Laboratory work 5 - Tracking of a moving object which trajectory is disturbed by random acceleration

Abramov Semen, Belikov Ilia, Nikolay Zherdev, Mikhail Kulbeda

Skoltech, 11.10.2018

The objective of this laboratory work is to develop standard Kalman filter for tracking a moving object which trajectory is disturbed by random acceleration. Important outcome of this exercise is getting deeper understanding of Kalman filter parameters and their role in estimation. Students will analyze the sensitivity of estimations to choice of non-optimal parameters and dependence on initial conditions.

In [5]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go
from sklearn.metrics import mean_squared_error


init_notebook_mode(connected=True)
```

1. Generate a true trajectory X_i of an object motion disturbed by normally distributed random acceleration

$$x_i = x_{i-1} + V_{i-1}T + \frac{a_{i-1}T^2}{2}$$
$$V_i = V_{i-1} + a_{i-1}T$$

Size of trajectory is 200 points.
Initial conditions: $x_1 = 5; V_1 = 1; T = 1$
Variance of noise a_i, $\sigma_a^2 = 0.2^2$

2. Generate measurements z_i of the coordinate x_i

$$z_i = x_i + \eta_i$$

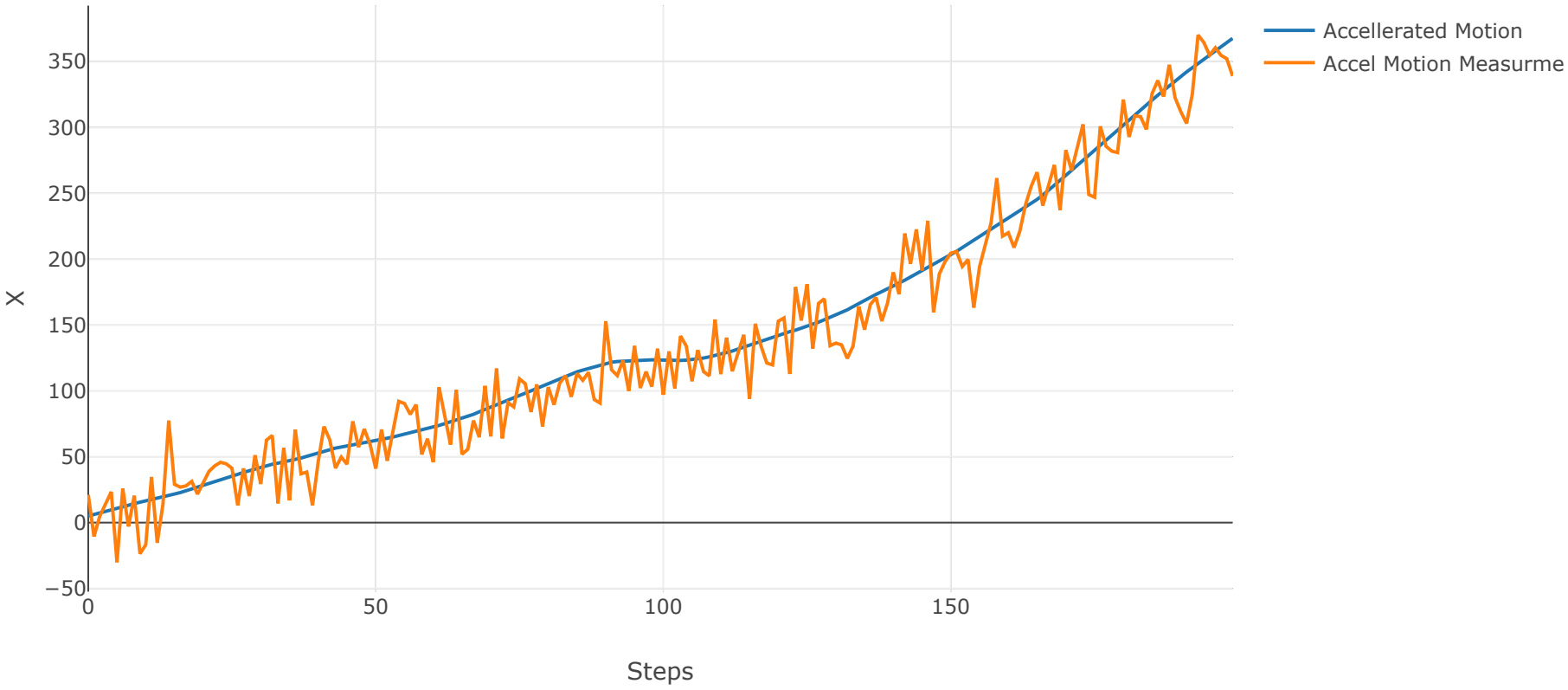
η_i –normally distributed random noise with zero mathematical expectation and variance $\sigma_\eta^2 = 20^2$.

In [8]:
```
# Accelerated motion
def generate_trajectory(X0 = 5, V0 = 0, T = 0.1, mean = 0, var = 0.1, steps = 300):

    velocity = np.zeros(steps)
    velocity[0] = V0
    accels = np.random.normal(loc = mean, scale = np.sqrt(var), size = steps)
    velocity[1:] = accels[:-1]*T
    velocity = np.cumsum(velocity)


    X = np.zeros(steps)
    X[0] = X0
    X[1:] = velocity[:-1]*T + accels[:-1]*(T**2)/2
    return np.cumsum(X)
```

In [9]:
```
def measure(trajectory, mean = 0, var = 0.1):
    noise = np.random.normal(loc = mean, scale = np.sqrt(var), size = trajectory.shape)
    return np.add(trajectory, noise)
```

In [10]:
```
tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = 0.2**2, steps = 200)
measurments = measure(tr, var = 20**2) # array Nx1
```

In [11]:
```
data = [
    go.Scatter(
        y=tr,
        name='Accellerated Motion'
    ),
    go.Scatter(
        y=measurments,
        name='Accel Motion Measurments'
    ),
]

layout= go.Layout(
    title= 'Accellerated Motion and Accel Motion Measurments',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



3. Present the system at state space taking into account that only measurements of coordinate x_i are available

$$X_i = \Phi X_{i-1} + G a_{i-1}$$
$$z_i = H_i X_i + \eta_i$$

Here $\boldsymbol{X_i}$ - state vector, that describes full state of the system (coordinate x_i and velocity V_i);
$\boldsymbol{\Phi}$ – transition matrix that relates X_i and X_{i-1};
$\boldsymbol{G}$ – input matrix, that determines how random acceleration a_i affects state vector;
$\boldsymbol{z_i}$ – measurements of coordinate x_i
$\boldsymbol{H}$ – observation matrix

4. Develop Kalman filter algorithm to estimate state vector X_i (extrapolation and filtration)
Consult charts from lecture **Topic_3_Optimal approximation at state space.pdf**

Use initial conditions

Initial filtered estimate $X_0 = \begin{vmatrix} 2 \\ 0 \end{vmatrix}$

Initial filtration error covariance matrix

$$P_{0,0} = \begin{vmatrix} 10000 & 0 \\ 0 & 10000 \end{vmatrix}$$

Useful hints

To calculate covariance matrix Q of state noise Ga_{i-1} that is used in Kalman filter algorithm to determine prediction error covariance matrix use following equation

$$\begin{gathered} Q = E[\,(Ga_{i-1})(Ga_{i-1})^T] = \\ = GE[\,a_{i-1}^2]G^T = \\ = G\sigma_a^2 G^T = GG^T\sigma_a^2 \end{gathered}$$

