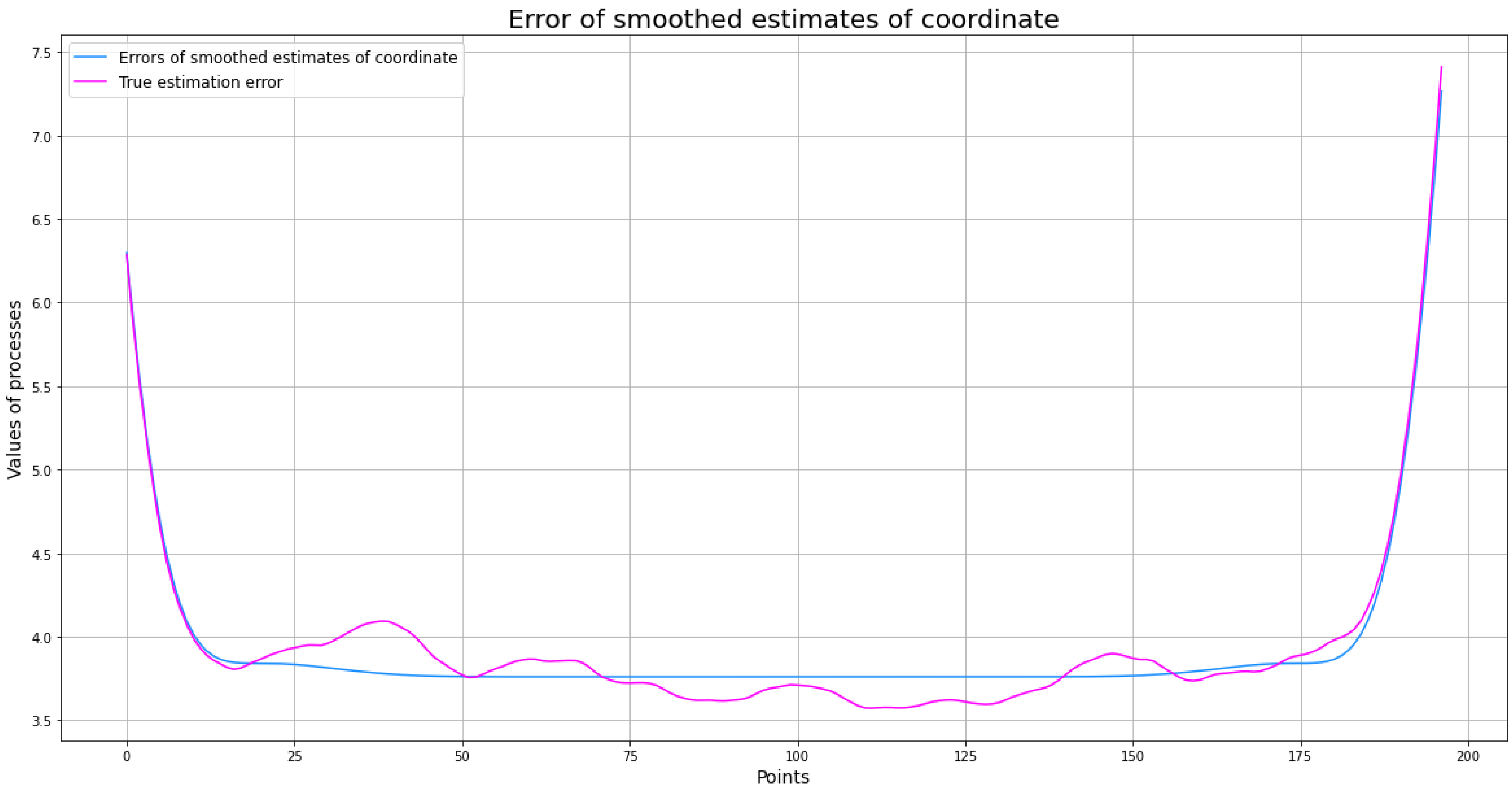
In [4]:

```
#Compare of true smoothing and filtration errors of estimation of velocities
fig, b = plt.subplots(figsize=(20,10))
b.set_title("Comparison of error of velocity", fontsize = 20)
b.set_xlabel("Points", fontsize = 14)
b.set_ylabel("Values of processes", fontsize = 14)
b.plot(Final_err_filt[1,3:], label = "Error of filtered coordinate", c='dodgerblue')
b.plot(Final_err_smoothed[1,3:], label = "Error of smoothed estimates coordinate", c='magenta')
b.legend(fontsize = 12)
b.grid()
plt.savefig('compare velocity')
```



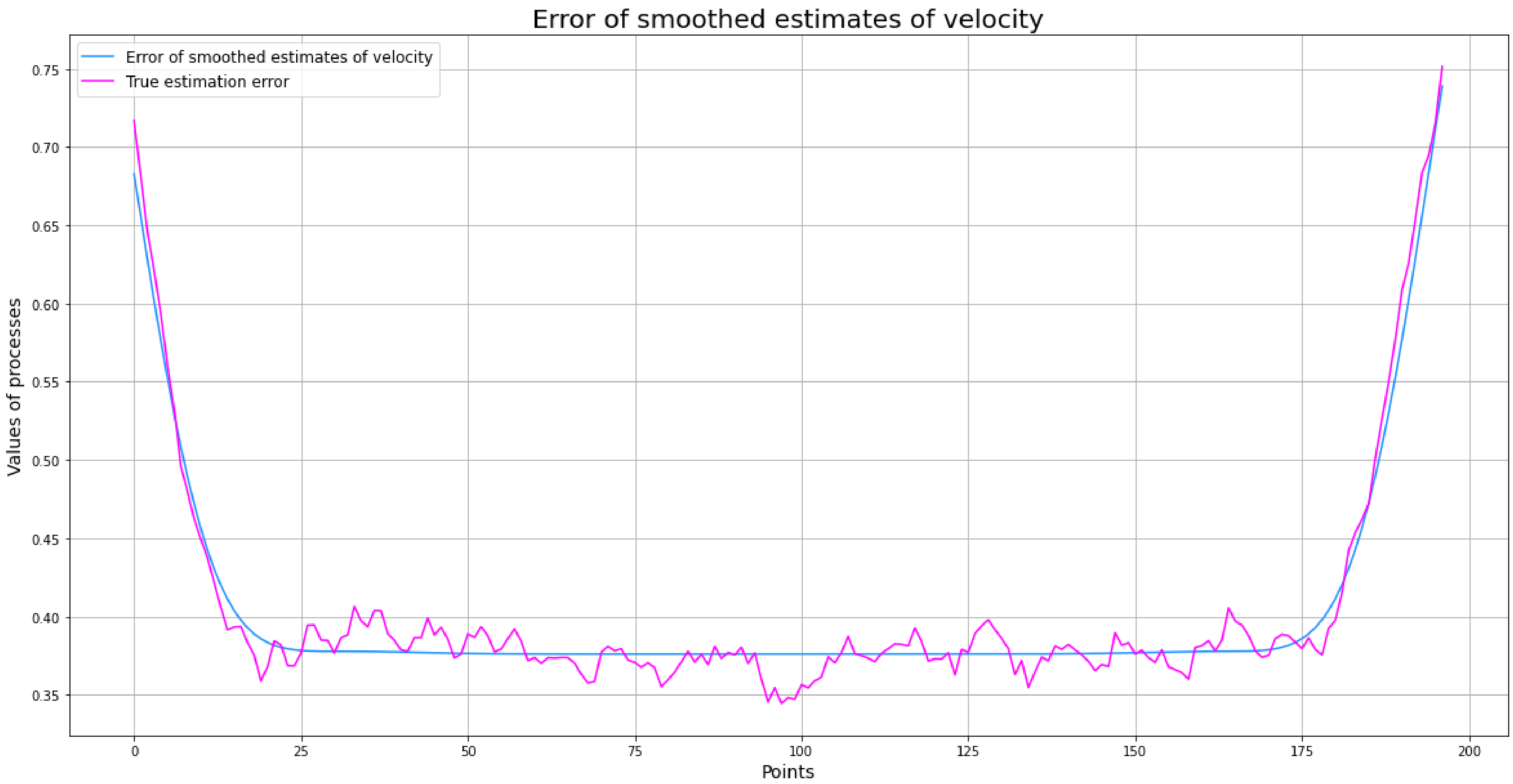
In [5]:

```
# Compare of true errors of coordinates and errors provided by smoothing algorithm
fig, c = plt.subplots(figsize=(20,10))
c.set_title("Error of smoothed estimates of coordinate", fontsize = 20)
