Skoltech

Skolkovo Institute of Science and Technology

I.Backward exponential smoothing

II.Drawbacks of running mean

Experimental Data Processing

Assignment №3

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 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.

2 Methodology

2.1 Backward exponential smoothing

Question 2
Apply backward exponential smoothing to forward exponential estimates to further smooth measurements errors.

Question 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

2.2 Drawbacks of running mean

2.2.1 First trajectory

Analyze a process which rate of change is changed insignificantly and measurement noise is great. Study a cyclic process, and measurement noise is small.

Question 1
Generation of true trajectory X_i of an object motion distributed by normally distributed random acceleration

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

Size of trajectory is 300 points, initial condition: $X_i = 5$, $V_1 = 0$, $T = 0.1$ and variance of noise $\sigma_\alpha^2 = 10$

Measurement z_i of the process X_i:

$$z_i = X_i + \eta_i \tag{2}$$

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

Question 2
Determination of 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.

Question 3
Choose better smoothing method using deviation and variability indicators.

2.2.2 Second trajectory

Question 4
Generation of cyclic trajectory X_i according to Equation 3.

$$\begin{aligned} X_i &= A_i sin(\omega i + 3) \\ A_i &= A_{i-1} + w_i \end{aligned} \tag{3}$$

$T = 32$ - is the period of oscillations, w_i - normally distributed random noise with zero mathematical expectation and variance $\sigma_w^2 = 0.08^2$, initial condition is $A_1 = 1$ and size of trajectory is 300 steps.

Question 5
Generation of measurements z_i of the process X_i using 4.

$$z_i = X_i + \eta_i \tag{4}$$

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

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

Question 7
Determine the period of oscillations for which running mean with given for every group window size:

1. produces inverse oscillations
2. leads to the loss of oscillations (zero oscillations)
3. 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: $M = 27$

Question 8
Make conclusion about conditions of 1,2,3.

3 Results

3.1 Backward exponential smoothing

Question 3
It can be seen from the Figure 1 that backward exponential is much smoother than the running mean.

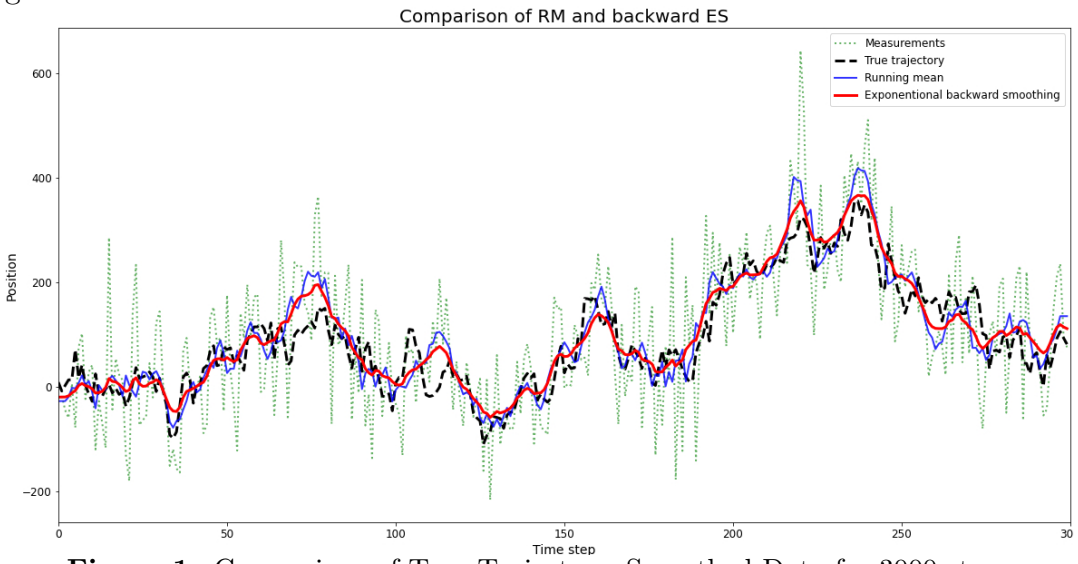


Figure 1: Comparison of True Trajectory, Smoothed Data for 3000 steps

Using Equations 5 and 6 we can compare deviation and variability indicators:

$$I_d = \sum_{i=1}^{N} (z_i - \hat{X}_i)^2 \tag{5}$$

$$I_\nu = \sum_{i=1}^{N} (\hat{X_{i+2}} - 2\hat{X_{i+1}} + \hat{X}_i)^2 \tag{6}$$

As a proof a visual representation, both deviation and variability indicator shows that backward exponential is more accurate and smoother than the running mean.

