Laboratory work 3 - Determining and removing drawbacks of exponential and running mean. Task 1

Nikolay Zherdev, Dmitry Baluev, Carolina Latserus

Skoltech, 08.10.2018

The objective of this laboratory work is to determine conditions for which broadly used methods of running and exponential mean provide effective solution and conditions under which they break down. Important outcome of this exercise is getting skill to choose the most effective method in conditions of uncertainty.

Backward exponential smoothing - is used so the target would reflect only future %name% behavior, not past action that would induce spurious correlation.

Backward exponential smoothing

1. In lab 2 (part II) you have already applied running and exponential mean to random walk model with noise statistics $\sigma_w^2 = 28^2, \sigma_\eta^2 = 97^2$. In this conditions results of exponential smoothing demonstrated significant shift (delay) of estimations.
2. Please apply backward exponential smoothing to forward exponential estimates to further smooth measurement errors.
3. Make visual comparison of results. Plot true trajectory X_i, measurements z_i, running and backward exponential mean. Make conclusions which method provides better accuracy. Compare estimation results of running mean and backward exponential smoothing using deviation and variability indicators (Lab2_Short_discussion_October_5_2017.pdf).

In [437]:
```
%matplotlib inline
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
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure


init_notebook_mode(connected=True)
```

In [2]:
```
def generate_trajectory(x0 = 10, mean = 0, var = 0.1, steps = 100):
    trajectory = np.random.normal(loc = mean, scale = np.sqrt(var), size = steps)
    trajectory[0] = x0
    return np.cumsum(trajectory)
```

In [3]:
```
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)
```

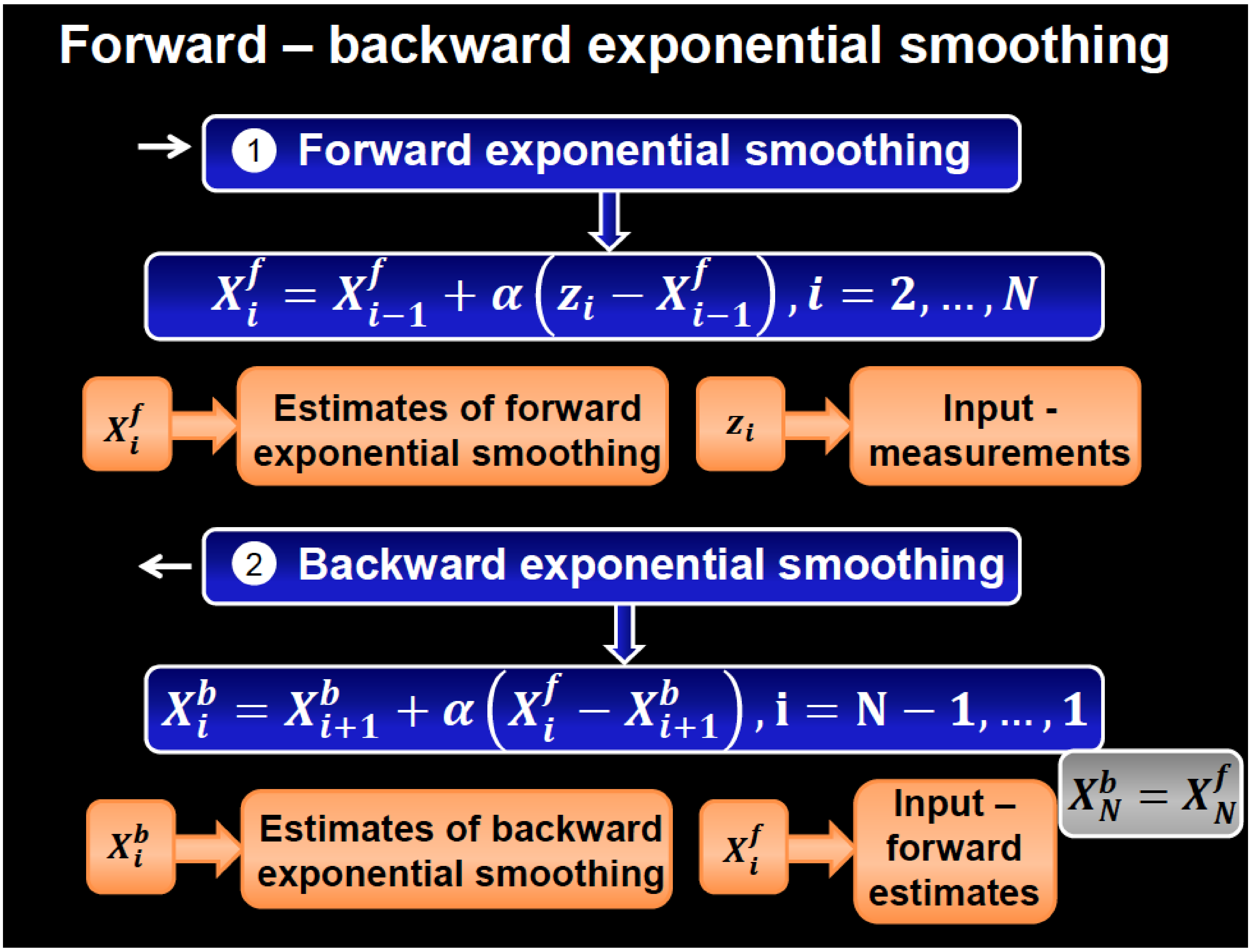
In [5]:
```
def exp_smooth(z, alpha):
    X = np.zeros(z.shape)
    X[0] = z[0]
    for i in range(1, z.shape[0]):
        X[i] = X[i-1] + alpha*(z[i] - X[i-1])
    return X
```

In [6]:
```
trajectory = generate_trajectory(x0 = 10, var = 28**2, steps = 300)
```