To calculate covariance matrix R of measurements noise η_i that is used in Kalman filter algorithm to determine filter gain use following recommendation: Dimension of covariance matrix R is determined by a number of state vector components that are measured. In this particular case, only coordinate x_i is measured. Thus

$$R = \sigma_\eta^2$$

❶ Prediction (extrapolation)

Prediction of state vector at time i using $i-1$ measurements

$$X_{i,i-1} = \Phi_{i,i-1} X_{i-1,i-1}$$

Prediction error covariance matrix

$$P_{i,i-1} = \Phi_{i,i-1} P_{i-1,i-1} \Phi_{i,i-1}^T + Q_i$$

$$P_{i,i-1} = E[(X_i - X_{i,i-1})(X_i - X_{i,i-1})^T]$$

❷ Filtration

Adjustment of predicted estimate

Improved estimate by incorporating a new measurement

$$X_{i,i} = X_{i,i-1} + K_i(z_i - HX_{i,i-1})$$

Residual

Filter gain, weight of residual

$$K_i = P_{i,i-1}H_i^T\left(H_iP_{i,i-1}H_i^T + R_i\right)^{-1}$$

Filtration error covariance matrix

$$P_{i,i} = (I - K_iH_i)P_{i,i-1}$$

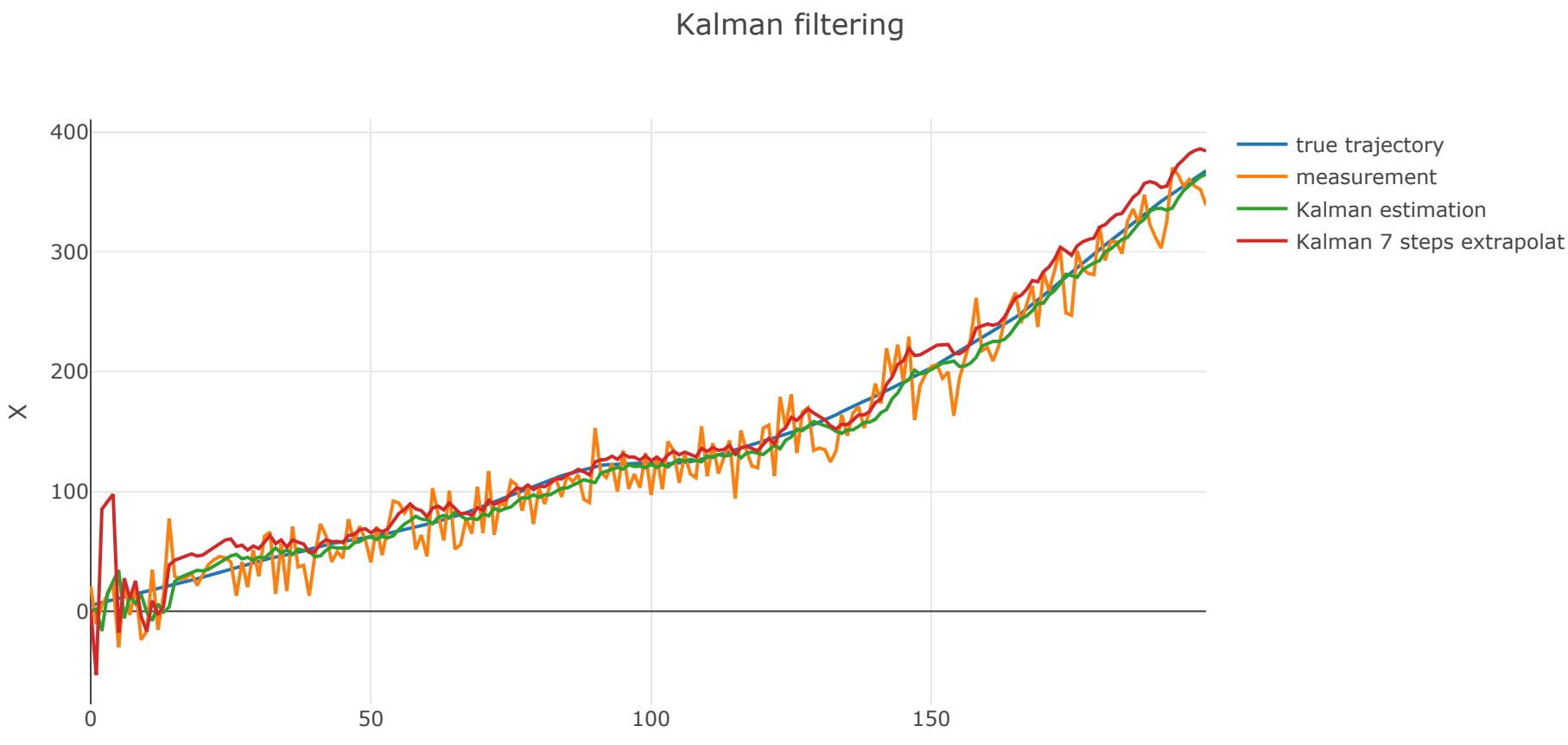
$$P_{i,i} = E[(X_i - X_{i,i})(X_i - X_{i,i})^T]$$

In [14]:
```
def kalman(measr, T = 1, X_init = [2.0, 0], P_init = [[10000, 0], [0, 10000]], var_a = 0.2**2, var_n = 20**2):

    size = (len(measr), 2) # size of array
    G = np.array([T**2/2, T]).T
    Q = G.dot(G.T.dot(var_a))
    R = var_n
    F = np.array([[1, T], [0, 1]])
    H = np.array([1, 0])
    X = np.zeros(size)
    X_ = np.zeros(size)
    P = np.zeros((size[0], 2, 2))
    P_ = np.zeros((size[0], 2, 2))
    K = np.zeros(size)

    X_[0] = X_init
    P_[0] = P_init

    for k in range(1, size[0]):

        # Prediction
        X[k] = np.dot(F, X_[k-1])
        P[k] = np.dot(F, np.dot(P_[k-1], F.T)) + Q

        # Update
        K[k] = np.dot(P[k], H.T)/(np.dot(H, np.dot(P[k], H.T)) + R)
        X_[k] = X[k] + K[k]*(measr[k] -  np.dot(H, X[k]))
        P_[k] = np.dot((np.eye(2) - np.outer(K[k], H)), P[k])

    return X[:,0], X_[:,0], K[:,0], np.sqrt(P[:,0,0]), np.sqrt(P_[:,0,0]), np.linalg.matrix_power(F,7).dot(X_.T)[0]
```