	Backward ES	**RM**
Deviation Indicator I_d	2.2454×10^6	2.2858×10^6
Variability Indicator I_ν	1.5553×10^4	2.0744×10^5

3.2 Drawback of running mean

3.2.1 First trajectory

Question 2

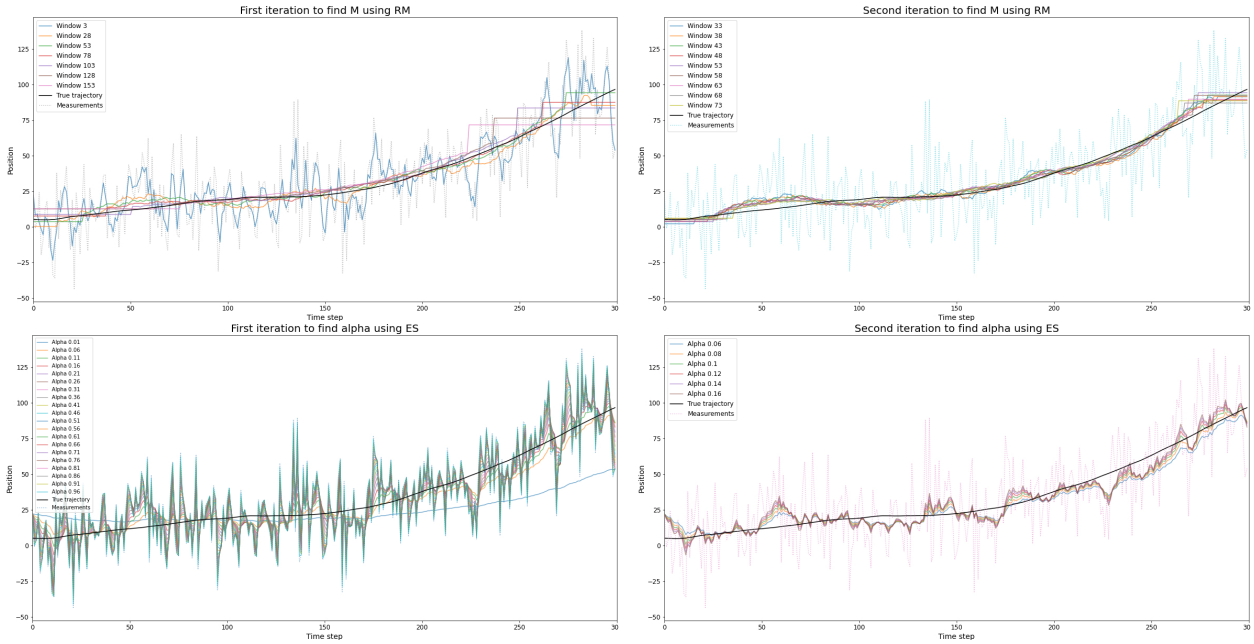


Figure 2: Iterations for finding optimal α and M using RM and ES

After many iterations presented in Figure 2, it is shown that smaller alpha values give much smoother results and also measurement errors are a lot whereas out trajectory is pretty close to the line so it is plausible to use larger M numbers. Therefore, we have come with:

$$\alpha = 0.1$$
$$M = 63$$

Question 3

After obtaining deviation and variability indicator, it is clear that running mean is much more smoother and accurate than the forward exponential smoothing.