In [67]:
```
measurments = measure(trajectory, var = 97**2)
```

In [8]:
```
alpha = 0.25
```

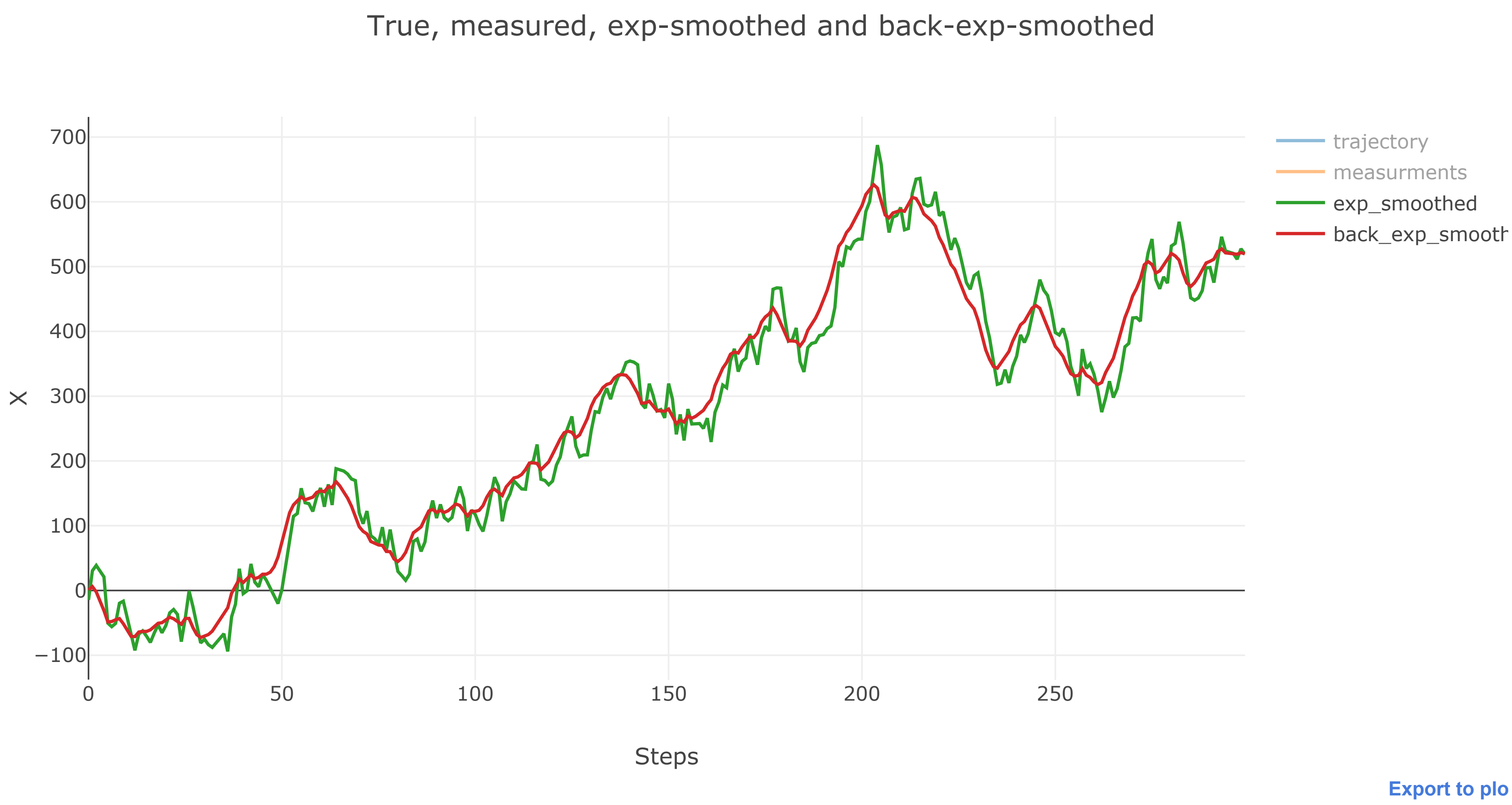
In [10]:
```
exp_smoothed = exp_smooth(measurment, alpha)
```



In [62]:
```
def back_exp_smooth(exp_smoothed, alpha):
    B = np.zeros(exp_smoothed.shape[0])
    B[-1] = exp_smoothed[-1]
    for i in reversed(range(exp_smoothed.shape[0]-1)):
        B[i] = B[i+1] + alpha*(exp_smoothed[i] - B[i+1])
    return B
```

In [63]:
```
back_exp_smoothed = back_exp_smooth(exp_smoothed, alpha)
```

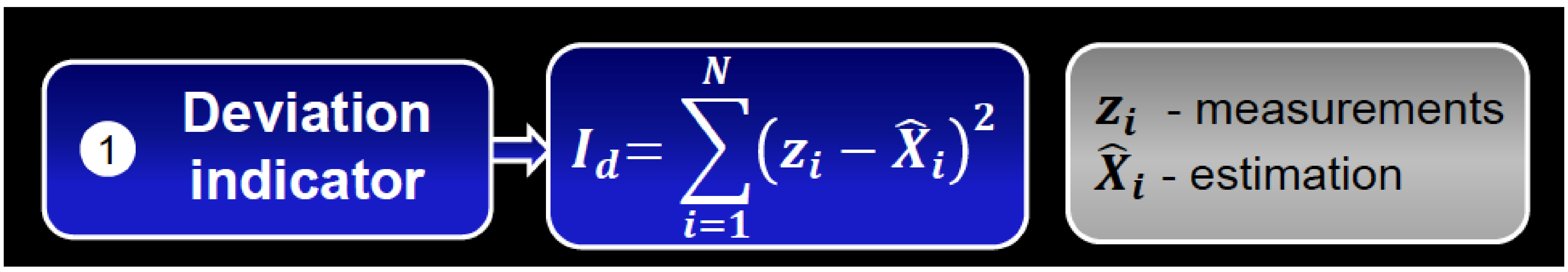
In [69]:
```
data = [
    go.Scatter(
        y=trajectory,
        name='trajectory'
    ),
    go.Scatter(
        y=measurments,
        name='measurments'
    ),
    go.Scatter(
        y=exp_smoothed,
        name='exp_smoothed'
    ),
    go.Scatter(
        y=back_exp_smoothed,
        name='back_exp_smoothed'
    ),
]

layout= go.Layout(
    title= 'True, measured, exp-smoothed and back-exp-smoothed',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



By visual comparison it is clear, that backward exponential smoothing (BES) provides us with estimates that are closer to real trajectory. Curve gained with BES doesn't have delay, that we can see in forward exponential estimates. But it is less responsible to sudden changes. Still, it looks like BES provides better accuracy.

Compare estimation results of running mean and backward exponential smoothing using deviation and variability indicators

In [72]:
```
M = 7
running_mean = pd.Series(measurments).rolling(window = M, center = True).mean()
```



In [171]:
```
def deviation_indicator(measurments, estimations):
    error = measurments - estimations
    return np.mean(error**2)
```

In [172]:
```
running_mean_dev_ind = deviation_indicator(measurments, running_mean)
exp_smooth_dev_ind = deviation_indicator(measurments, exp_smoothed)
back_exp_smooth_dev_ind = deviation_indicator(measurments, back_exp_smoothed)
```

In [174]:
```
running_mean_dev_ind, back_exp_smooth_dev_ind
```

Out[174]:
```
(2371.367784665608, 1896.3446528470504)
```

2 Variability indicator

$$I_v = \sum_{i=1}^{N-2} \left(\hat{X}_{j+2} - 2\hat{X}_{j+1} + \hat{X}_j\right)^2$$

$\hat{X}_i$ - estimation

In [130]:
```
#RM_variability = np.sum((np.add(np.subtract(running_mean[5:-3], 2*running_mean[4:-4]), running_mean[3:-5]))**2)
```

In [380]:
```
def running_mean_variability(sequence, M):
    X_j2 = sequence[int(M/2)+2:-int(M/2)]
    X_j1 = sequence[int(M/2)+1:-(int(M/2)+1)]
    X_j = sequence[int(M/2):-(int(M/2)+2)]
    RM_variability = np.sum((np.add(np.subtract(X_j2, 2*X_j1), X_j))**2)
    return RM_variability
```

In [381]:
```
RM_variability = running_mean_variability(running_mean, M)
```

In [371]:
```
def exp_smooth_variability(sequence):
    return np.sum((sequence[2:] - 2*sequence[1:-1] + sequence[:-2])**2)
```

In [383]:
```
BES_variability = exp_smooth_variability(back_exp_smoothed)
```

In [376]:
```
RM_variability, BES_variability
```

Out[376]:
```
(112.72842413029761, 15851.283484420555)
```

First we will analyze a process which rate of change is changed insignificantly and measurement noise is great.

First trajectory

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 300 points.
Initial conditions: $X_1 = 5; V_1 = 0; T = 0.1$
Variance of noise a_i, $\sigma_a^2 = 10$

In [384]:
```
# 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 [394]:
```
accel_motion = generate_trajectory(X0 = 5, V0 = 0, var = 10, steps = 300)
```