c.set_xlabel("Points", fontsize = 14)
c.set_ylabel("Values of processes", fontsize = 14)
c.plot(np.sqrt(P_back[0, 0, 3:]), label = "Errors of smoothed estimates of coordinate", c='dodgerblue')
c.plot(Final_err_smoothed[0, 3:], label = "True estimation error", c='magenta')
c.legend(fontsize = 12)
c.grid()
plt.savefig('coordinate error')
```



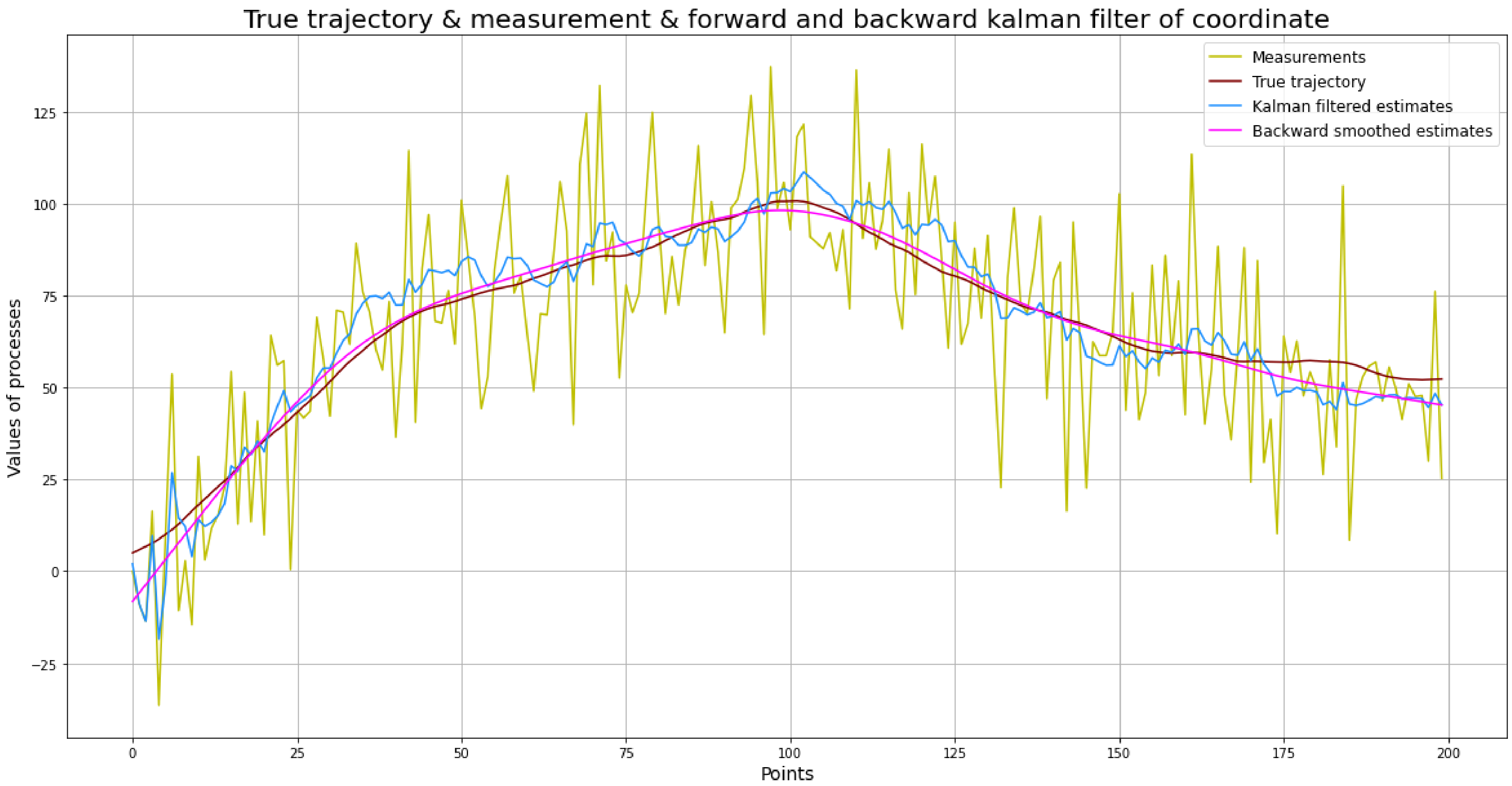
In [6]:

```
# Compare of true errors of velocities and errors provided by smoothing algorithm
fig, d = plt.subplots(figsize=(20,10))
d.set_title("Error of smoothed estimates of velocity", fontsize = 20)
d.set_xlabel("Points", fontsize = 14)
d.set_ylabel("Values of processes", fontsize = 14)
d.plot(np.sqrt(P_back[1, 1, 3:]), label = "Error of smoothed estimates of velocity", c='dodgerblue')