	Forward ES	**RM**
Deviation Indicator I_d	1.5716×10^5	1.6024×10^5
Variability Indicator I_ν	1254.0	180.74

3.2.2 Second trajectory

Question 4 - 6

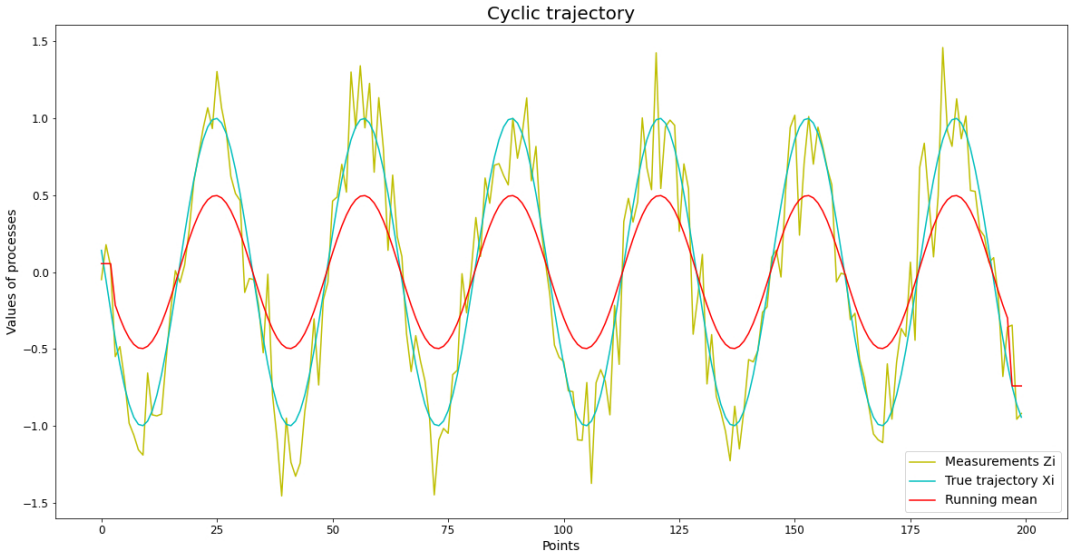


Figure 3: Cyclic trajectory

Question 7 - 8

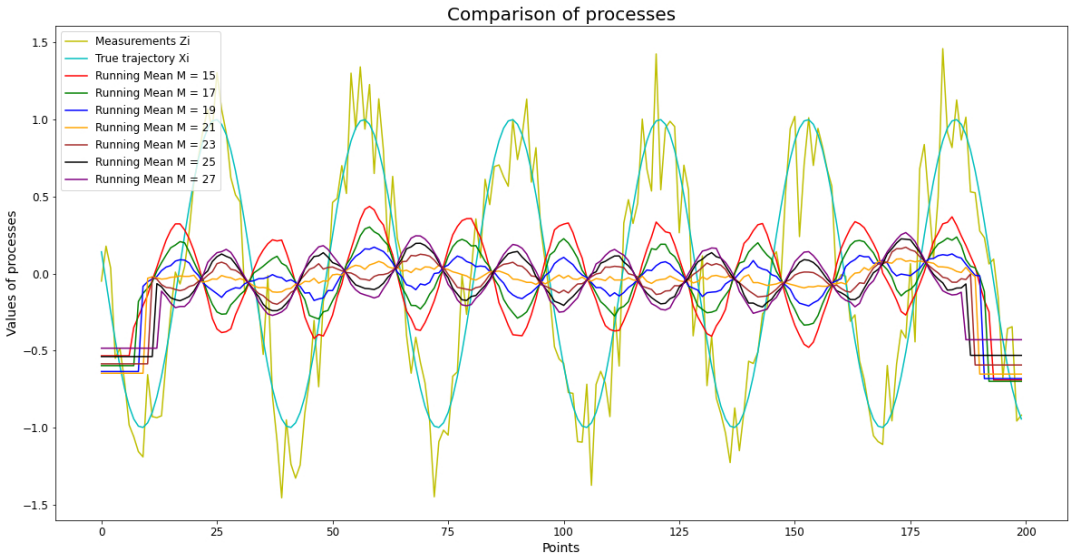


Figure 4: True trajectory, measurement, cyclic trajectory with $T = 21$ with various M

Looking at the 4 we can make the conclusion of Question 8. The period is taken as $T = 21$. If the window size is:

- smaller than the given period then it produces inverse oscillations.
- same as the given period then negligible oscillations have been observed.
- higher than the given period then smaller change in oscillations has been observed.