Generate measurements z_i of the process X_i

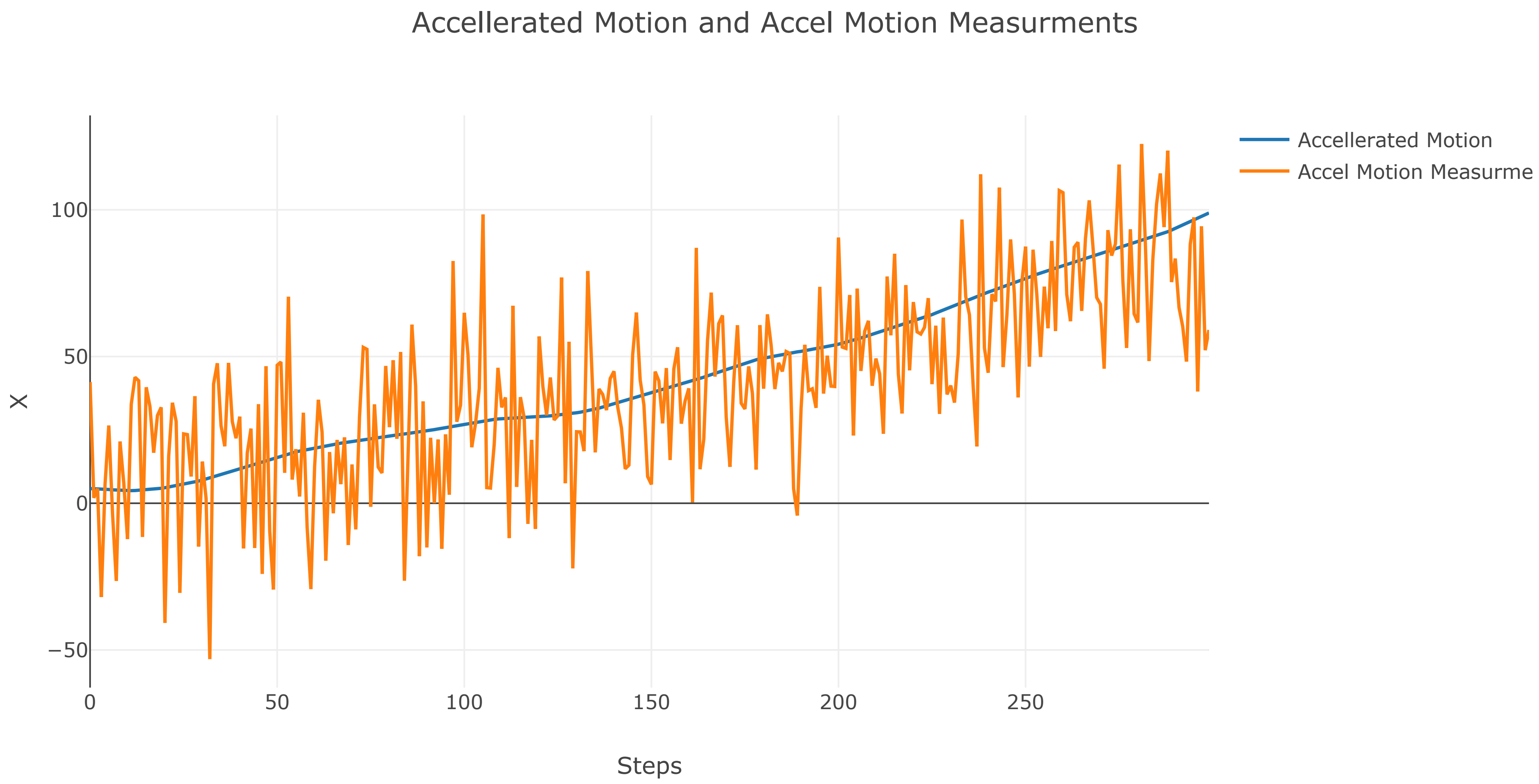
$$z_i = X_i + \eta_i$$

η_i –normally distributed random noise with zero mathematical expectation and variance $\sigma_\eta^2 = 500$.

In [387]:
```
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 [388]:
```
AccMot_measurments = measure(accel_motion, var = 500)
```

In [404]:
```
data = [
    go.Scatter(
        y=accel_motion,
        name='Accellerated Motion'
    ),
    go.Scatter(
        y=AccMot_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)
```



Export to plo

2. Determine empirically the window size M of running mean and smoothing coefficient α (forward exponential smoothing) that provide the best estimation of the process X_i using measurements z_i. As this process is not random walk model you cannot apply equations for optimal smoothing coefficient.
   Hint: the trajectory is close to the line, but measurement errors are huge. Then which window width is better in this case?

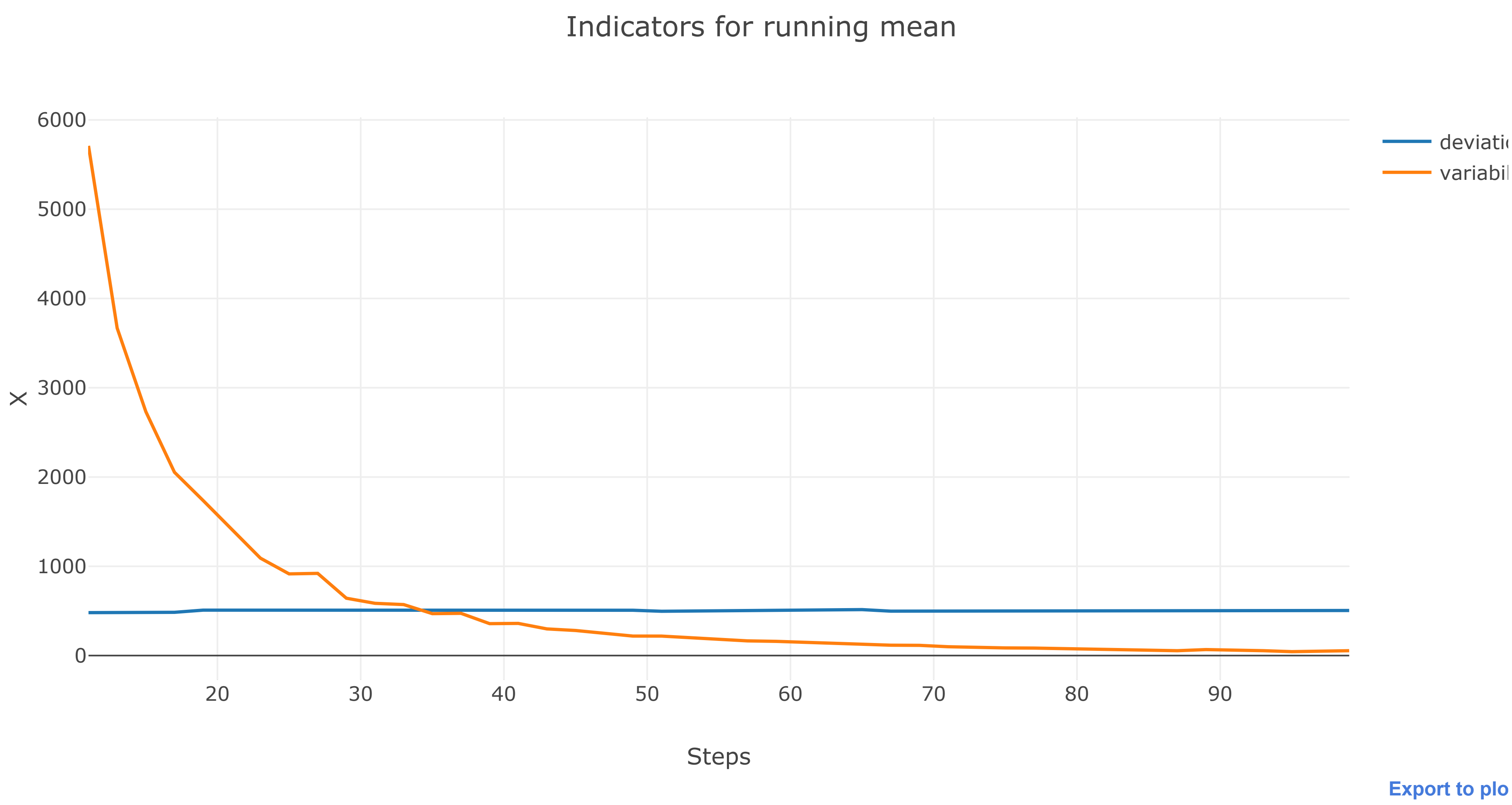
3. Chose better smoothing method using deviation and variability indicators.

Determine empirically the window size M of running mean that provide the best estimation of the process Xi using measurements zi

In [464]:
```
RM_dev_ind_dict = {}
RM_variability_dict = {}

