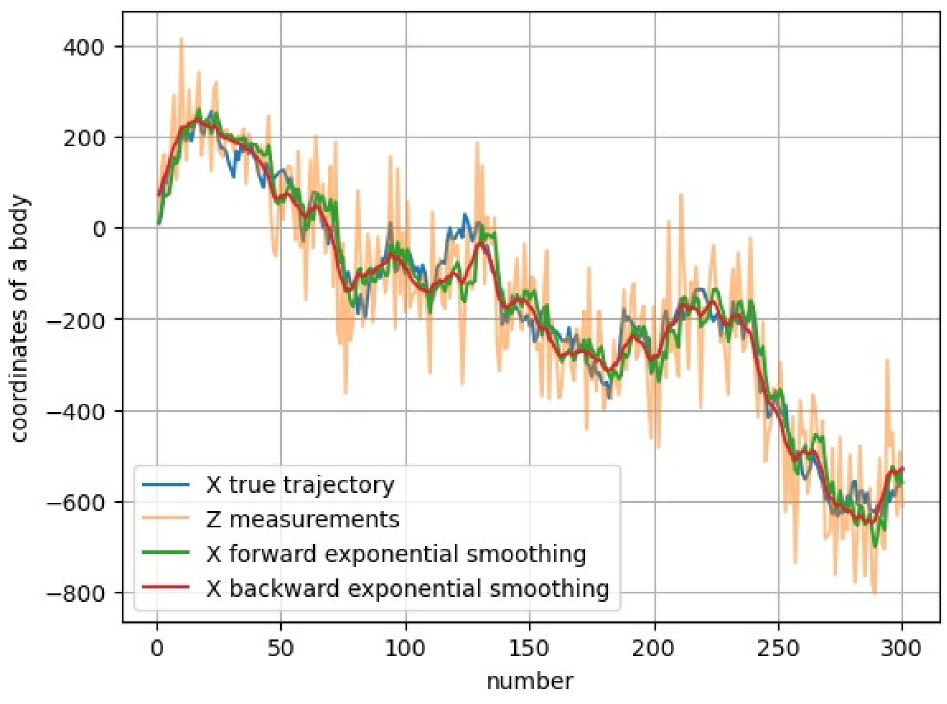
```
#Assignment 3
#Determining and removing drawbacks of exponential and running mean.
Task 1
#I. Backward exponential smoothing
#II. Drawbacks of running mean
#Team 12
#Yaroslav Savotin, Elizaveta Pestova, Selamawit Asfaw
#Skoltech, 2023

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#Part 1
#Step 1 - prepare dataset for backward exponential smoothing

#we use code from Assignment 2 with forward exponential smoothing to
apply a new method on it
#------------№1------------

#create our dataset
n2 = 300
X2 = [0 for _ in range(n2)]
X2[0] = 10

# normally distributed random noise with zero mathematical
expectation and variance
mu, sigma1 = 0, 28**2
w2 = np.random.normal(mu, np.sqrt(sigma1), [n2,1])

for i in range(n2-1):
    X2[i+1] = X2[i] + w2[i+1]

#------------№2------------
Z2 = [0 for _ in range(n2)]

#make measurmets of created data with certain noise
#normally distributed random noise with zero mathematical
expectation and variance
mu, sigma2 = 0, 97**2
nu2 = np.random.normal(mu, np.sqrt(sigma2), [n2,1])

for i in range(n2):
    Z2[i] = X2[i] + nu2[i]

#------------№3------------
ksi = sigma1/sigma2
alfa = (-ksi + np.sqrt(ksi**2 + 4*ksi))/2


#------------№4------------
M = int((2-alfa)/alfa)
sigmaRM = sigma2/M
```

```python
sigmaES = (sigma2*alfa)/(2-alfa)

#------------№5------------
#running mean

R = [0 for _ in range(n2)]
sum = 0
r = int((M-1)/2) #3

for j in range(3,n2-3):
    for i in range(r+1):
        if i == 0:
            sum += Z2[j-i]
        else:
            sum += Z2[j-i] + Z2[j+i]
    R[j] = sum/M
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z2[j+i]
    R[j] = sum/r
    sum = 0

for j in range(n2-r,n2):
    for i in range(r):
        sum += Z2[j-i]
    R[j] = sum/r
    sum = 0

#Exponential mean
Xsm = [0 for _ in range(n2)]
Xsm[0] = X2[0]
for i in range(1,n2):
    Xsm[i] = Xsm[i-1] + alfa*(Z2[i] - Xsm[i-1])

#Step 2 - apply backward exponential smoothing

Xsm_back = [0 for _ in range(n2)]
Xsm_back[n2-1] = X2[n2-1]
for i in range(n2-2,-1,-1):
    Xsm_back[i] = Xsm_back[i+1] + alfa*(Xsm[i] - Xsm_back[i+1])

#Step 3 - for the visual comparison of real measurments and
postpocessed data we create a plot
#preparation for plotting
k = [1 for _ in range(n2)]
for i in range(n2-1):
    k[i+1] = k[i] + 1
for i in range(1,n2):
    Xsm[i] = Xsm[i][0]
for i in range(1,n2):
    X2[i] = X2[i][0]
for i in range(0,n2):
    Xsm_back[i] = Xsm_back[i][0]

#plot
y0 = X2
y1 = Z2
y3 = Xsm_back
y2 = Xsm
x = k
plt.plot(x,y0,label = 'X true trajectory')
plt.plot(x,y1,label = 'Z measurements', alpha=0.5)
plt.plot(x,y2,label = 'X forward exponential smoothing')
plt.plot(x,y3,label = 'X backward exponential smoothing')
plt.xlabel('number')
plt.ylabel('coordinates of a body')
plt.suptitle('Comparison of true values of process, forward and 
backward exponential smoothing.')
plt.legend()
plt.grid(True)
```

Comparison of true values of process, forward and backward exponential smoothing.