4 Conclusion

What we have learnt and tried:

1. Applied backward exponential smoothing
2. Backward exponential smoothing is more accurate and smoother than the running mean
3. Way to choose the optimal M and α
4. Generation of cyclic trajectory

What we have reflected upon:

1. It is possible to use the indicators to choose the better method if it's hard to determine it visually
2. Backward exponential smoothing takes into consideration both present and future measurements, resulting in more accurate predictions
3. Via deviation and variable indicators it's possible to check the accuracy of estimations
4. Oscillation of cyclic trajectory depends on the period when window size is constant

Contribution of each members:

1. Ilya: wrote the code and plotted for the first part.
2. Ruslan: wrote the code and plotted for first trajectory of the second part
3. Yunseok: wrote the code and plotted for second trajectory of the second part, wrote the report

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

ASSIGMENT III

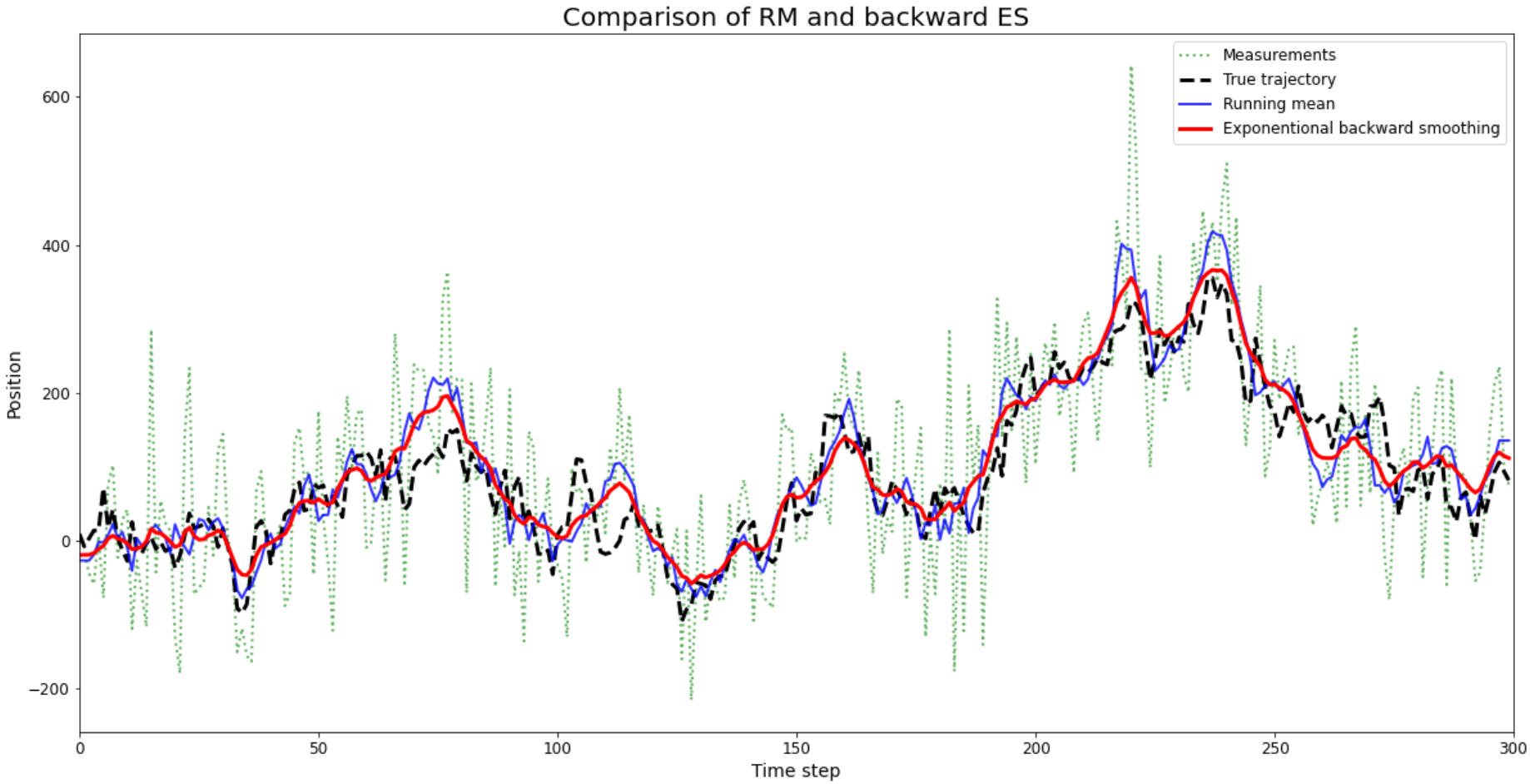
PART I

In [2]:
```
sigma_w_2 = 28 ** 2
sigma_n_2 = 97 ** 2
Points = 300
position = 10
hi = sigma_w_2 / sigma_n_2
alfa = (-hi + np.sqrt(hi ** 2 + 4 * hi)) / 2
M = 7

path = []
noise_path = []

measurements = []
noise_measurments = []

measurements_smoothed = np.zeros((1, Points))
measurements_smoothed_b = np.zeros((1, Points))
```

In [3]:
```
## Making plot and noise generating
for i in range(0, Points):
    noise_path.append(np.random.normal(0, np.sqrt(sigma_w_2)))
    position += noise_path[i]
    path.append(position)
    noise_measurments.append(np.random.normal(0, np.sqrt(sigma_n_2)))
    step = position + noise_measurments[i]
    measurements.append(step)

    if i == 0:
        measurements_smoothed[0, i] = measurements[0]
    else:
        measurements_smoothed[0, i] = measurements_smoothed[0, i - 1] + alfa*(measurements[i] - measurements_smoothed[0, i - 1])
```

In [4]:

```
#Running mean
running_mean = np.zeros((1, 300))
beg_RM = sum(measurements[0:3]) / 3
end_RM = sum(measurements[297:300]) / 3