for M in range(11, 100, 2):
    running_mean = pd.Series(AccMot_measurments).rolling(window = M, center = True).mean()
    RM_error = running_mean - accel_motion # running mean
    RM_var = np.var(RM_error, ddof = 1)
    running_mean_dev_ind = deviation_indicator(AccMot_measurments, running_mean)
    RM_dev_ind_dict.update({M:running_mean_dev_ind})
    RM_variability = running_mean_variability(running_mean, M)
    RM_variability_dict.update({M:RM_variability})
#     print("M is ", M)
#     print("I-deviation is", float("{:.2f}".format(running_mean_dev_ind)))
#     print("I-variability is", float("{:.2f}".format(RM_variability)))
#     print("variance is", float("{:.2f}".format(RM_var)))
#     print()


# RM_dev_ind_dict = pd.DataFrame(data=RM_dev_ind_dict, index=[0])
# RM_variability_dict = pd.DataFrame(data=RM_variability_dict, index=[0])
```

In [459]:
```
#pd.Series(RM_dev_ind_dict).to_frame()
#pd.Series(RM_variability_dict).to_frame()
```

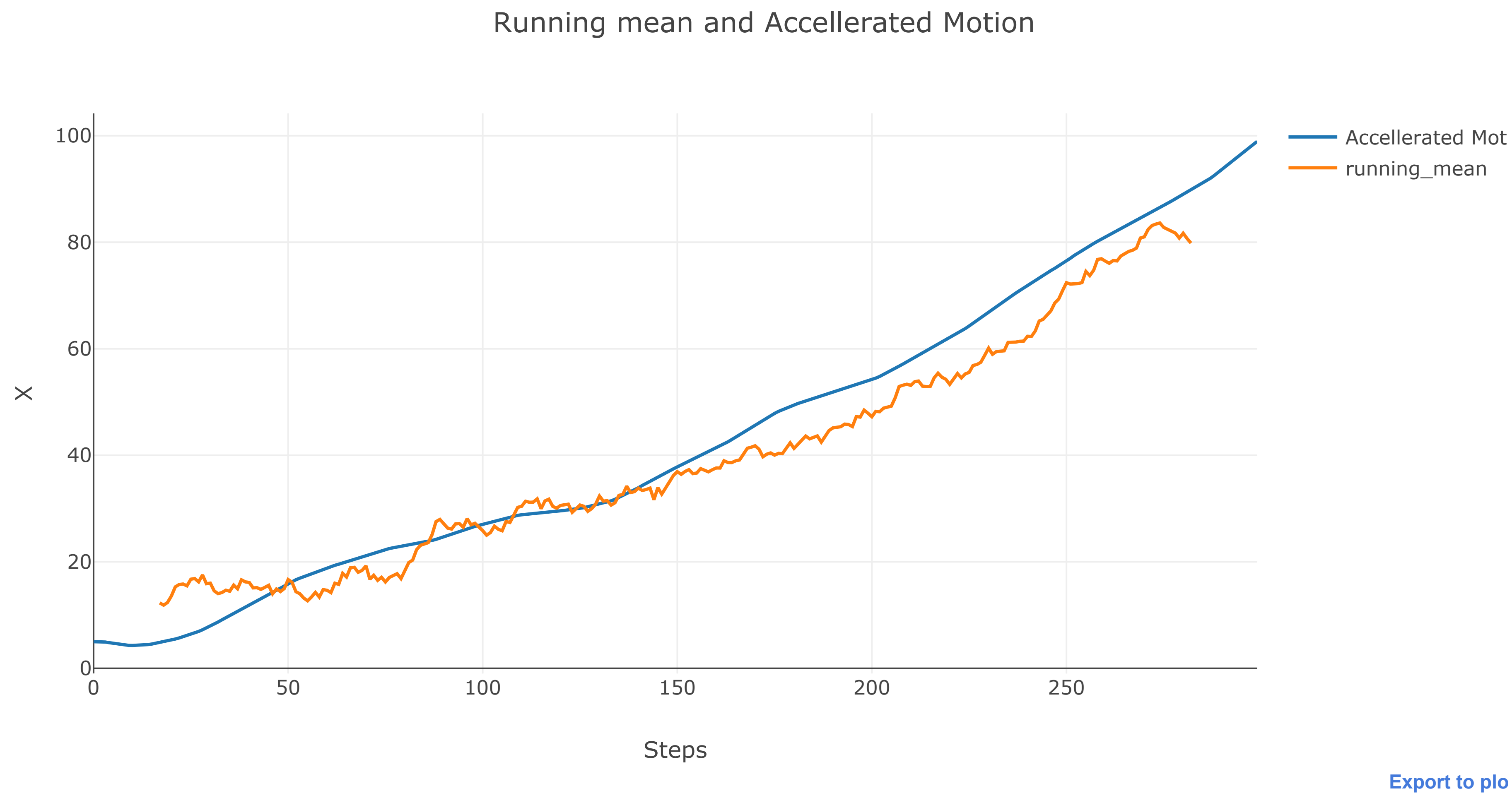
In [476]:
```
data = [
    go.Scatter(
        y = list(RM_dev_ind_dict.values()),
        x = list(RM_dev_ind_dict.keys()),
        name = 'deviation'
    ),
    go.Scatter(
        y=list(RM_variability_dict.values()),
        x = list(RM_variability_dict.keys()),
        name='variability'
    ),
]

layout= go.Layout(
    title= 'Indicators for running mean',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



The window sise, that provides the best results is about 35, if my calculations are correct.