In [21]:
```
X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments)
```

5

In [23]:
```
data = [
    go.Scatter(
        y = tr,
        name='true trajectory'
    ),
    go.Scatter(
        y = measurments,
        name='measurement'
    ),
    go.Scatter(
        y = kalman(measurments)[0],
        name='Kalman estimation'
    ),
    go.Scatter(
        y = kalman(measurments)[5],
        name='Kalman 7 steps extrapolation'
    ),
]

layout= go.Layout(
    title= 'Kalman filtering',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



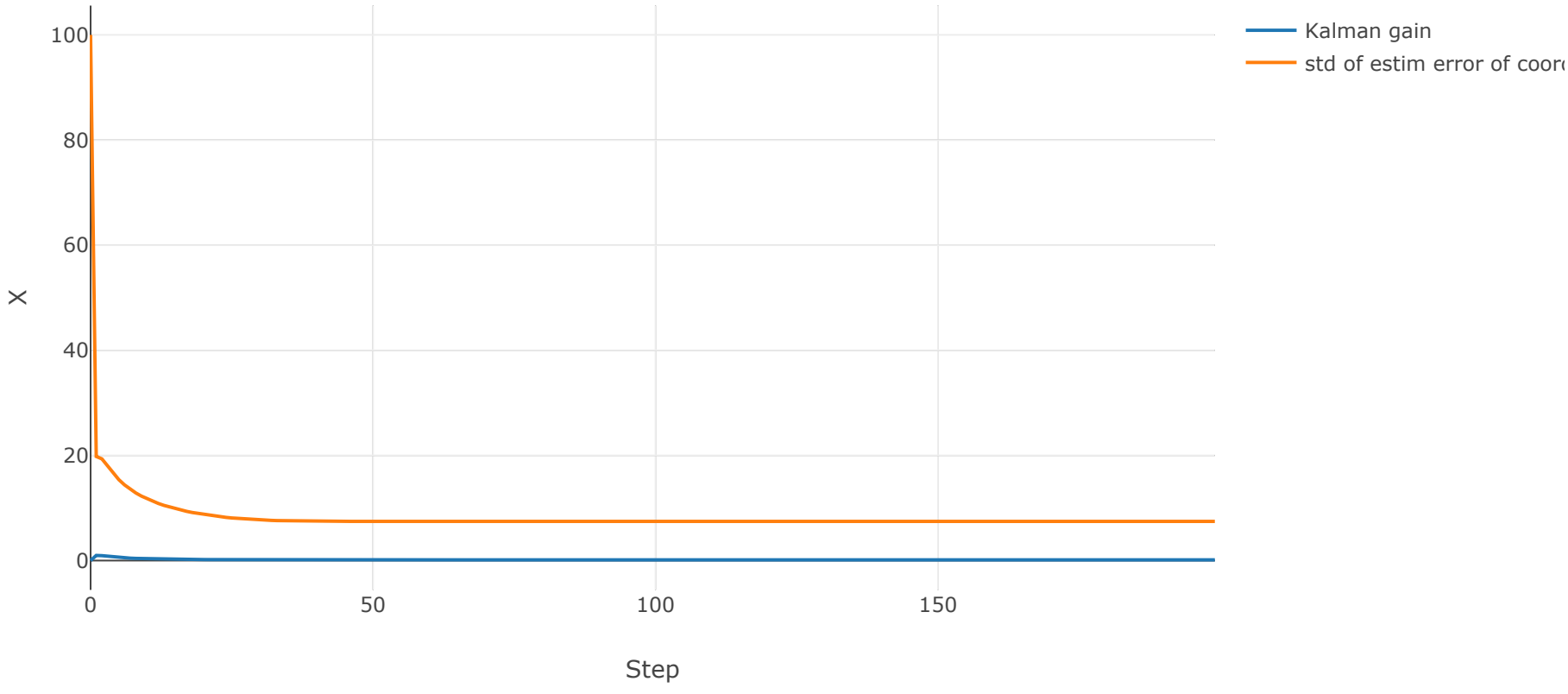
Export to plo

6. Plot filter gain K over the whole filtration interval.

   To analyze filtration error covariance matrix $P_{i,i}$ over observation period, please also make another plot of square root of its first diagonal element corresponding to standard deviation of estimation error of coordinate x_i.

   Verify whether filter gain K and filtration error covariance matrix become constant very quickly. It means that in conditions of a trajectory disturbed by random noise we cannot estimate more than established limit of accuracy due to uncertainty.

In [24]:
```
data = [
    go.Scatter(
        y = kalman(measurments)[2],
        name='Kalman gain'
    ),
    go.Scatter(
        y=kalman(measurments)[4],
        name='std of estim error of coord x'
    ),
]

layout= go.Layout(
    title= 'Kalman gain',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Filter gain K and filtration error covariance matrix become constant very quickly. It means that in conditions of a trajectory disturbed by random noise we cannot estimate more than established limit of accuracy due to uncertainty

7. Add to the code extrapolation on $m = 7$ steps ahead on every time step.

$$X_{i+m-1,i} = \Phi_{i+m-1,i} X_{i,i}$$

Here

$$\Phi_{i+m-1,i} = \Phi_{i+m-1,i+m-2} \Phi_{i+m-2,i+m-3} \cdots \Phi_{i+2,i+1} \Phi_{i+1,i}$$

For example

$$X_{7,1} = \Phi_{7,1} X_{1,1}$$

$$\Phi_{7,1} = \Phi_{7,6} \Phi_{6,5} \Phi_{5,4} \Phi_{4,3} \Phi_{3,2} \Phi_{2,1}$$

done, please see above

8. Make $M = 500$ runs of filter and estimate dynamics of mean-squared error of estimation over observation interval. Please calculate this error for filtered estimate of coordinate $x_{i,i}$ and its forecasting (extrapolation) m steps ahead $x_{i+m-1,i}$.

   Hint how to do:
   Calculate squared deviation of true coordinate x_i from its estimation $\hat{x}_{i,i}$ for every run over the whole observation interval N=200.

$$Error^{Run}(i) = \left(x_i - \hat{x}_{i,i}\right)^2$$

Run – number of run;

$i = 3, \dots, N$ - observation interval

(please start error calculation from step $i = 3$);

$Run = 1, \dots, M$ - number of runs;

Find average value of $Error^{Run}(i)$ over M runs for every step i and calculate its square root