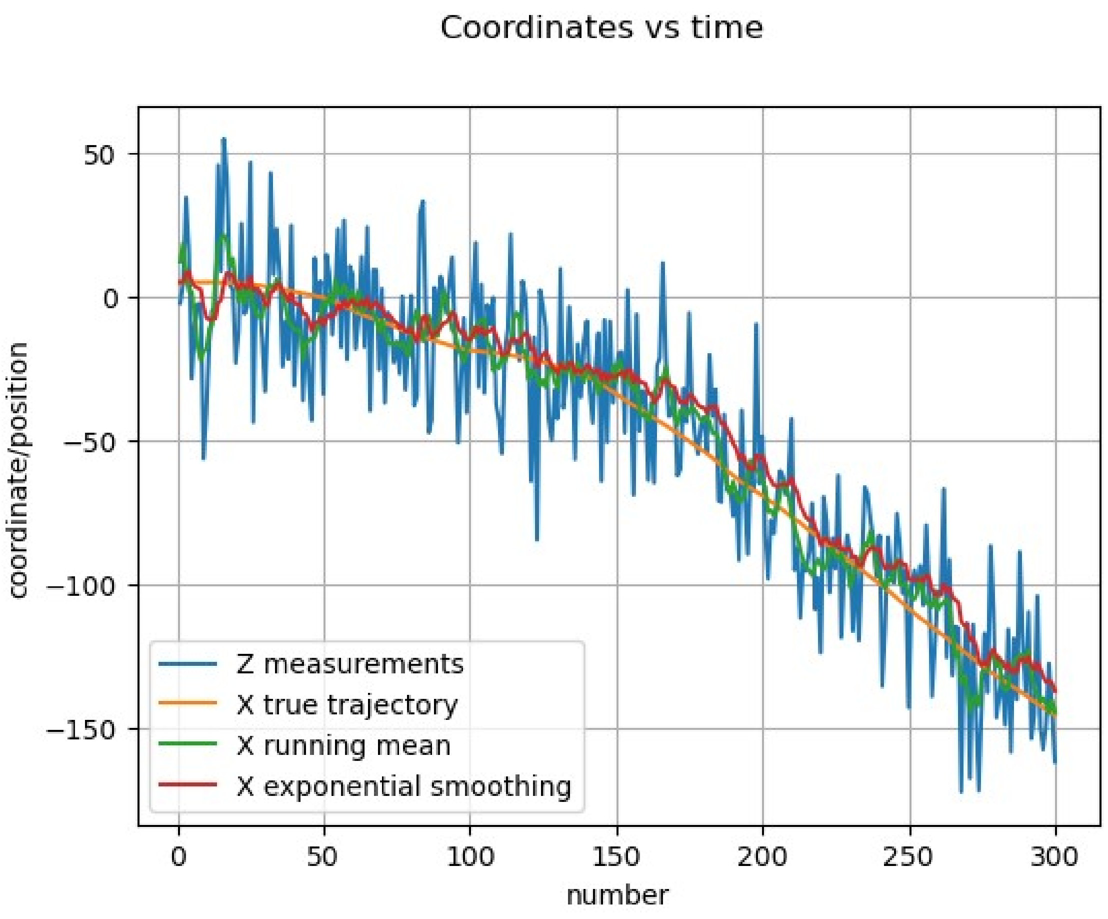


```
#Conclusion: plot shows us how applying  backward smoothing helps us 
to put away the shift of forward smoothing
```

From the hint on task we know that our trajectory is close to the line, but measurements noise is huge. And that give us understanding that to effectivly filtrate this noise, we must chose big window for Running mean method and small alfa for Exponential mean because we want our smoothing to bee less sensitive to noise.

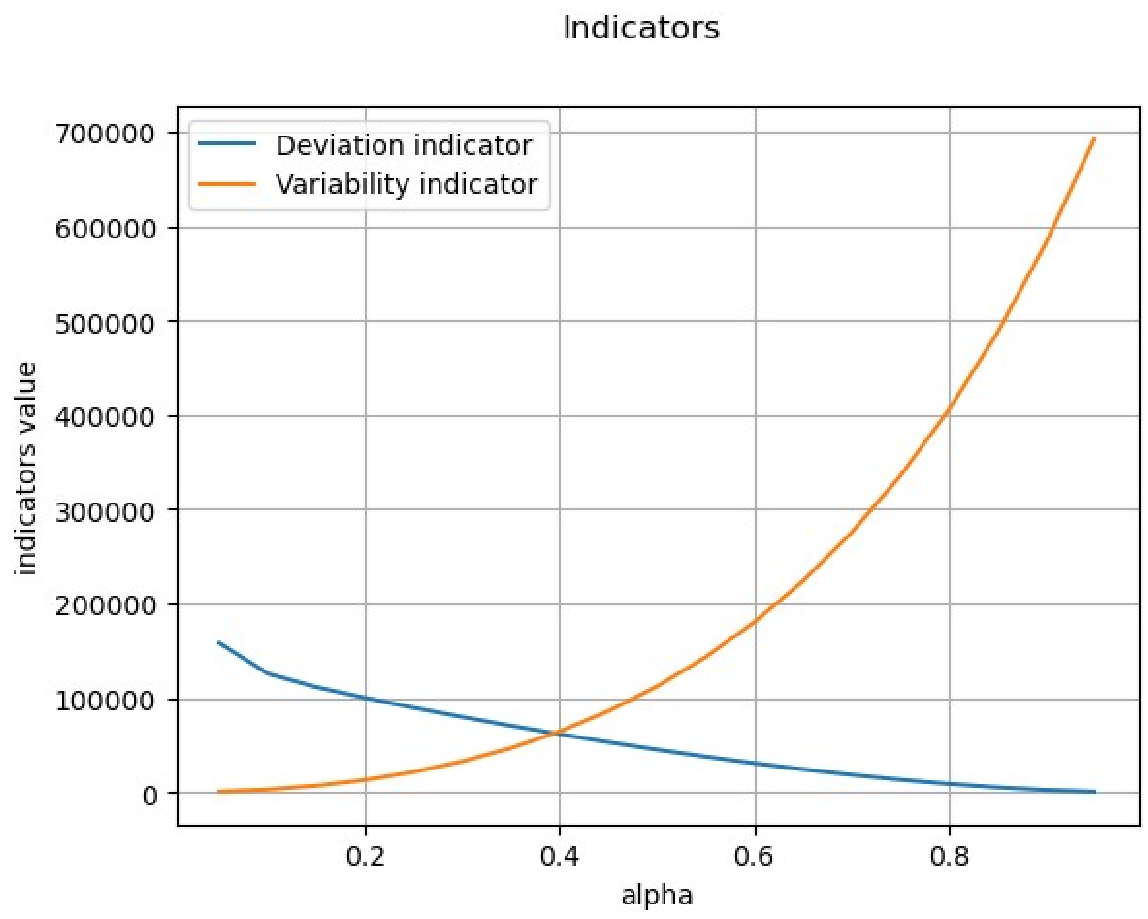
```
#Part 2 - investigations of drawbacks of running mean
#First trajectory
#Step 1 - for the First trajectory we create process which rate of
change is changed insignificantly but measurement noise is huge
n = 300

mu, sigma1 = 0, 10
a = np.random.normal(mu, np.sqrt(sigma1), [n,1])

X = [0 for _ in range(n)]
V = [0 for _ in range(n)]
X[0] = 5
V[0] = 0
T = 0.1
#with setted formulas we create our model of moution
for i in range(1,n):
    V[i] = V[i-1] + T*a[i-1]

for i in range(1,n):
    X[i] = X[i-1] + T*V[i-1] + (a[i-1]*T**2)/2

#then we create measurements of our motion process with certain
variance
Z = [0 for _ in range(n)]

# normally distributed random noise with zero mathematical
expectation and variance
mu, sigma2 = 0, 500
nu = np.random.normal(mu, np.sqrt(sigma2), [n,1])


for i in range(n):
    Z[i] = X[i] + nu[i]

#Step 2 - determine empirically the window size M
#because we want our filter be strong against high level of noise
and we know that our trajectory is close to line,
#for running mean we chose big window size, for example M=30, but it
can be even bigger

R = [0 for _ in range(n)]
sum = 0
r = int((M-1)/2)

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R[j] = sum/M
    sum = 0

for j in range(r):
```

```python
    for i in range(r):
        sum += Z[j+i]
    R[j] = sum/r
    sum = 0

for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R[j] = sum/r
    sum = 0

#for Exponential smoothing we want our alfa be small enough
alfa = 0.1
def Exp_smoothing(Z_f,n_f, alpha):
    Xsm_f = [0 for _ in range(n_f)]
    Xsm_f[0] = X[0]
    for i in range(1,n):
        Xsm_f[i] = Xsm_f[i-1] + alpha*(Z_f[i] - Xsm_f[i-1])
    return Xsm_f

Xsm = Exp_smoothing(Z,n,alfa)

#preparation for plotting
k = [1 for _ in range(n)]
for i in range(n-1):
    k[i+1] = k[i] + 1
for i in range(1,n):
    X[i] = X[i][0]

for i in range(1,n):
    Xsm[i] = Xsm[i][0]
for i in range(0,n):
    R[i] = R[i][0]

#plot
#alpha = 0.1
y1 = Z
y2 = X
y3 = R
y4 = Xsm
x = k
plt.plot(x,y1,label = 'Z measurements')
plt.plot(x,y2,label = 'X true trajectory')
plt.plot(x,y3,label = 'X running mean')
plt.plot(x,y4,label = 'X exponential smoothing')
plt.xlabel('number')
plt.ylabel('coordinate/position')
plt.suptitle('Coordinates vs time')
plt.legend()
plt.grid(True)
```