In [472]:
```
running_mean = pd.Series(AccMot_measurments).rolling(window = 35, center = True).mean()
running_mean_dev_ind = deviation_indicator(AccMot_measurments, running_mean)
```

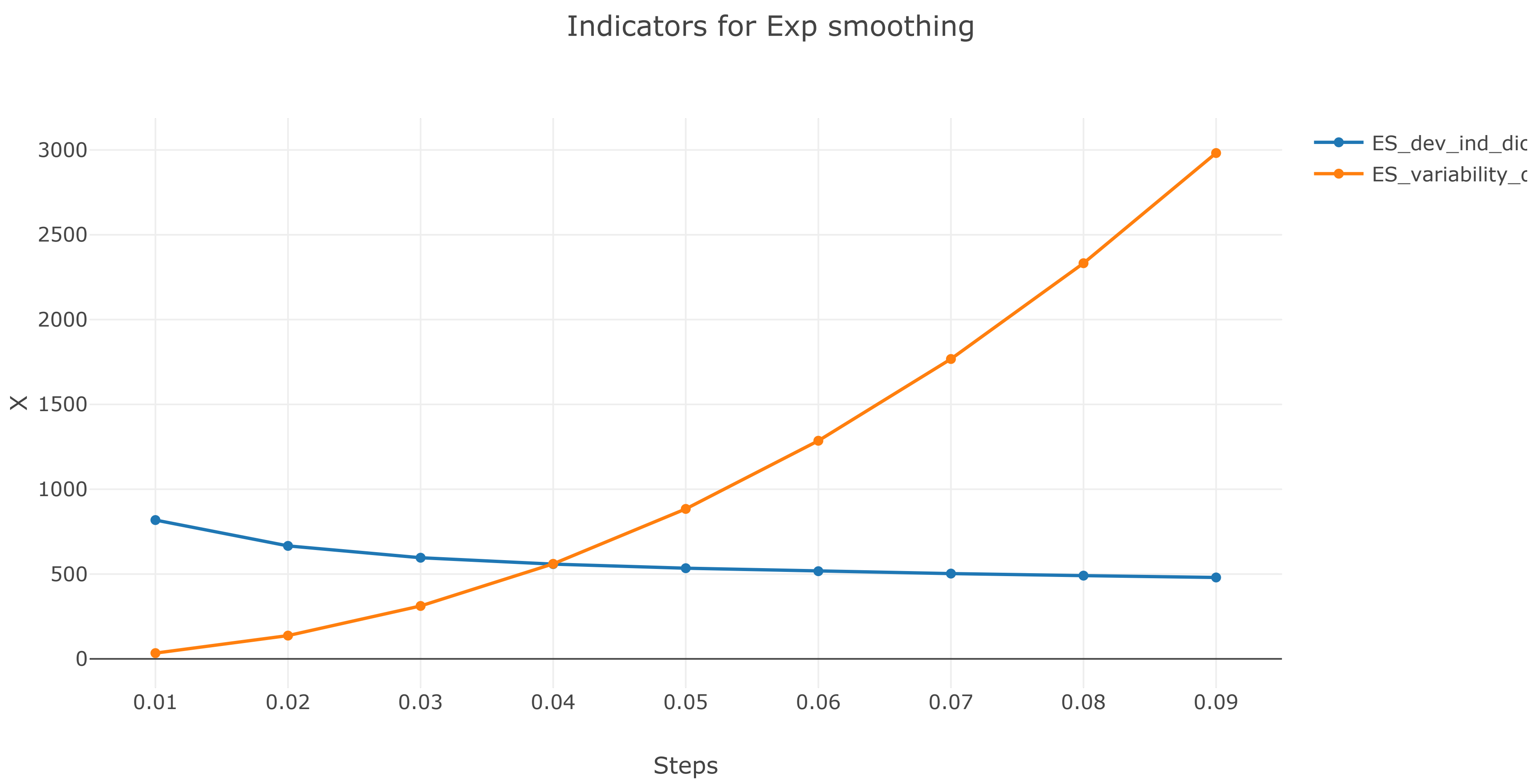
In [473]:
```
data = [
    go.Scatter(
        y=accel_motion,
        name='Accellerated Motion'
    ),
#     go.Scatter(
#         y=AccMot_measurments,
#         name='Accel Motion Measurments'
#     ),
    go.Scatter(
        y=running_mean,
        name='running_mean ' # + str(M)
    )
]

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



Determine empirically smoothing coefficient α (forward exponential smoothing) that provides the best estimation of the process Xi using measurements zi

In [470]:
```
ES_dev_ind_dict = {}
ES_variability_dict = {}

#a = np.arange(0, 1, 0.05)
b = np.arange(0.01, 0.1, 0.01)
for alpha in b:
    exp_smoothed = exp_smooth(AccMot_measurments, alpha)
    ES_error = exp_smoothed - accel_motion # exp smoothing
    ES_var = np.var(ES_error, ddof = 1)
    exp_smooth_dev_ind = deviation_indicator(AccMot_measurments, exp_smoothed)
    ES_variability = exp_smooth_variability(exp_smoothed)

    ES_dev_ind_dict.update({alpha:exp_smooth_dev_ind})
    ES_variability_dict.update({alpha:ES_variability})

#     print("alpha is ", float("{:.2f}".format(alpha)))
#     print("I-deviation is", float("{:.2f}".format(exp_smooth_dev_ind)))
#     print("I-variablity is", float("{:.2f}".format(ES_variability)))
#     print("variance is", float("{:.2f}".format(ES_var)))
#     print()
```

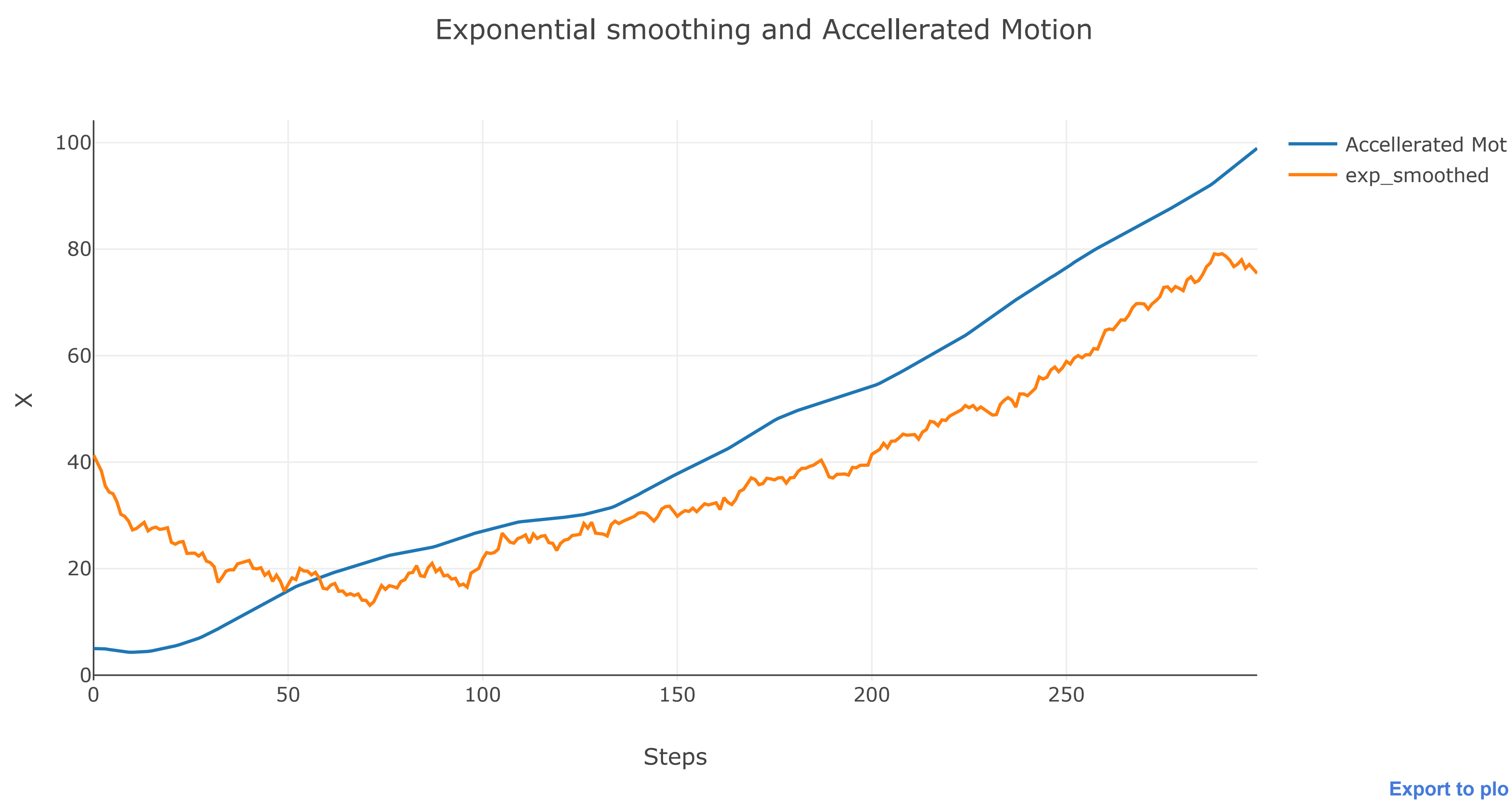
In [471]:
```
data = [
    go.Scatter(
        y = list(ES_dev_ind_dict.values()),
        x = list(ES_dev_ind_dict.keys()),
        name = 'ES_dev_ind_dict'
    ),
    go.Scatter(
        y=list(ES_variability_dict.values()),
        x = list(ES_variability_dict.keys()),
        name='ES_variability_dict'
    ),
]