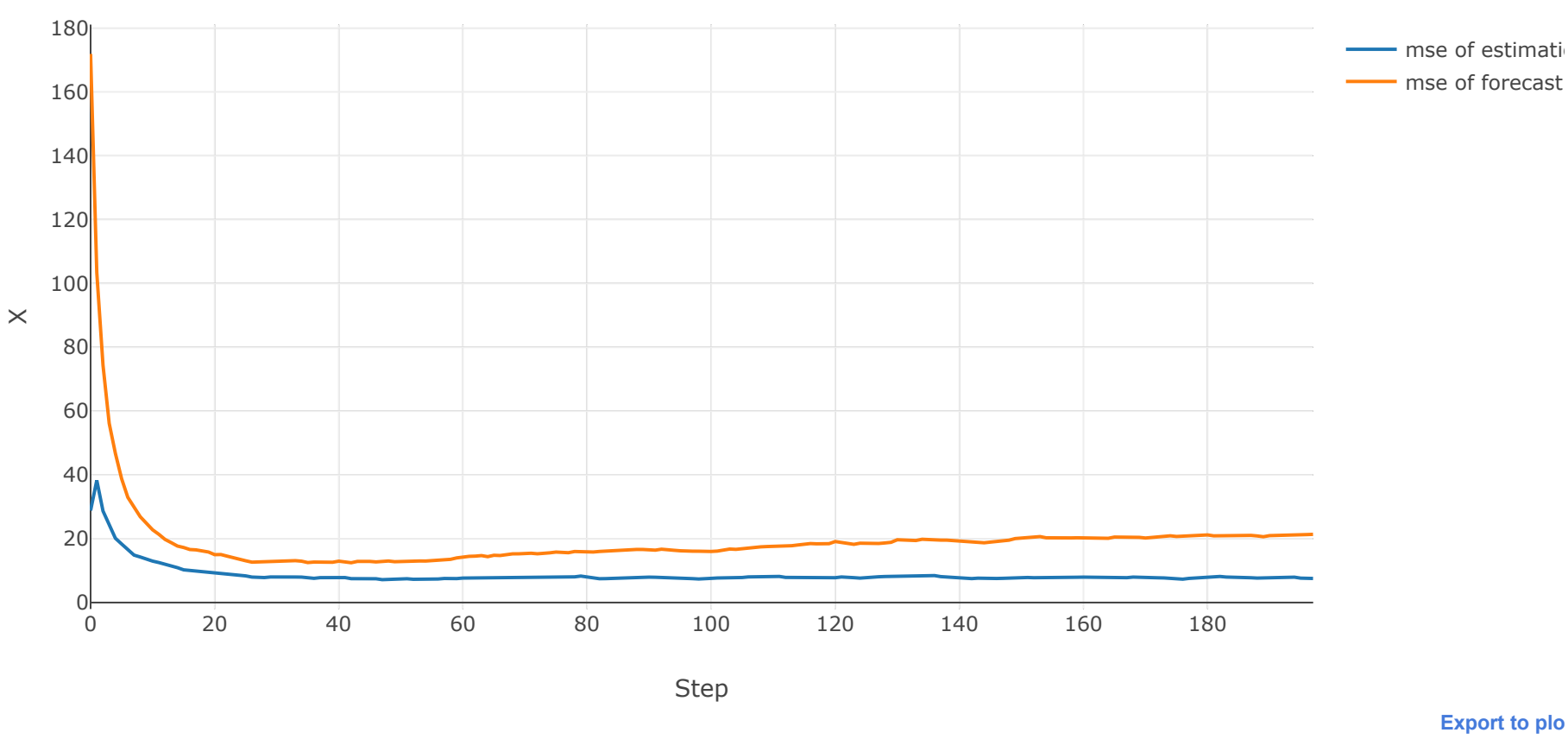
$$Final\_Error(i) = \sqrt{\frac{1}{M-1}\sum_{Run=1}^{M} Error^{Run}(i)}$$

Plot final error and check when it becomes almost constant and estimation accuracy doesn't increase anymore. At this moment filter becomes stationary and in practice this constant filter gain can be used in the algorithm instead of calculating filter gain at every time step.

In [49]:
```
def experiment(runs = 500, steps = 200, P_init = [[10000, 0], [0, 10000]], X_init = [2.0, 0], var_a = 0.2**2, var_n
    error_e = np.zeros(steps)
    error_f = np.zeros(steps)

    for i in range(runs):

        tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = var_a, steps = steps)
        measurments = measure(tr, var = var_n)
        X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments, P_init = P_init, var_a = var_

        error_e += (X_predicted - tr)**2
        error_f += (X7 - tr)**2

    error_e = np.sqrt(error_e/(runs-1))
    error_f = np.sqrt(error_f/(runs-1))

    return error_e, error_f
```

In [71]:
```
error_e, error_f = experiment()
```

In [72]:
```
data = [
    go.Scatter(
        y=error_e[2:],
        name='mse of estimation'
    ),
    go.Scatter(
        y=error_f[2:],
        name='mse of forecasting'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

9. Compare mean-squared error of filtered estimate of coordinate $x_{i,i}$ with standard deviation of measurement errors. Make conclusions about effectiveness of filtration.

Standart deviation of measurment errors is sqrt(var_n = 20**2) = 20. And MSE of filtered estimate is perturbating around value 8. Filtration is effective.

10. Make $M = 500$ runs again, but with more accurate initial filtration error covariance matrix

$$P_{0,0} = \begin{vmatrix} 100 & 0 \\ 0 & 100 \end{vmatrix}$$

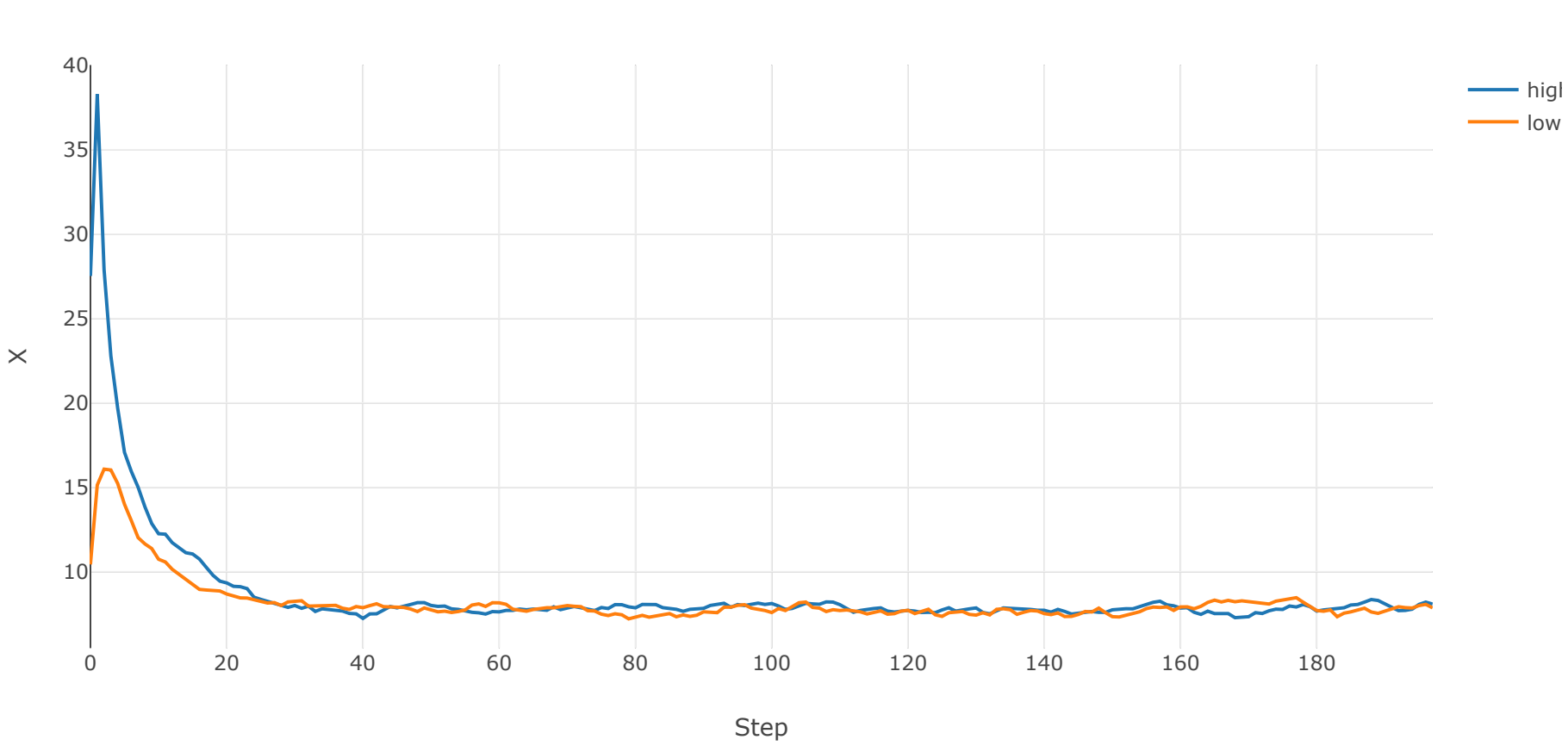
Calculate mean-squared error of filtered estimate of coordinate $x_{i,i}$ again as in item 8. Compare estimation results for both variants of initial $P_{0,0}$.

Please analyze how the accuracy of initial conditions $P_{0,0}$ affects the estimation results? When the choice of initial conditions doesn't affect the estimation results?

In [50]:
```
error_e_acc, error_f_acc = experiment(P_init = [[100, 0], [0, 100]])
```

With more accurate covariance matrix P MSE of estimations faster reach stable value.

In [54]:
```
data = [
    go.Scatter(
        y=error_e[2:],
        name='high P'
    ),
    go.Scatter(
        y=error_e_acc[2:],
        name='low P'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