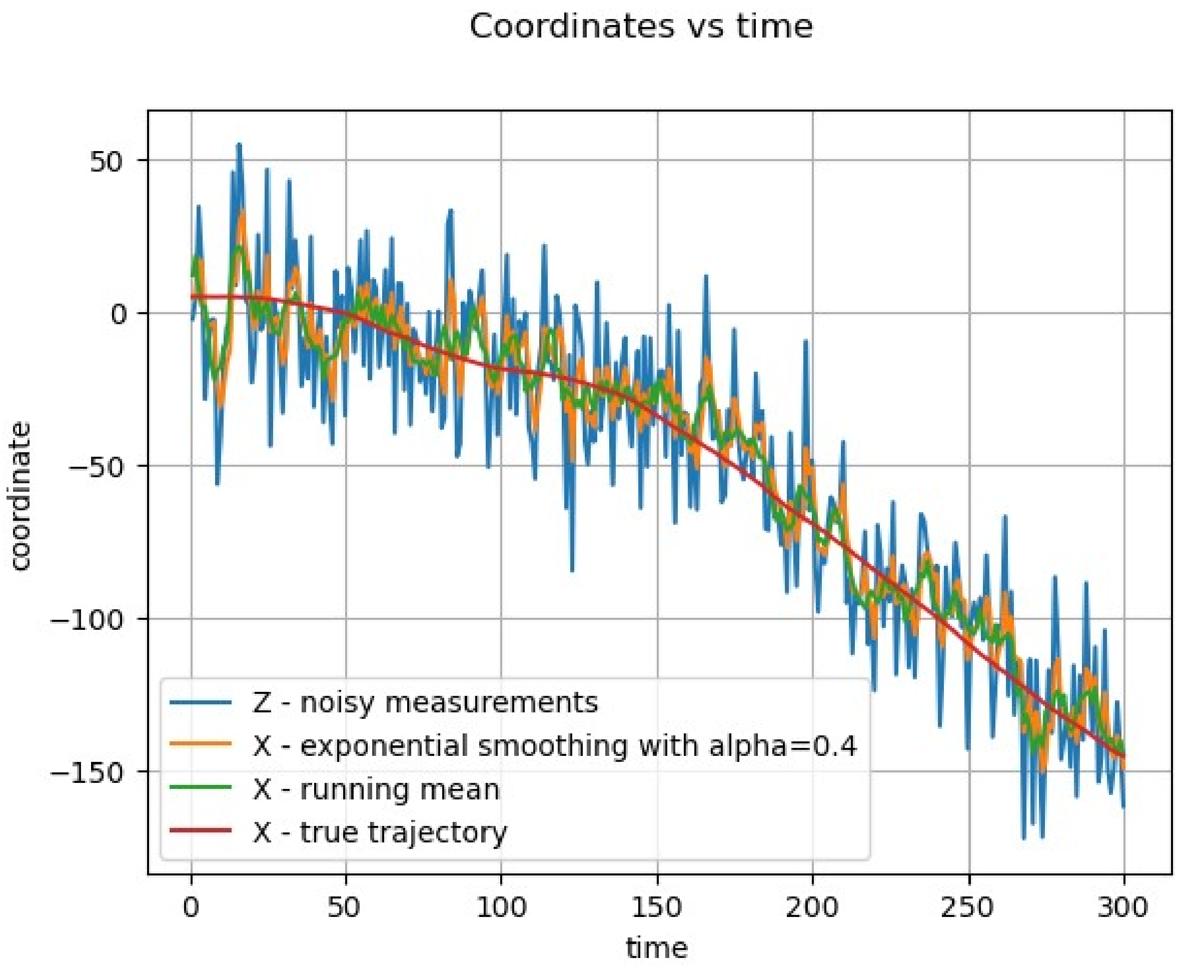
```
#Conclusion: because we have chosen small alfa, we sucsessfully
approached smoothed line in exponential method but to be more
presize we have to use indicators

#Step 3 - chosing the best method of smoothing based on deviation
and variability indicators

# to compare two methods of smoothing, we need to set the best alfa
for exponential smoothing

#smoothing at different alpha values
alfa_plot = [0 for _ in range(19)]
Xsm_array = [0 for _ in range(19)]
k = 0
Xsm_array = [0 for _ in range(19)]
for i in range(5,100,5):
    alfa = i/100
    Xsm_array[k] = Exp_smoothing(Z,n,alfa)
    alfa_plot[k] = i/100
    k +=1
#Xsm_array
for j in range(19):
    for i in range(1,n):
        Xsm_array[j][i] = Xsm_array[j][i][0]
for i in range(n):
    Z[i] = Z[i][0]
```

```python
#calculating deviation and variability indicators for Running Mean

# deviation indicator Running Mean
Z = np.array(Z)
R = np.array(R)
Id_rm = np.sum(np.square(Z-R))

#variation indicators for Running Mean
Iv_rm = 0

for j in range(n-2):
    Iv_rm += (R[j+2] - 2*R[j+1] + R[j])**2


#calculating deviation and variability indicators for Exponential
Mean with different alpha

#deviation indicator for Exponential Mean
Id_em = [0 for _ in range(19)]
for i in range(19):
    for j in range(n):
        Id_em[i] += np.square(Z[j] - Xsm_array[i][j])

#variability indicator for Exponential Mean
Iv_em = [0 for _ in range(19)]
for i in range(19):
    for j in range(n-2):
        Iv_em[i] += (Xsm_array[i][j+2] - 2*Xsm_array[i][j+1] +
Xsm_array[i][j])**2

#for different alfa we are getting different deviation and
variability indicators, so for easier choice we plot them both in
dependance of alfa
y1 = Id_em
y2 = Iv_em
y4 = Xsm
x = alfa_plot
plt.plot(x,y1,label = 'Deviation indicator')
plt.plot(x,y2,label = 'Variability indicator')
plt.xlabel('alpha')
plt.ylabel('indicators value')
plt.suptitle('Indicators')
plt.legend()
plt.grid(True)
```