d.plot(Final_err_smoothed[1, 3:], label = "True estimation error", c='magenta')
d.legend(fontsize = 12)
d.grid()
plt.savefig('velocity error')
```



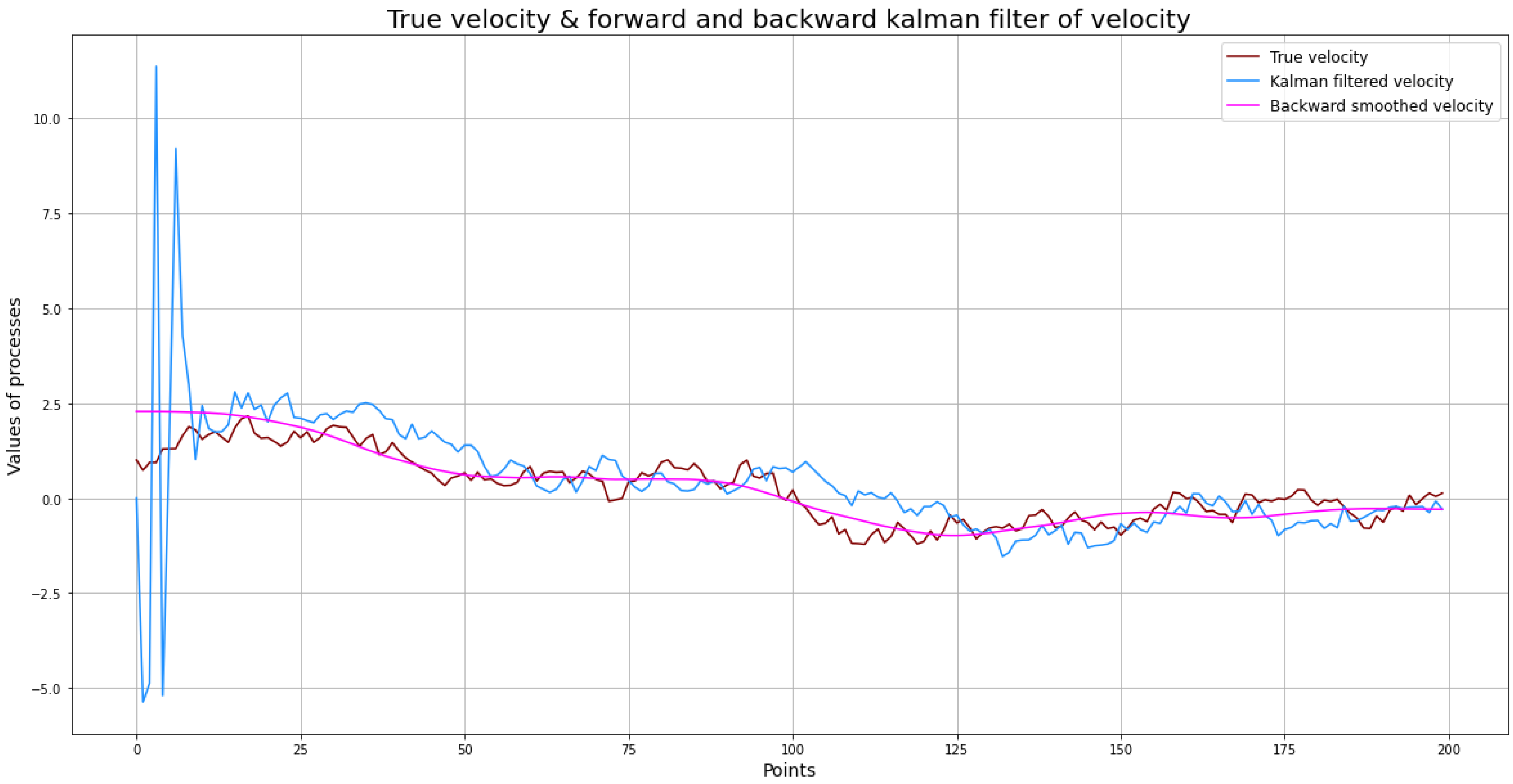
In [7]:

```
# Plot of comparison of coordinate trajectories
fig, e = plt.subplots(figsize=(20,10))
e.set_title("True trajectory & measurement & forward and backward kalman filter of coordinate", fontsize = 20)
e.set_xlabel("Points", fontsize = 14)
e.set_ylabel("Values of processes", fontsize = 14)
e.plot(z, label = "Measurements", c='y')
e.plot(X[0,:], label = "True trajectory", c='maroon')
e.plot(X_filt[0,:], label = "Kalman filtered estimates", c='dodgerblue')
e.plot(X_back[0,:], label = "Backward smoothed estimates", c='magenta')
e.legend(fontsize = 12)
e.grid()
plt.savefig('coordinate')
```



In [8]:

```
# Plot of comparison of coordinate trajectories
fig, f = plt.subplots(figsize=(20,10))
f.set_title("True velocity & forward and backward kalman filter of velocity", fontsize = 20)
f.set_xlabel("Points", fontsize = 14)
f.set_ylabel("Values of processes", fontsize = 14)
f.plot(X[1,:], label = "True velocity", c='maroon')
f.plot(X_filt[1,:], label = "Kalman filtered velocity", c='dodgerblue')
f.plot(X_back[1,:], label = "Backward smoothed velocity", c='magenta')
f.legend(fontsize = 12)
f.grid()
plt.savefig('velocity')
```


True velocity & forward and backward kalman filter of velocity
True velocity
Kalman filtered velocity
Backward smoothed velocity
Values of processes
Points
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
25
50
75
100
125
150
175
200