11. Compare calculation errors of estimation $P_{i,i}$ provided Kalman filter algorithm with true estimation errors.

Hint how to do:

Make a plot of two curves:

a) Final error (true estimation error) obtained over $M = 500$ runs according to item 8.

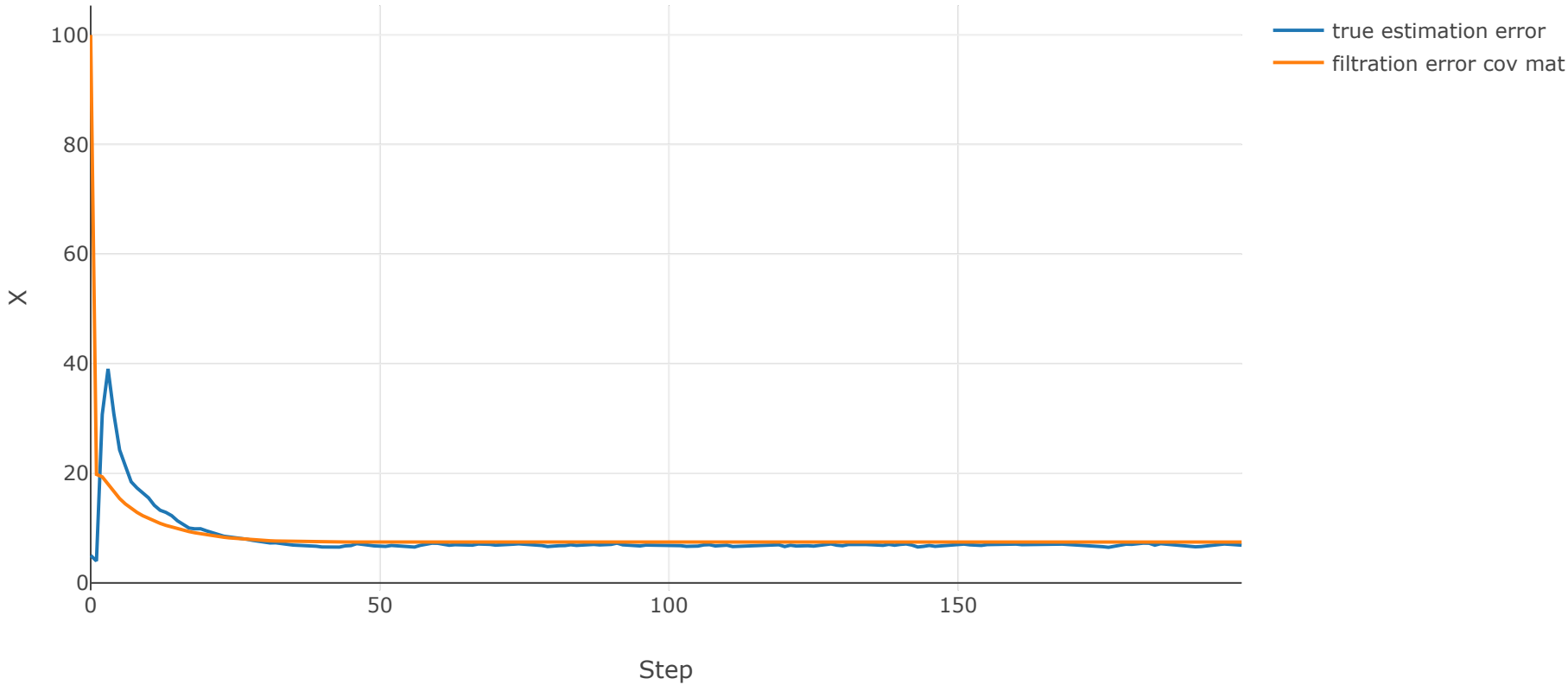
b) Filtration error covariance matrix $P_{i,i}$ (calculation error provided by Kalman filter)
Please use square root of the first diagonal element of $P_{i,i}$ that corresponds to standard deviation of estimation error of coordinate x_i. It doesn't depend on runs, for every run it is the same, as it depends only on model parameters Φ, H, Q, R.

Verify if calculation errors of estimation correspond to true estimation errors.

In [61]:
```
X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments)
```

square root of the first diagonal element of Pi,i corresponds to standard deviation of estimation error of coordinate xi.