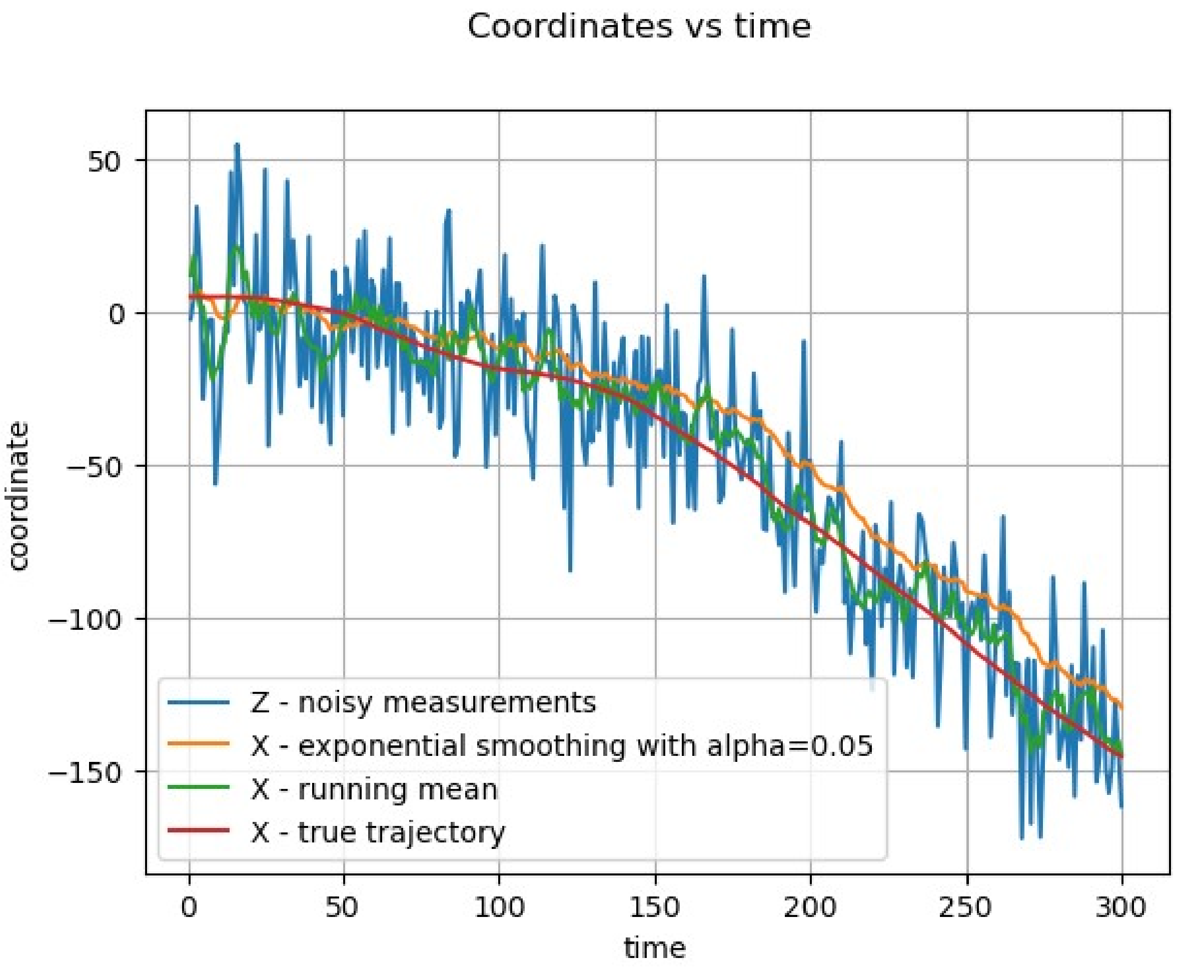


```
#Conclusion: we know that Deviation indicator(it represents the
difference between our noisy measurements and eestimation values)
#should be big - it represents good smoothing of noisy peaks, at the
same time our Variability Indicator
#(which represenrs the difference betweem close values of dataset)
should be small, to indicate a good quality of smoothing.
#From the plot above it may look like the optimal alpha is on the
intersection of indicators lines and equal to 0.4, so lets see what
smoothing this alfa will give us
```

#smoothing with alpa=0.4 gives us some level of smoothing but the result is still too noisy