layout= go.Layout(
    title= 'Indicators for Exp smoothing',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

In [474]:
```
exp_smoothed = exp_smooth(AccMot_measurments, 0.04)
```

In [475]:
```
data = [
    go.Scatter(
        y=accel_motion,
        name='Accellerated Motion'
    ),
#     go.Scatter(
#         y=AccMot_measurments,
#         name='Accel Motion Measurments'
#     ),
    go.Scatter(
        y=exp_smoothed,
        name='exp_smoothed'
    )
]

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



Optimal coef alpha for exponential smoothing is 0.04.

Looks like running mean performs better with this magnitude of measurment noise.

Second trajectory

4. Generate cyclic trajectory X_i according to the equation

$$X_i = A_i \cdot \sin(\omega i + 3)$$
$$A_i = A_{i-1} + w_i$$

Periods of oscillations is T=32 steps.
Hint: To determine period, please define corresponding angle frequency ω from equation $\omega T = 2\pi$ (radian per one step).

w_i – normally distributed random noise with zero mathematical expectation and variance $\sigma_w^2 = 0.08^2$.

Size of trajectory is 200 points.
Initial conditions: $A_1 = 1$.

5. Generate measurements z_i of the process X_i

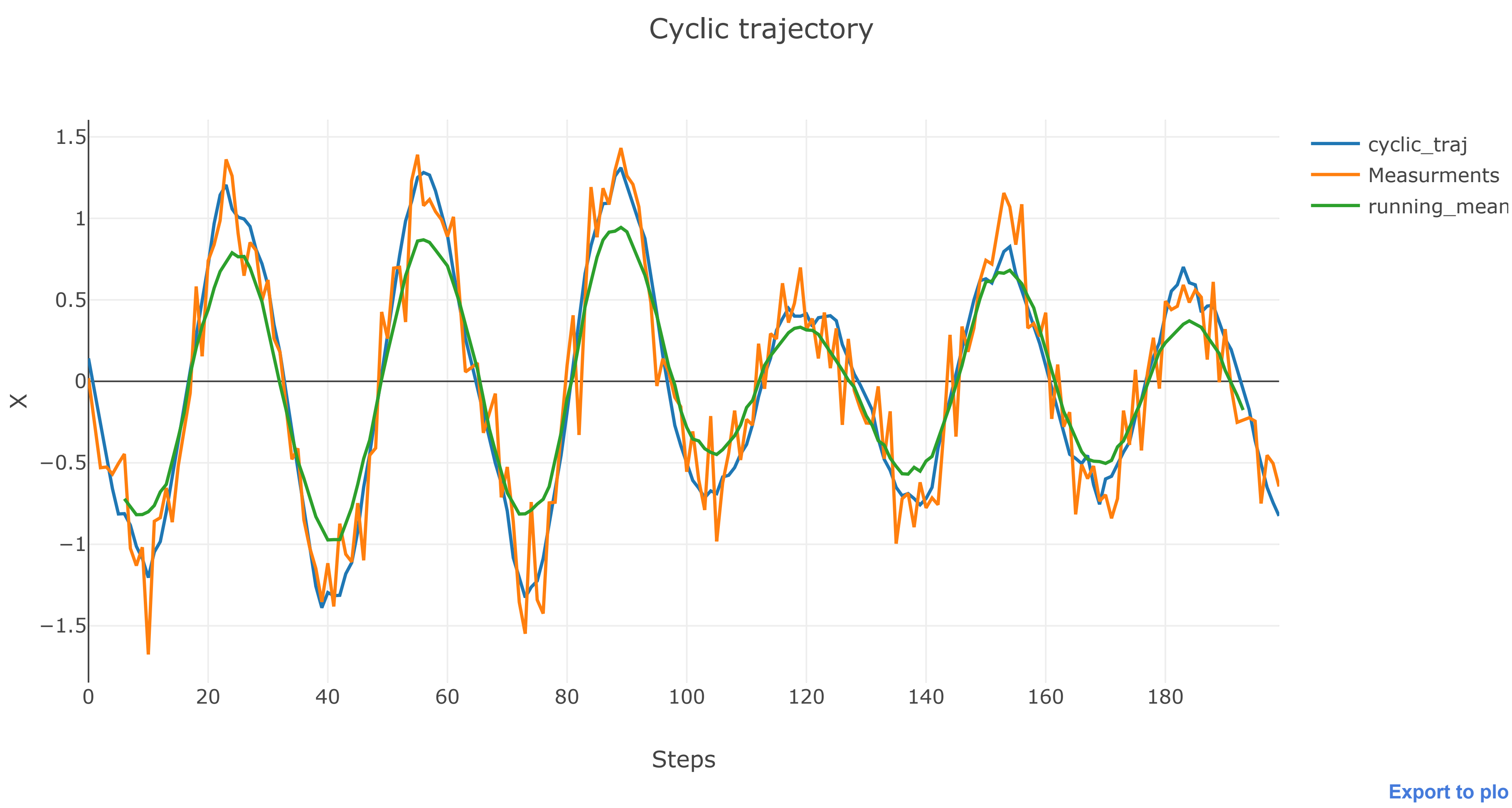
$$z_i = X_i + \eta_i$$

η_i –normally distributed random noise with zero mathematical expectation and variance $\sigma_\eta^2 = 0.05$

In [481]:
```
def generate_trajectory(A0, mean = 0, T = 32, var = 0.1, steps = 200):
    i = np.arange(0, 200)
    w = np.random.normal(loc = mean, scale = np.sqrt(var), size = steps)
    w[0] = A0
    ampl = np.cumsum(w)
    X = np.multiply(ampl, np.sin((2*np.pi/T)*i + 3))
    return X
```

In [498]:
```
cyclic_traj = generate_trajectory(A0 = 1, T = 32, var = 0.08**2, steps = 200)
CycTraj_measurments = measure(cyclic_traj, var = 0.05)
cycl_running_mean13 = pd.Series(CycTraj_measurments).rolling(window = 13, center = True).mean()
```

In [499]:
```
data = [
    go.Scatter(
        y=cyclic_traj,
        name='cyclic_traj'
    ),
    go.Scatter(
        y=CycTraj_measurments,
        name='Measurments'
    ),
    go.Scatter(
        y=cycl_running_mean13,
        name='running_mean13'
    ),