In [63]:
```
data = [
    go.Scatter(
        y=error_e,
        name='true estimation error'
    ),
    go.Scatter(
        y=P_filtered,
        name='filtration error cov matr P'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```

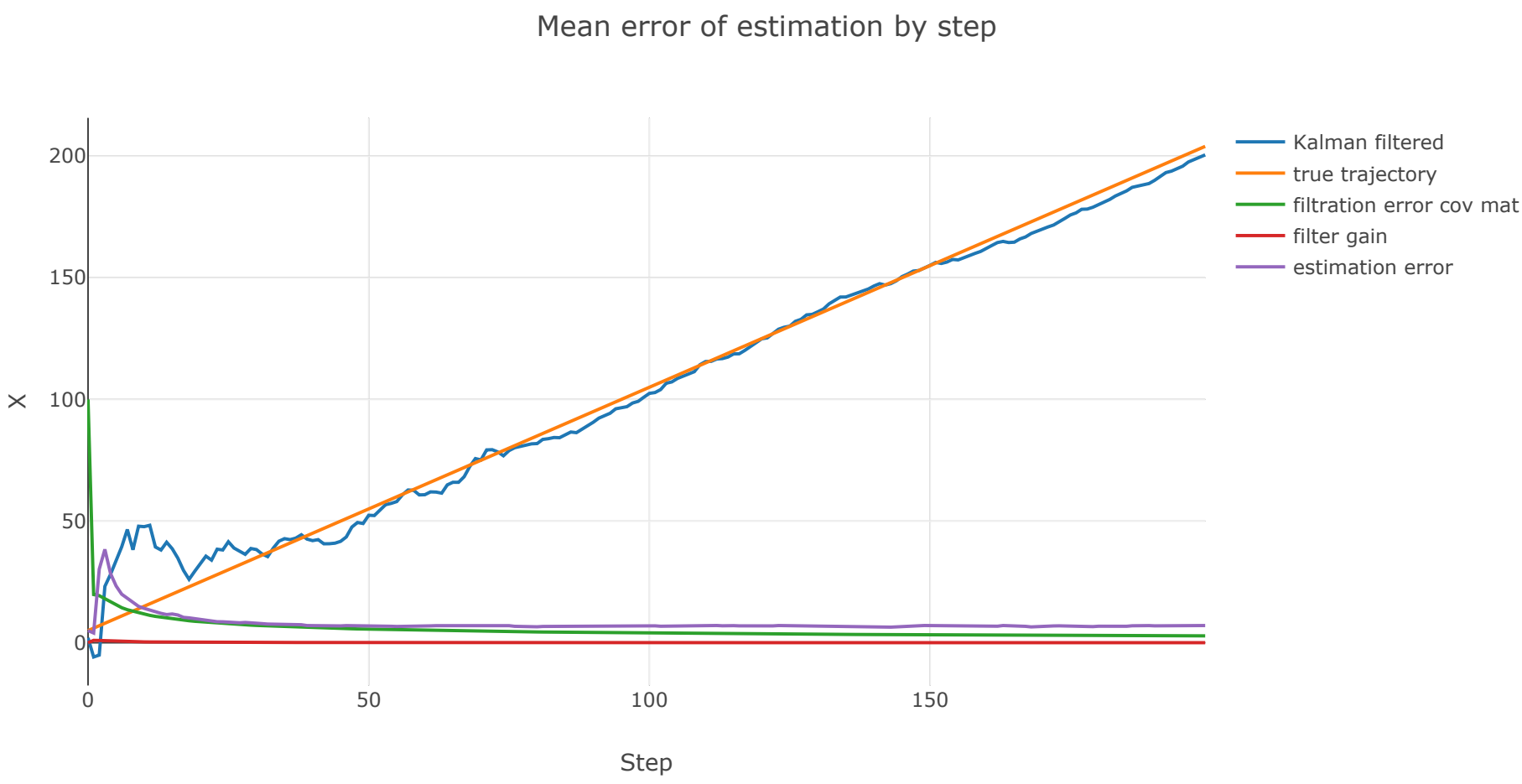


Calculation errors of estimation correspond to true estimation errors.

12. Run filter for deterministic trajectory (no random disturbance). It can be easily done by indicating in the code that variance of state noise equals to zero $\sigma_a^2 = 0$.
Make $M = 500$ runs and verify
1) Filter gain approaches to zero
2) Both true estimation errors defined according to item 8 and calculation errors $P_{i,i}$ (square root of the first diagonal element of $P_{i,i}$ that corresponds to standard deviation of estimation error of coordinate x_i) also approach to zero.

In [73]:
```
tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = 0, steps = 200)
measurments = measure(tr, var = 20**2) # array Nx1

X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments, var_a = 0)

error_e, error_f = experiment(var_a = 0)
```

In [80]:
```
data = [
    go.Scatter(
        y=X_filtered,
        name='Kalman filtered'
    ),
    go.Scatter(
        y=tr,
        name='true trajectory'
    ),
    go.Scatter(
        y=P_filtered,
        name='filtration error cov matr P'
    ),
    go.Scatter(
        y=K,
        name='filter gain'
    ),
    go.Scatter(
        y=error_e,
        name='estimation error'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

All three parameters indeed are approaching zero

This means that in conditions of motion without any random disturbances, estimation error approaches to zero and filter switches off from measurements (new measurements almost do not adjust estimates).

13. Verify what happens if you use deterministic model of motion, but in fact motion is disturbed by random acceleration. In other words you don't take into account the covariance matrix Q of state noise $w_i = Ga_i$ in assimilation algorithm.

Hint how to do
Generate a trajectory with variance of state noise $\sigma_a^2 = 0.2^2$
Use $Q = 0$ in Kalman filter algorithm.

a) Make $M = 500$ runs of filter and estimate dynamics of mean-squared error of estimation over observation interval. Please calculate this error for both filtered estimate of coordinate $x_{i,i}$ and its forecasting (extrapolation) m steps ahead $x_{i+m-1,i}$. Make conclusions about estimation in conditions of neglecting state noise in Kalman filter algorithm.
b) Compare calculation errors of estimation $P_{i,i}$ (only estimation error of coordinate x_i) provided Kalman filter algorithm with true estimation errors. Make conclusions.

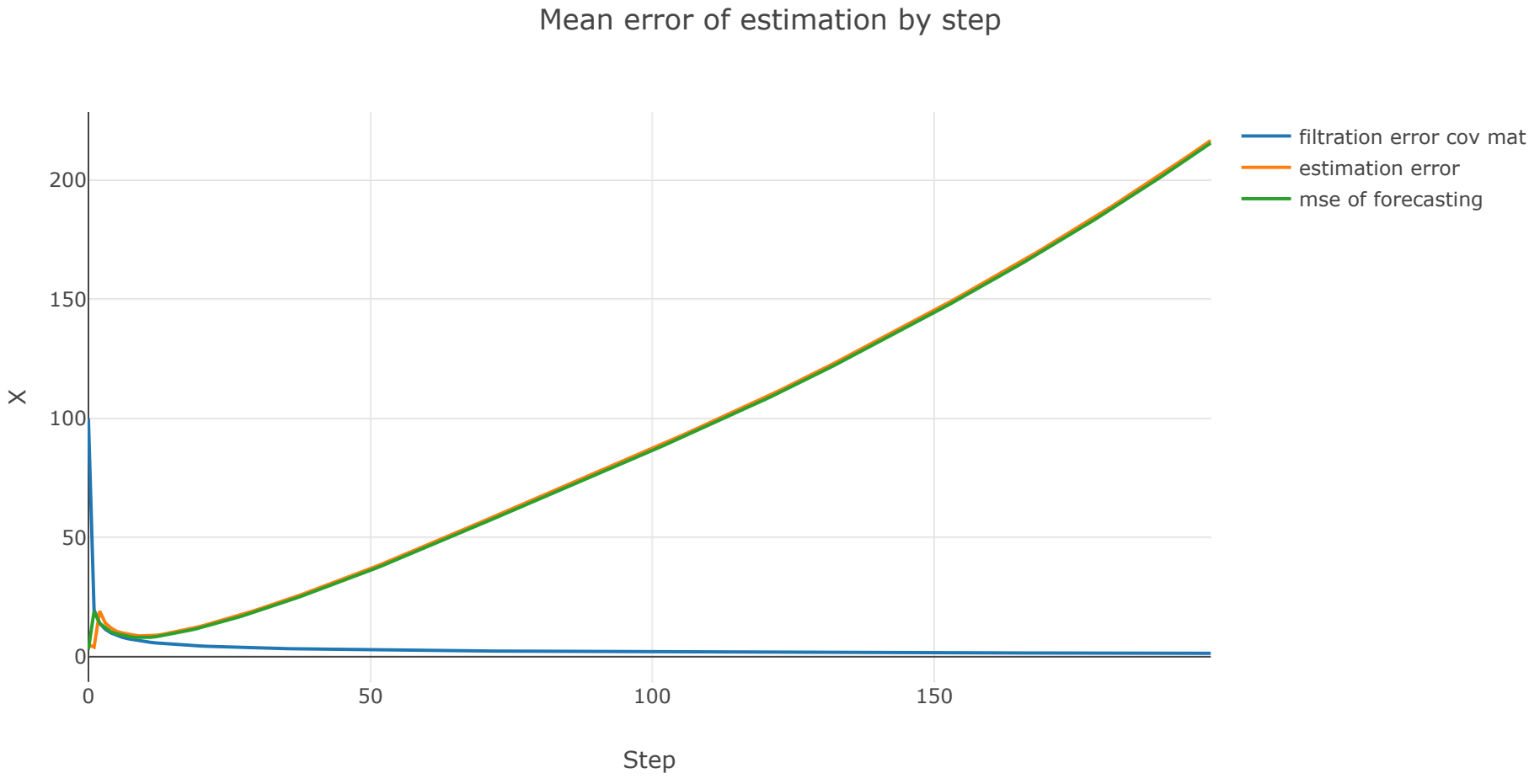
In [82]:
```
G = np.array([0, 0]).T
Q = G.dot(G.T.dot(0))
Q
```