#thats how we understand that statement about optimal alpha on the intersection of indicators lines is not true. Also from remaining high level of noise, we should consider smaller values of alfa. And based on nature of indicators we want alpha to be very close to zero, so lets try alpa=0.05



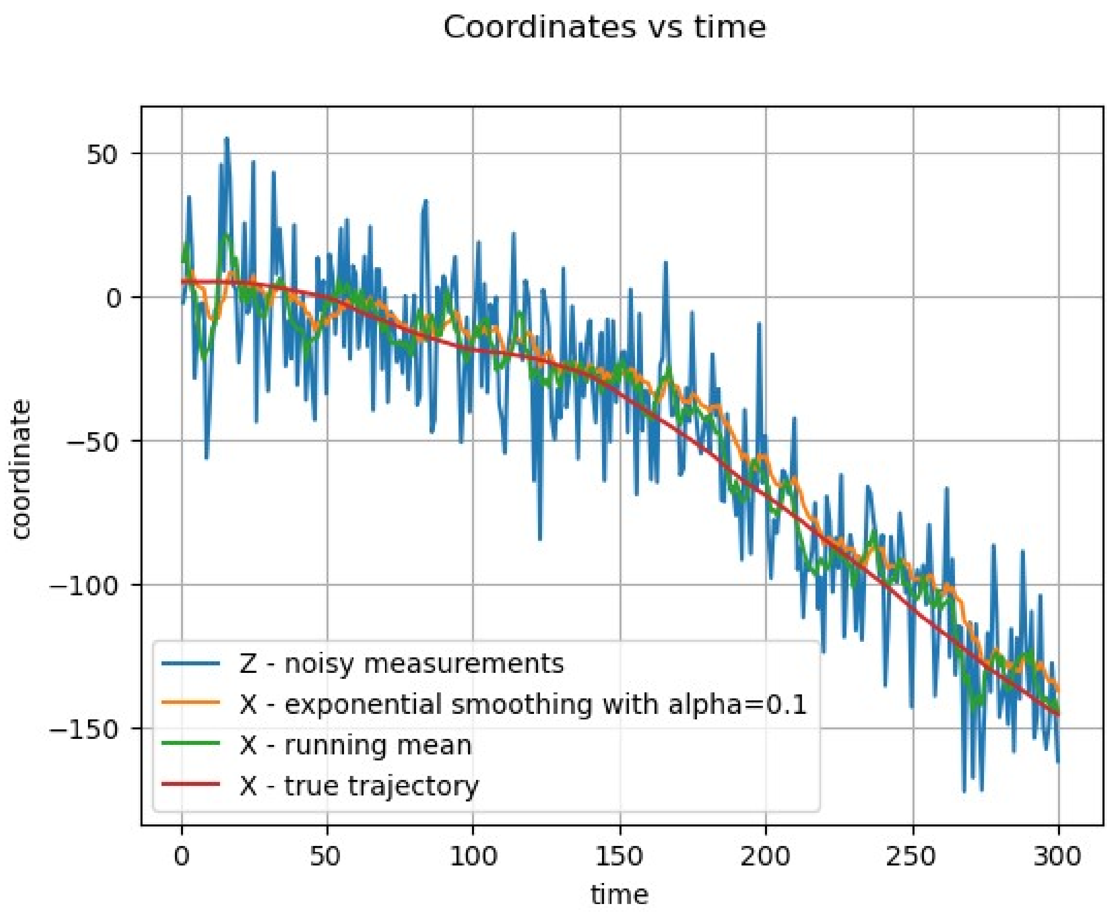
#small alpha always gives high level of smoothing, but the shift from measurements is also huge

#After consulting with Prof. Tatiana from her rich experience we get that the smallest alpha which is usuallytaken is alpa=0.1

#All in all by the empirical way the best alfa = 0.1 was chosen. Values which were even smaller, were leading to the significant shift from true trajectory. And bigger ones couldnt provide enough filtration

```
#the balance point value of both indicators was selected alpha =
0.1, so lets see how it perform our smoothing task

Xsm = Exp_smoothing(Z,n,0.1)
#for i in range(1,n):
#    Xsm[i] = Xsm[i][0]
k = [1 for _ in range(n)]
for i in range(n-1):
    k[i+1] = k[i] + 1

#calculating deviation and variability indicators for for
Exponential Mean

# deviation indicator for Exponential Mean
```

```
Id_em = np.sum(np.square(Z-Xsm))

#variation indicators for Exponential Mean
Iv_em = 0
for j in range(n-2):
    Iv_em += (Xsm[j+2] - 2*Xsm[j+1] + Xsm[j])**2

#plot everything again, but now wirh exponential smothing
(alpha=0.1)
#alpha = 0.1
y1 = Z
y2 = Xsm
y3 = R
y4 = X
x = k
plt.plot(x,y1,label = 'Z - noisy measurements')
plt.plot(x,y2,label = 'X - exponential smoothing with alpha=0.1')
plt.plot(x,y3,label = 'X - running mean')
plt.plot(x,y4,label = 'X - true trajectory')
plt.xlabel('time')
plt.ylabel('coordinate')
plt.suptitle('Coordinates vs time')
plt.legend()
plt.grid(True)
```



```
#Conclusion: this plot clearly shows us that both methods of
smoothing visually perform pretty simular, so  our final chioce of
method will be based on indicators values

print('Deviation ind for running mean =',Id_rm)
print('Deviation ind for exponential mean =',Id_em)
print('Variabiliry ind for running mean =',Iv_rm)
print('Variabiliry ind for exponential mean =',Iv_em)
```

```
Deviation ind for running mean = 107799.71329175869
Deviation ind for exponential mean = 115252.996723701
Variabiliry ind for running mean = 11101.191802474705
Variabiliry ind for exponential mean = 2804.379166018918
```