]

layout= go.Layout(
    title= 'Cyclic trajectory',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



6. Apply running mean with window size $M = 13$ to measurements z_i.

7. Determine the period of oscillations for which running mean with given for every group window size M
   a) produces inverse oscillations
   b) leads to the loss of oscillations (zero oscillations)
   c) changes the oscillations insignificantly

   Group 1: $M = 15$; Group 2: $M = 17$; Group 3: $M = 19$; Group 4: $M = 21$; Group 5: $M = 23$; Group 6: $M = 25$; Group 7: $M = 27$;

8. Make conclusions about conditions of 7a,b,c.

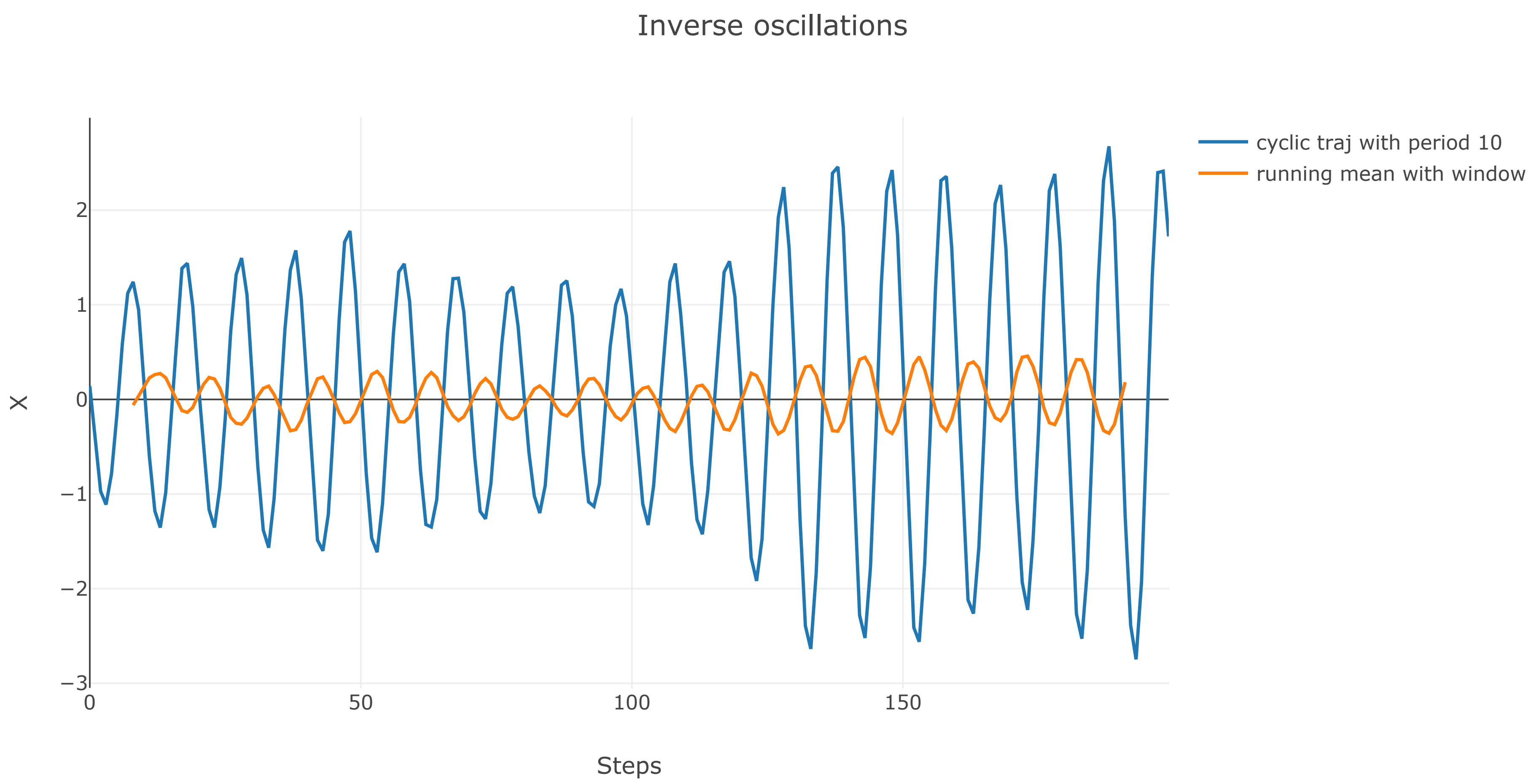
M = 17

gives us inverse oscillations: 5, 10, 11, 12, 13, 14, 15,

Decreases to zero: 17,

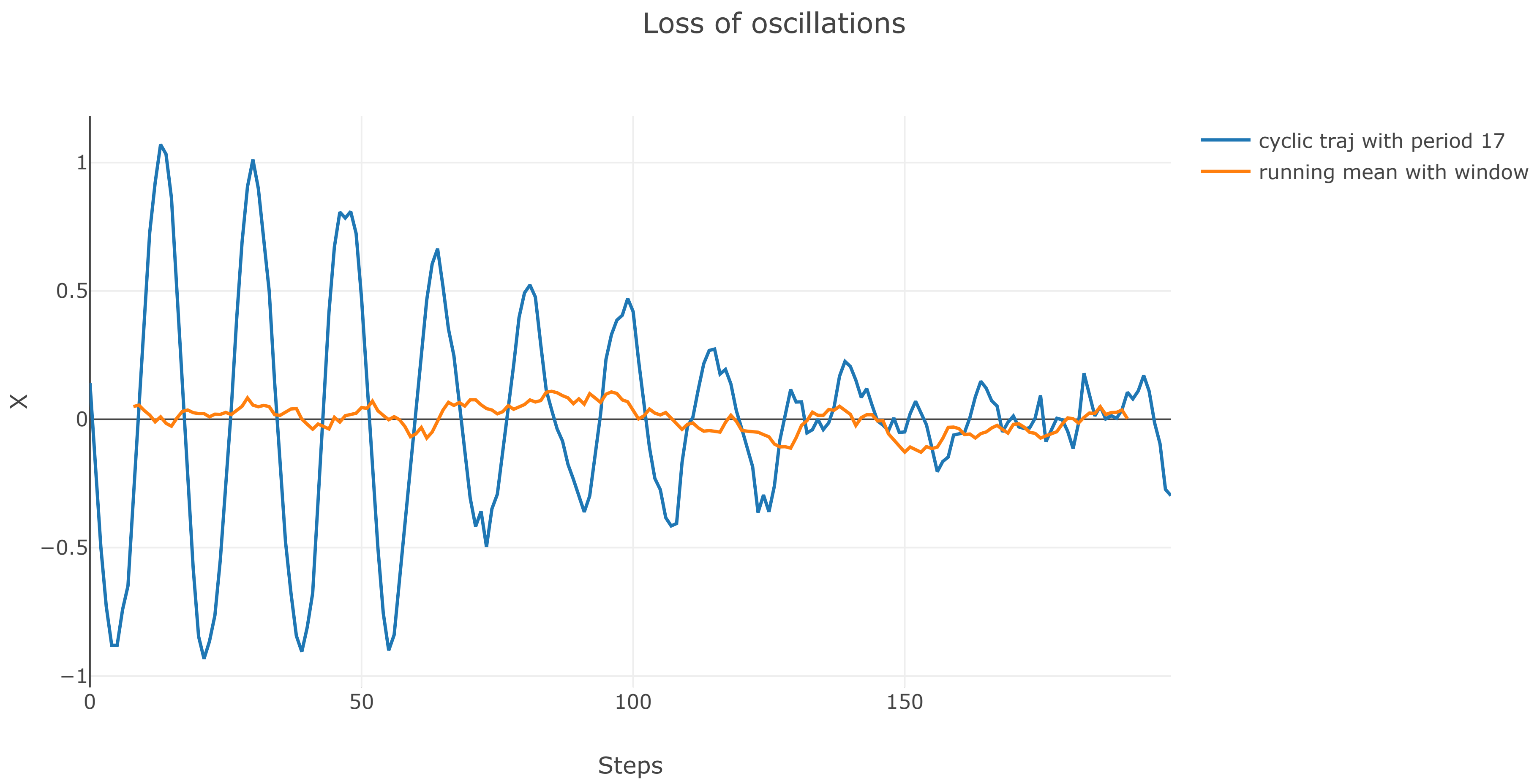
Not distorded in general: 20+

In [543]:
```
cyclic_traj_10 = generate_trajectory(A0 = 1, T = 10, var = 0.08**2, steps = 200)
CycTraj_measurments_10 = measure(cyclic_traj_10, var = 0.05)
cycl_running_mean_10 = pd.Series(CycTraj_measurments_10).rolling(window = 17, center = True).mean()


data = [
    go.Scatter(
        y=cyclic_traj_10,
        name='cyclic traj with period 10'
    ),
    go.Scatter(
        y=cycl_running_mean_10,
        name='running mean with window 17'
    ),
]

layout= go.Layout(
    title= 'Inverse oscillations',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



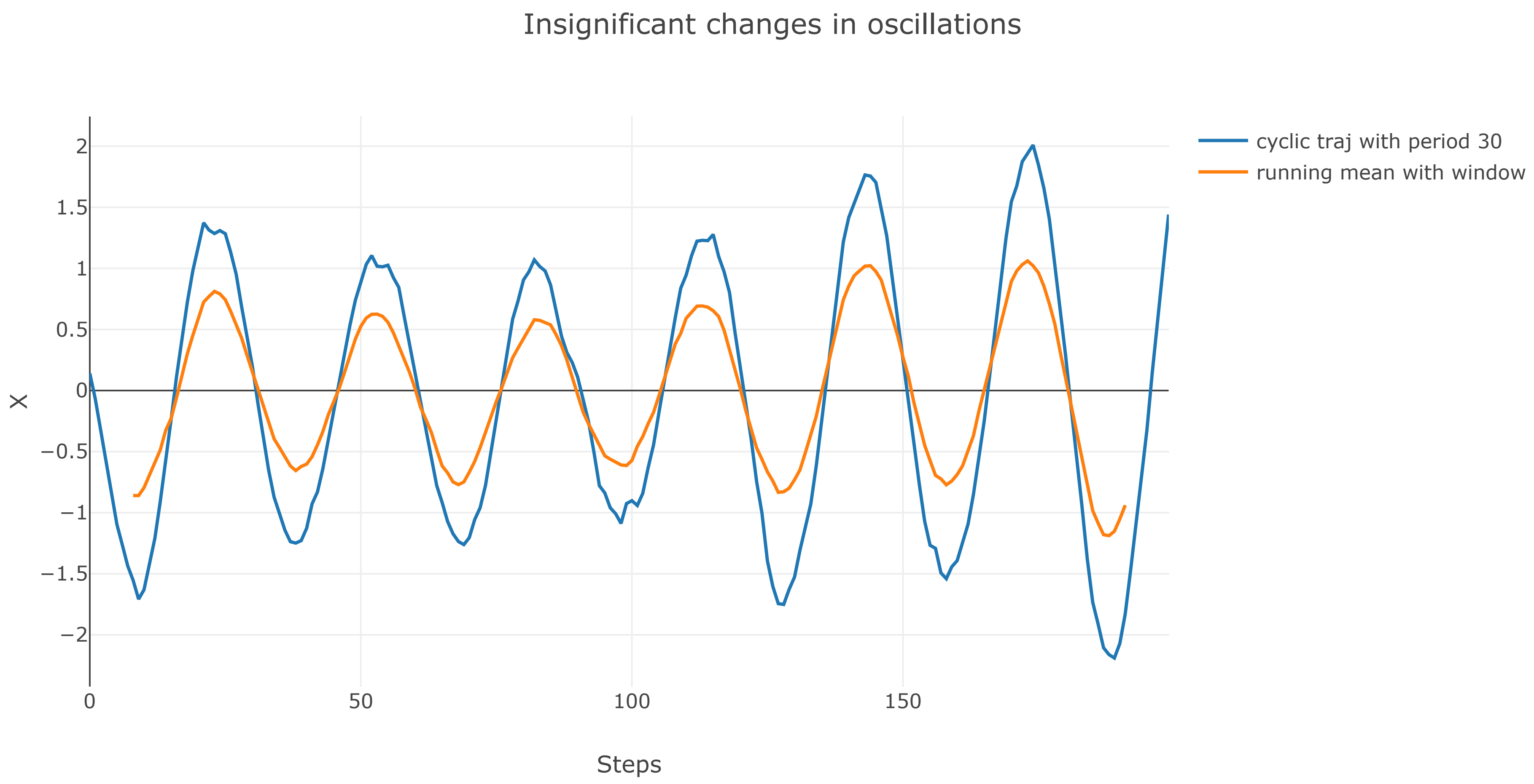
Export to plo

In [551]:
```
cyclic_traj_17 = generate_trajectory(A0 = 1, T = 17, var = 0.08**2, steps = 200)