Out[82]: 0

In [91]:
```
def experiment2(runs = 500, steps = 200, P_init = [[10000, 0], [0, 10000]], X_init = [2.0, 0], var_a = 0.2**2, var_
    error_e = np.zeros(steps)
    error_f = np.zeros(steps)

    for i in range(runs):

        tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = var_a, steps = steps)
        measurments = measure(tr, var = var_n)
        X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments, T = 0, P_init = P_init, var_a

        error_e += (X_predicted - tr)**2
        error_f += (X7 - tr)**2

    error_e = np.sqrt(error_e/(runs-1))
    error_f = np.sqrt(error_f/(runs-1))

    return error_e, error_f
```

In [92]:
```
error_e, error_f = experiment2()
```

In [95]:
```
data = [
    go.Scatter(
        y=P_filtered,
        name='filtration error cov matr P'
    ),
    go.Scatter(
        y=error_e,
        name='estimation error'
    ),
    go.Scatter(
        y=error_f,
        name='mse of forecasting'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



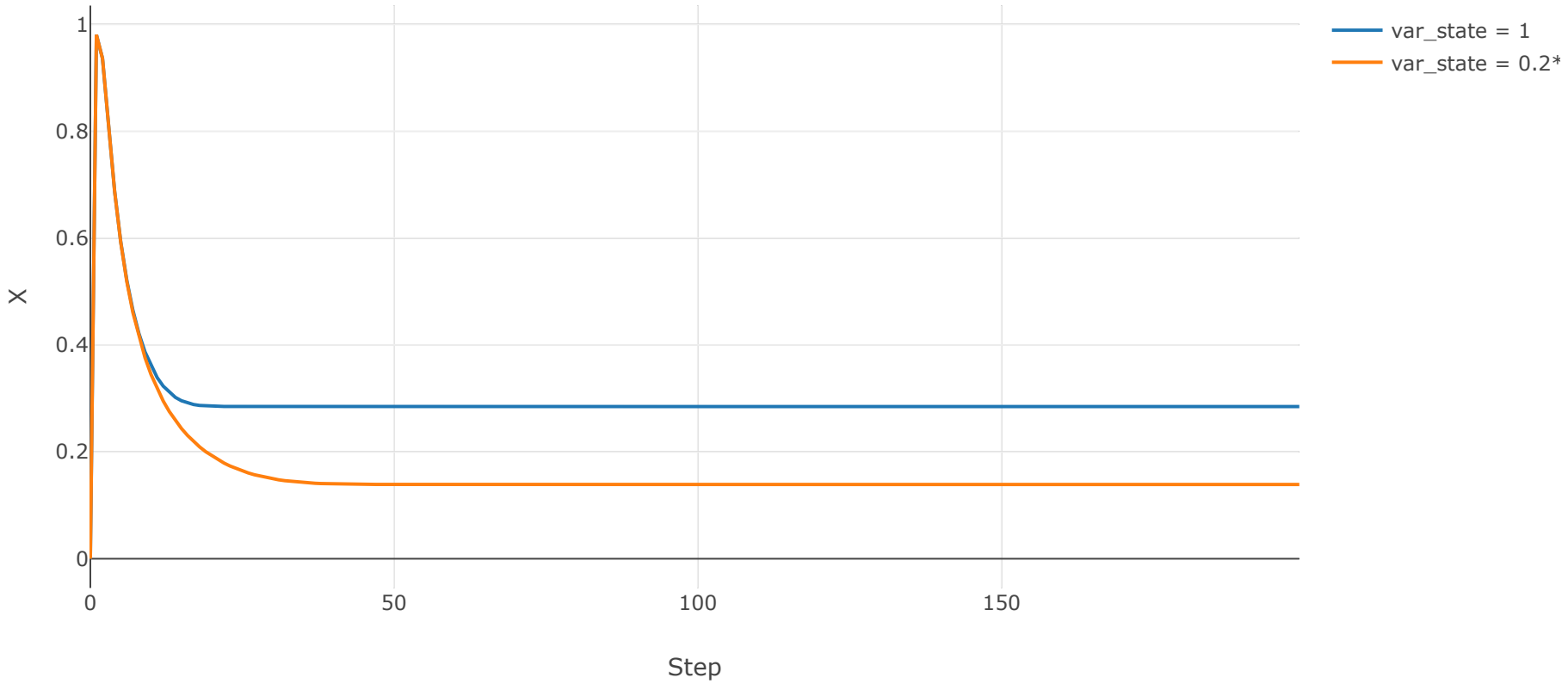
Estimation in conditions of neglecting state noise in Kalman filter algorithm leads to divergence. According to covariance matrix P, variance of coord xi approaches zero

14. Analyze how the relationship between state and measurement noise $\frac{\sigma_w^2}{\sigma_\eta^2}$ affect time when filter gain become almost constant and estimation accuracy doesn't increase anymore.

    Generate a trajectory with variance of state noise $\sigma_a^2 = 1$ and compare estimation results with that when $\sigma_a^2 = 0.2^2$. Make conclusions.

In [98]:
```
tr1 = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = 1, steps = 200)
measurments1 = measure(tr1, var = 20**2) # array Nx1
_, _, K1, _, _, _ = kalman(measurments1, var_a = 1)


tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = 0.2**2, steps = 200)
measurments = measure(tr, var = 20**2) # array Nx1
_, _, K, _, _, _ = kalman(measurments, var_a = 0.2**2)
```

In [99]:
```
data = [
    go.Scatter(
        y=K1,
        name='var_state = 1'
    ),
    go.Scatter(
        y=K,
        name='var_state = 0.2**2'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

With bigger variance of state noise Kalman gain becomes constant a little bit earlier. Why? That's an open question.

15. Analyze sensitivity of filter to underestimated non-optimal filter gain K

Generate a trajectory with variance of state noise $\sigma_a^2 = 0.2^2$.