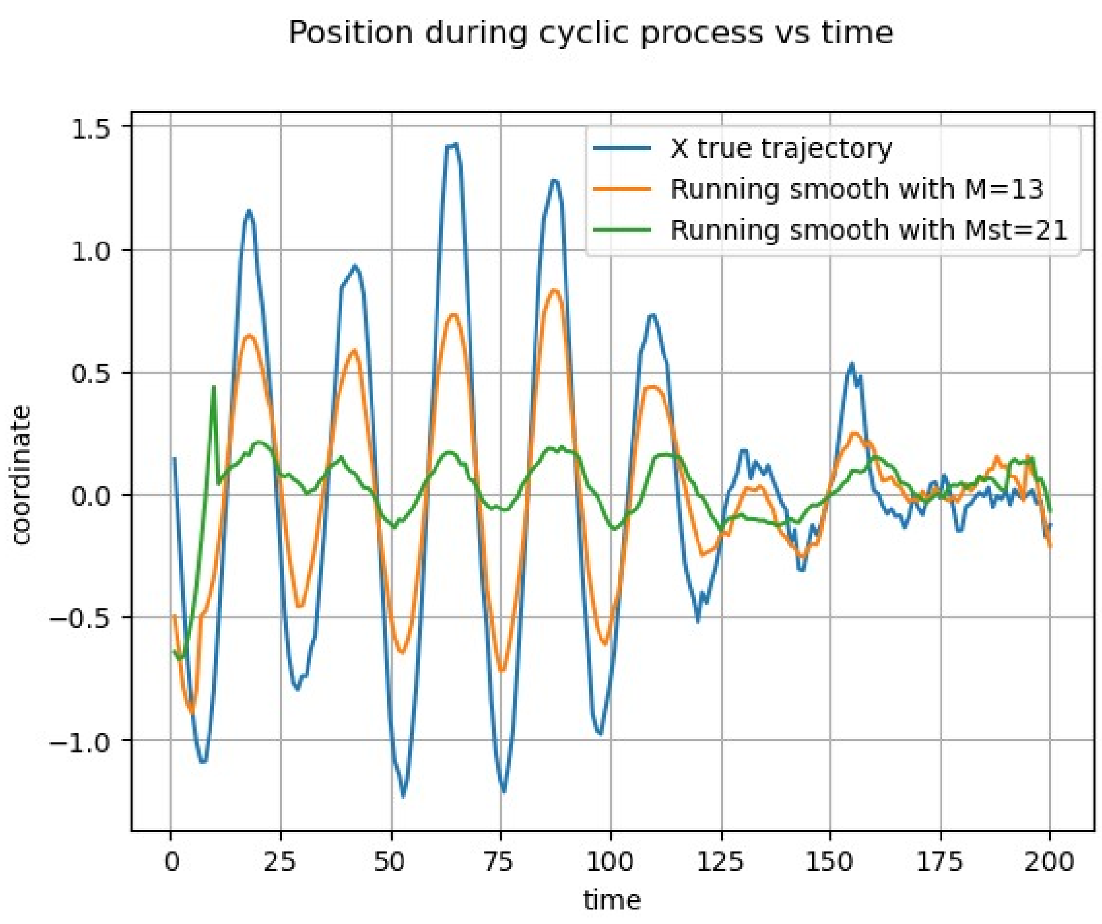
Conclusion: verifying best method for smoothing in case of absence of true trajectory data,we consider two indicators: Deviation Indicator and Variability Indicator.

For Deviation Indicator we know, that the closer it is to zero, the more sensitive to noise is our filter, so in our case: Dev ind for Running mean method=112167.9 and for Exponential Mean method=116291.6, so they are pretty similar. So lets see on the next indicator.

Considering the fact that the closer Variability Indicator is to zero, the more effective is our filter in smoothing, and getting: Var ind for Running mean = 12750.2 and Var ind for exponential mean = 2905.8, we can conclude that in our case Exponential smoothing performs better. So we chose Exponential smoothing method as the most suitable for this case.

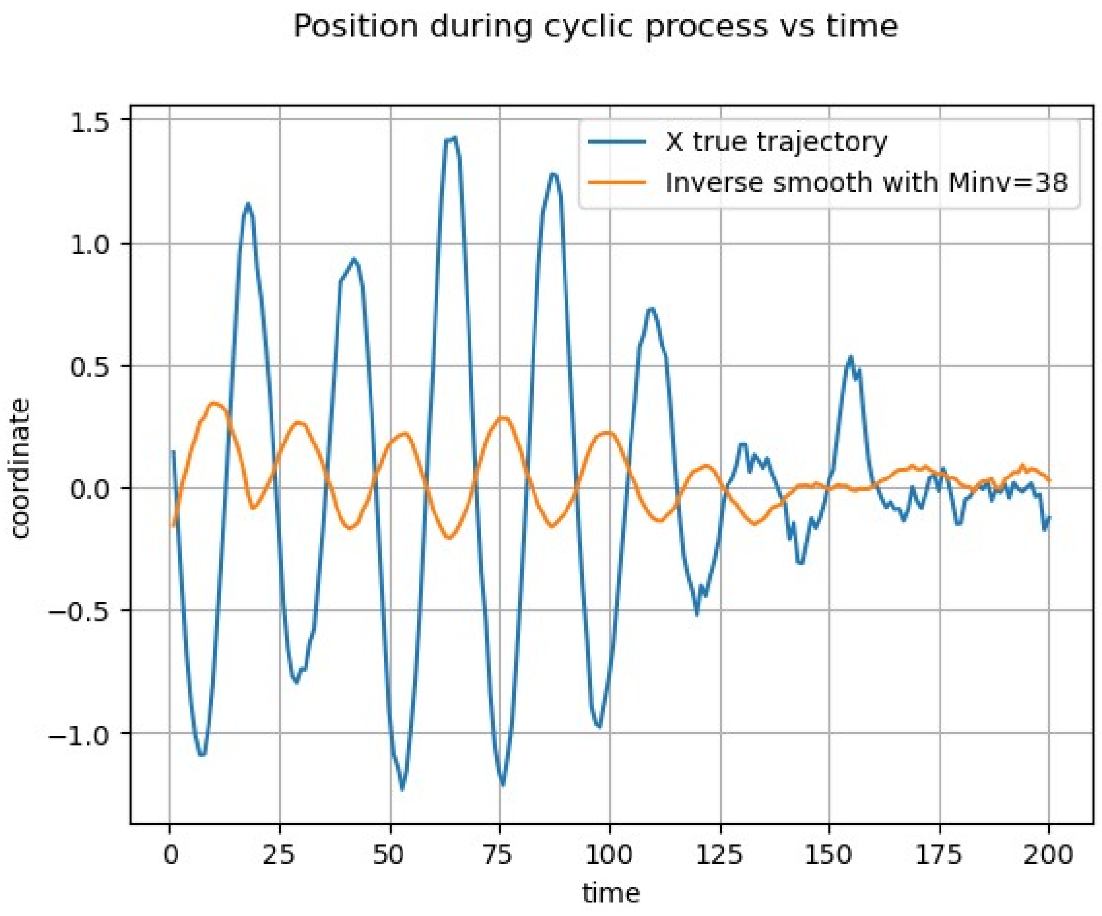
```
#Second trajectory

#Step 4 - generate cyclic trajectory
n = 200

mu, sigma1 = 0, 0.08**2
w = np.random.normal(mu, np.sqrt(sigma1), [n,1])

X = [0 for _ in range(n)]
A = [0 for _ in range(n)]
A[0] = 1
T = 32
omega = 2**np.pi/T

for i in range(1,n):
    A[i] = A[i-1] + w[i]

for i in range(0,n):
    X[i] = A[i]*np.sin(omega*i + 3)

#Step 5 - making measurments of generated trajectory with small
noise
Z = [0 for _ in range(n)]

# normally distributed random noise with zero mathematical
expectation and variance
mu, sigma2 = 0, 0.05
```

```python
nu = np.random.normal(mu, np.sqrt(sigma2), [n,1])


for i in range(n):
    Z[i] = X[i] + nu[i]

#Step 6 - applying smoothing with window M = 13
M = 13
R1 = [0 for _ in range(n)]
sum = 0
r = int((M-1)/2) #3

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R1[j] = sum/M
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z[j+i]
    R1[j] = sum/r
    sum = 0

for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R1[j] = sum/r
    sum = 0
#getting dataset R1 for window M=13

#Step 6  - applying running mean method with window size for our
Group4 M=21,

Mst = 21
R2 = [0 for _ in range(n)]
sum = 0
r = int((Mst-1)/2)

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R2[j] = sum/Mst
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z[j+i]
    R2[j] = sum/r
    sum = 0

for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R2[j] = sum/r
    sum = 0
#getting dataset R2 for window Mst=21

#preparation for plotting
k = [1 for _ in range(n)]
for i in range(n-1):
    k[i+1] = k[i] + 1
for i in range(1,n):
    X[i] = X[i][0]

 for i in range(0,n):
    R1[i] = R1[i][0]
    R2[i] = R2[i][0]

#plot
y0 = X
y1 = R1
y2 = R2
x = k
plt.plot(x,y0,label = 'X true trajectory')
plt.plot(x,y1,label = 'Running smooth with M=13')
plt.plot(x,y2,label = 'Running smooth with Mst=21')
plt.xlabel('time')
plt.ylabel('coordinate')
plt.suptitle('Position during cyclic process vs time ')
plt.legend()
plt.grid(True)
```