CycTraj_measurments_17 = measure(cyclic_traj_17, var = 0.05)
cycl_running_mean_17 = pd.Series(CycTraj_measurments_17).rolling(window = 17, center = True).mean()


data = [
    go.Scatter(
        y=cyclic_traj_17,
        name='cyclic traj with period 17'
    ),
    go.Scatter(
        y=cycl_running_mean_17,
        name='running mean with window 17'
    ),
]

layout= go.Layout(
    title= 'Loss of oscillations',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

In [552]:
```
cyclic_traj_30 = generate_trajectory(A0 = 1, T = 30, var = 0.08**2, steps = 200)
CycTraj_measurments_30 = measure(cyclic_traj_30, var = 0.05)
cycl_running_mean_30 = pd.Series(CycTraj_measurments_30).rolling(window = 17, center = True).mean()

data = [
    go.Scatter(
        y=cyclic_traj_30,
        name='cyclic traj with period 30'
    ),
    go.Scatter(
        y=cycl_running_mean_30,
        name='running mean with window 17'
    ),
]

layout= go.Layout(
    title= 'Insignificant changes in oscillations',
    xaxis= dict(
        title= 'Steps',
    ),
    yaxis=dict(
        title= 'X',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

Make conclusions about conditions of 7a,b,c.

If choose size of the window bigger that oscillation period, than running mean curve tends to produce inverse oscillations. When size of the window and period of oscillation nearly equal than we can observe the loss of oscillations. And in case the size of a window is lesser than oscillation period, than we observe insignificant changes in oscillations.