Use initial filtered estimate $X_0 = \begin{vmatrix} 100 \\ 5 \end{vmatrix}$

Run filter for two conditions

a) Calculate optimal filter gain according to Kalman filter equations and calculate mean-squared error of filtered estimate of coordinate $x_{i.i}$ for $M = 500$ runs.

b) Run filter with underestimated filter gain K.

   Use steady-state value of optimal filter gain divided by 5 $K = \frac{K_{steady-state}}{5}$

   Calculate mean-squared error of filtered estimate of coordinate $x_{i.i}$ for in this conditions for $M = 500$ runs and compare with estimation results obtained in (a).

In [101]:
```
tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = 0.2**2, steps = 200)
measurments = measure(tr, var = 20**2) # array Nx1
X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman(measurments, X_init = [100.0, 5.0], var_a = 0.2**2
```

In [104]:
```
K[-1]
```

Out[104]:
```
0.13895150388783037
```

In [116]:
```
error_e, error_f = experiment()
```

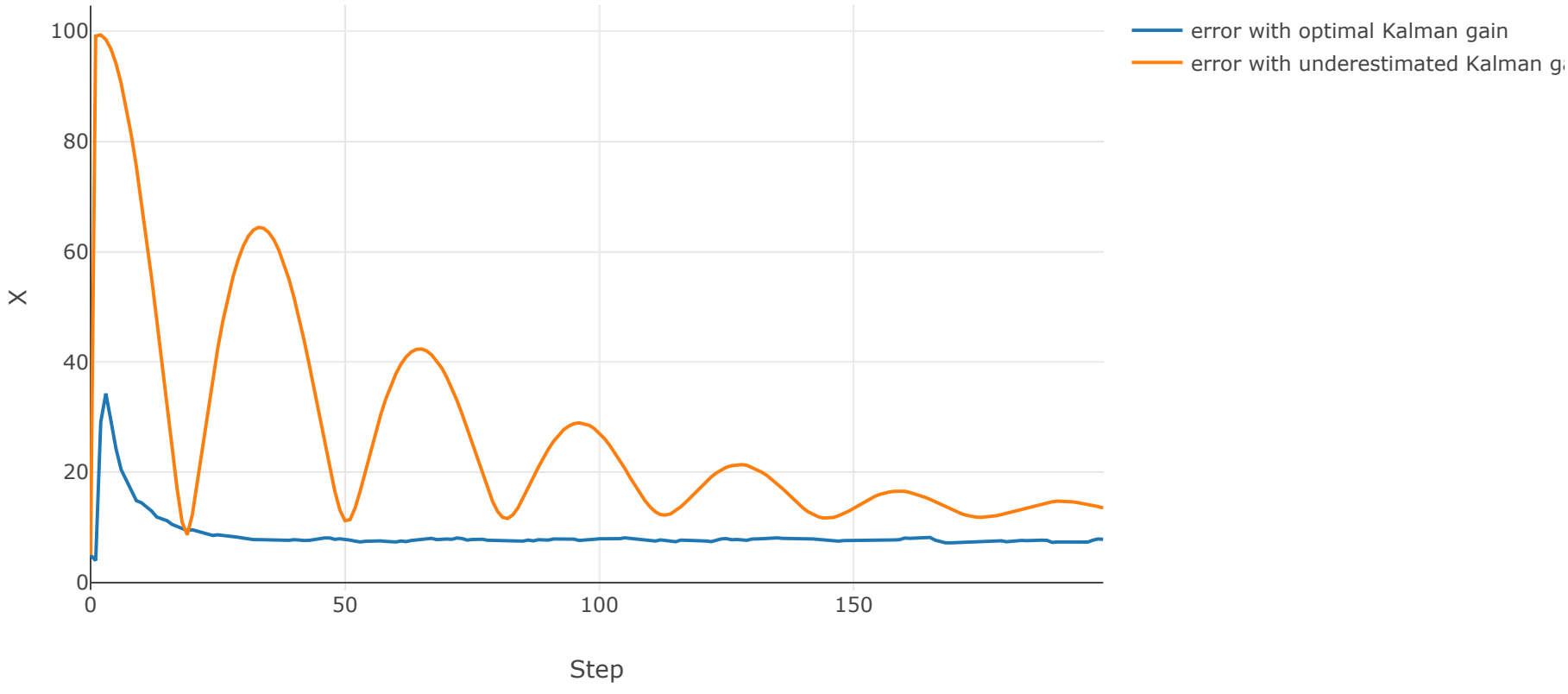
In [131]:
```
def kalman_under(measr, T = 1, X_init = [2.0, 0], P_init = [[10000, 0], [0, 10000]], var_a = 0.2**2, var_n = 20**2)

    size = (len(measr), 2) # size of array
    G = np.array([T**2/2, T]).T
    Q = G.dot(G.T.dot(var_a))
    R = var_n
    F = np.array([[1, T], [0, 1]])
    H = np.array([1, 0])
    X = np.zeros(size)
    X_ = np.zeros(size)
    P = np.zeros((size[0], 2, 2))
    P_ = np.zeros((size[0], 2, 2))
    #K = np.zeros(size)
    K_ = 0.139/5
    K = np.array([K_, 0.01])

    X_[0] = X_init
    P_[0] = P_init

    for k in range(1, size[0]):

        # Prediction
        X[k] = np.dot(F, X_[k-1])
        P[k] = np.dot(F, np.dot(P_[k-1], F.T)) + Q

        # Update
        #K = np.dot(P[k], H.T)/(np.dot(H, np.dot(P[k], H.T)) + R)
        X_[k] = X[k] + K*(measr[k] -  np.dot(H, X[k]))
        P_[k] = np.dot((np.eye(2) - np.outer(K, H)), P[k])

    return X[:,0], X_[:,0], K, np.sqrt(P[:,0,0]), np.sqrt(P_[:,0,0]), np.linalg.matrix_power(F,7).dot(X_.T)[0]
```

In [132]:
```
def experiment3(runs = 500, steps = 200, P_init = [[10000, 0], [0, 10000]], X_init = [2.0, 0], var_a = 0.2**2, var_
    error_e = np.zeros(steps)
    error_f = np.zeros(steps)

    for i in range(runs):

        tr = generate_trajectory(X0 = 5, V0 = 1, T = 1, mean = 0, var = var_a, steps = steps)
        measurments = measure(tr, var = var_n)
        X_predicted, X_filtered, K, P_predicted, P_filtered, X7 = kalman_under(measurments, X_init = [100.0, 5.0],

        error_e += (X_predicted - tr)**2
        error_f += (X7 - tr)**2

    error_e = np.sqrt(error_e/(runs-1))
    error_f = np.sqrt(error_f/(runs-1))

    return error_e, error_f
```

In [133]:
```
error_e_under, error_f_under = experiment3()
```

```
/Users/nickzherdev/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:31: RuntimeWarning:

invalid value encountered in sqrt
```

In [134]:
```
data = [
    go.Scatter(
        y=error_e,
        name='error with optimal Kalman gain'
    ),
    go.Scatter(
        y=error_e_under,
        name='error with underestimated Kalman gain'
    ),
]

layout= go.Layout(
    title= 'Mean error of estimation by step',
    xaxis= dict(
        title= 'Step',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

Conclusions

Kalman filter provides estimates of variable parameters (x, V), when LSM – of constant variables. It is sensitive to estimation of optimal filter gain.