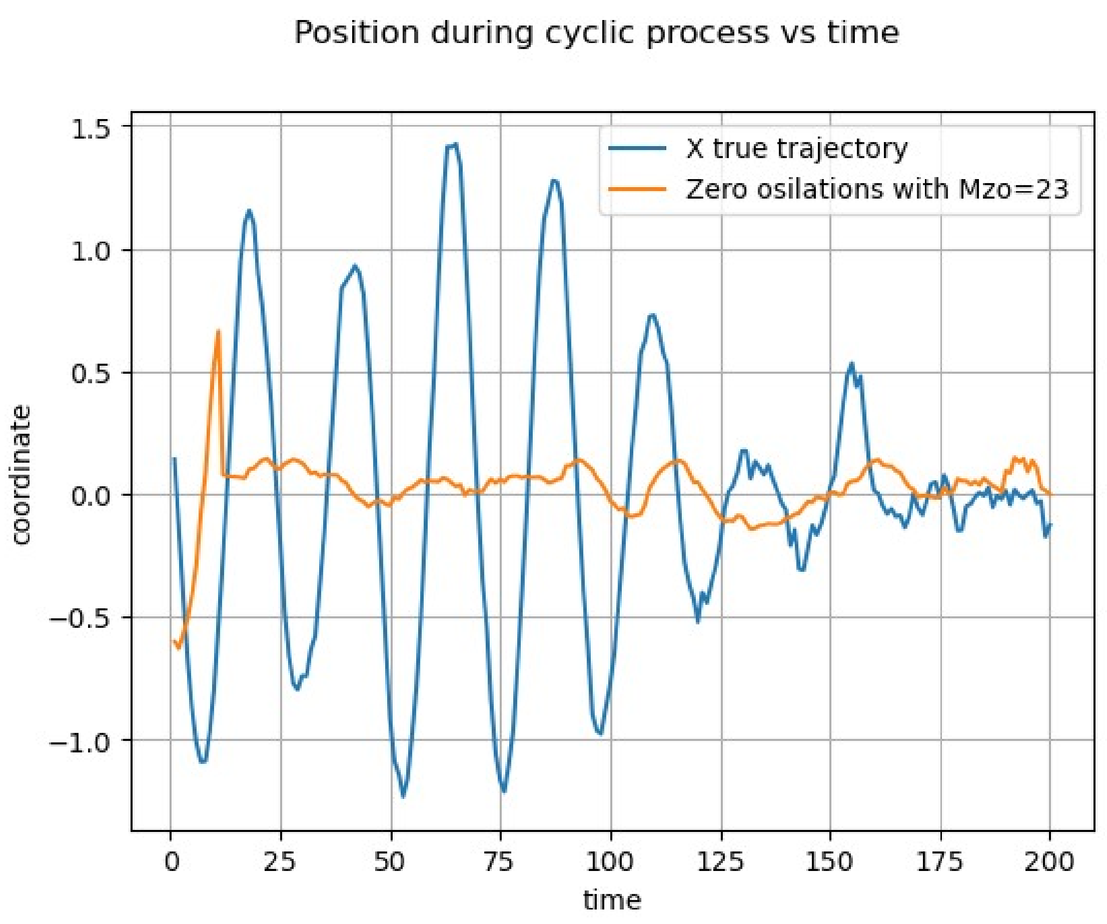
```
#Conclusion: here we see unsignificant change of phase and
significant change in amplitude for smoothing window M~0.5T=13
#for smoothing window Mst~T=21 change of phase is undetectable, but
change in amplitude becomes even bigger

#Step 7 a)
#by adjusting M we are getting inverse oscillations
Minv = 38
R3 = [0 for _ in range(n)]
sum = 0
r = int((Minv-1)/2) #3

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R3[j] = sum/Minv
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z[j+i]
    R3[j] = sum/r
    sum = 0
```

```
for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R3[j] = sum/r
    sum = 0
#getting dataset R3 for window M=26

#plot
y0 = X
y3 = R3
x = k
plt.plot(x,y0,label = 'X true trajectory')
plt.plot(x,y3,label = 'Inverse smooth with Minv=38')
plt.xlabel('time')
plt.ylabel('coordinate')
plt.suptitle('Position during cyclic process vs time')
plt.legend()
plt.grid(True)
```



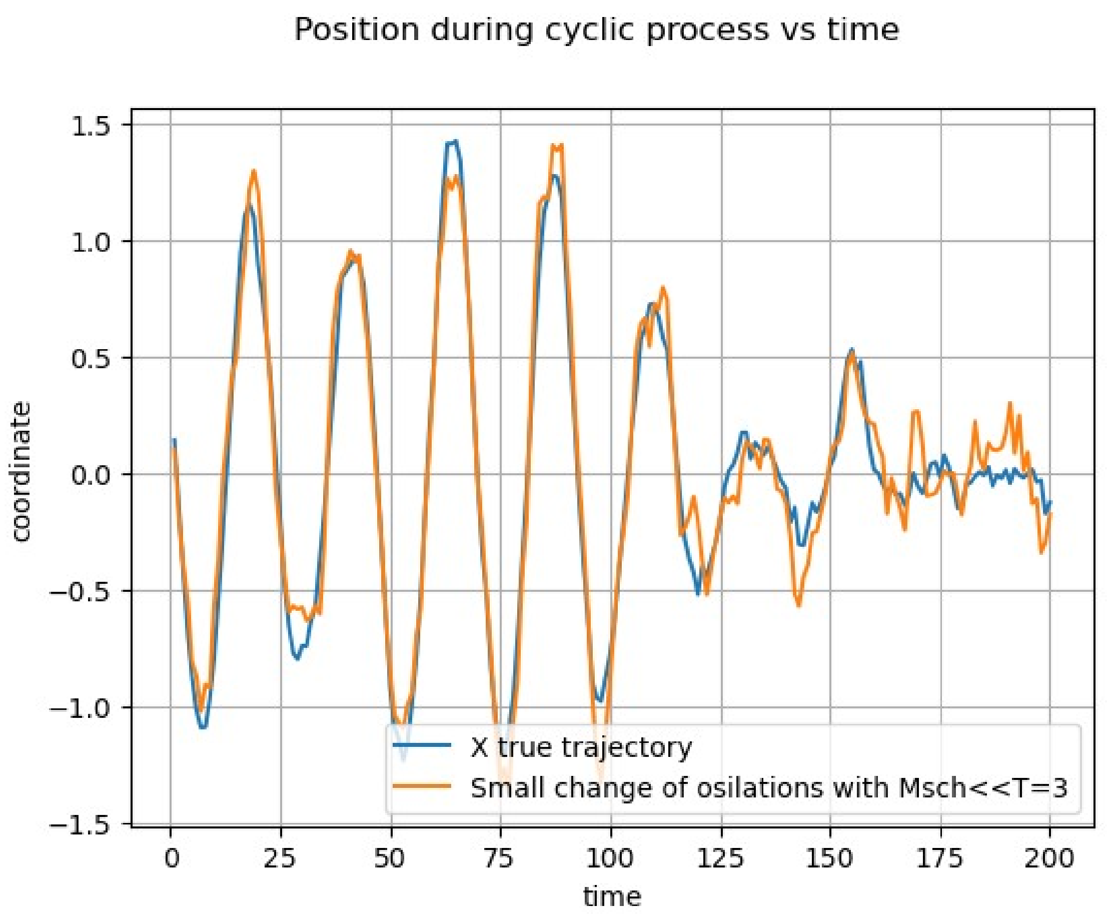
```
#Conclusion: inverse we see in conditions of Minv~1.5T=38

#Step 7 b)
#for this point we adjust M in the way so it produce loss of
oscillations
```

```python
Mzo = 23
R4 = [0 for _ in range(n)]
sum = 0
r = int((Mzo-1)/2) #3

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R4[j] = sum/Mzo
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z[j+i]
    R4[j] = sum/r
    sum = 0

for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R4[j] = sum/r
    sum = 0
#getting dataset R4 for window M=23

#plot
y0 = X
y4 = R4
x = k
plt.plot(x,y0,label = 'X true trajectory')
plt.plot(x,y4,label = 'Zero osilations with Mzo=23')
plt.xlabel('time')
plt.ylabel('coordinate')
plt.suptitle('Position during cyclic process vs time')
plt.legend()
plt.grid(True)
```



```
#Conclusion: almost full loss of osilations give as Mzo~T=23

#Step 7 b)
#for this point we are searching for conditions which do not change
theplot signeficaly

Msch = 3
R5 = [0 for _ in range(n)]
sum = 0
r = int((Msch-1)/2) #3

for j in range(r,n-r):
    for i in range(r+1):
        if i == 0:
            sum += Z[j-i]
        else:
            sum += Z[j-i] + Z[j+i]
    R5[j] = sum/Msch
    sum = 0

for j in range(r):
    for i in range(r):
        sum += Z[j+i]
    R5[j] = sum/r
    sum = 0
```

```python
for j in range(n-r,n):
    for i in range(r):
        sum += Z[j-i]
    R5[j] = sum/r
    sum = 0
#getting dataset R4 for window M=23

#plot
y0 = X
y5 = R5
x = k
plt.plot(x,y0,label = 'X true trajectory')
plt.plot(x,y5,label = 'Small change of osilations with Msch<<T=3')
plt.xlabel('time')
plt.ylabel('coordinate')
plt.suptitle('Position during cyclic process vs time')
plt.legend()
plt.grid(True)
```



```python
#Step 7 c)
#Conclusion: to change the oscillations insignificantly we need our
window Msch be very small, for example 3, which is 1/8 of T
```

#Step 8 - Make conclusions about conditions of 7a,b,c Conclusion: by adjusting M to conditions of every point of task a,b,c we get following results:

a) to produce inverse oscillations, size of M should be 1.5T, whete T is a period of oscillations

b) to loss of oscillations size of M should be approximately equal to T

c)to changes the oscillations insignificantly we should set M<<T

when M is approaching T the amplitude decreses and phase does not change a lot, period itself does not change at all

#Step 9 - conclusions to the Assignment:

personal Learning Log:

```
Yaroslav: from this task I learned how to structure and ajust code
written by other people, debug it. Also details and experience of
choosing the better method of smoothing with help of indicators was
new useful instrument in my data analisys toolkit.

Lisa: I learned how to create nested loops for ajusting variables to
automaticly check diffeerent variants of function. Creating plots
already become everyday routine.

Selamawit Asfaw: After we Conduct series of experiments by taking
different window size and smoothing constant(alpha) we obtain the
optimum alpha and window size. I have understood that the backward
exponential smoothing is a better smoothing method at optimum
smoothing constant compared to forward exponential smoothing and
running mean smoothing method.
```