for i in range(Points):
    if i <= 2:
        running_mean[0, i] = beg_RM
    elif i >= Points - 3:
        running_mean[0, i] = end_RM
    else:
        running_mean[0, i] = 1/M * sum(measurements[i - 3:i + 4])

#Exponential backward mean
measurements_smoothed_b[0, Points - 1] = measurements_smoothed[0, Points - 1]
for i in range(1, Points):
    measurements_smoothed_b[0, Points - i - 1] = measurements_smoothed_b[0, Points - i] + \
    alfa*(measurements_smoothed[0, Points - 1 - i] - measurements_smoothed_b[0, Points - i])

fig, ax = plt.subplots(figsize=(20, 10))
ax.set_title('Comparison of RM and backward ES', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)

ax.plot(measurements, ':g', alpha = 0.6,  linewidth = 2, label = 'Measurements')
ax.plot(path, '--k', linewidth = 3, alpha = 1, label = 'True trajectory')
ax.plot(running_mean[0, :], 'b', alpha = 0.8, linewidth = 2, label = 'Running mean')
ax.plot(measurements_smoothed_b[0, :], 'r', linewidth = 3, label = 'Exponentional backward smoothing')
plt.xlim(0, 300)
ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 12)
plt.savefig('different smoothing')
```



In [5]:
```
#Comparesing
Var_run = 0
Var_exp = 0
Deviation_run = sum((measurements[:] - running_mean[0, :]) ** 2)
Deviation_exp = sum((measurements[:] - measurements_smoothed_b[0, :]) ** 2)
print('Deviation of running mean:', Deviation_run)
print('Deviation of exponentional backward mean:', Deviation_exp)

for i in range(0, 298):
    Var_run += (running_mean[0, i + 2] - 2 * running_mean[0, i + 1] + running_mean[0, i]) ** 2
    Var_exp += (measurements_smoothed_b[0, i + 2] - 2 * measurements_smoothed_b[0, i + 1] + measurements_smoothed_b[0, i]) ** 2
```

```
print('Variability indicator of running mean:', Var_run)
print('Variability indicator of exponentional backward mean:', Var_exp)
```

```
Deviation of running mean: 2453961.81692039
Deviation of exponentional backward mean: 2260192.9168398734
Variability indicator of running mean: 233492.2899716213
Variability indicator of exponentional backward mean: 15683.586732913287
```

PART II

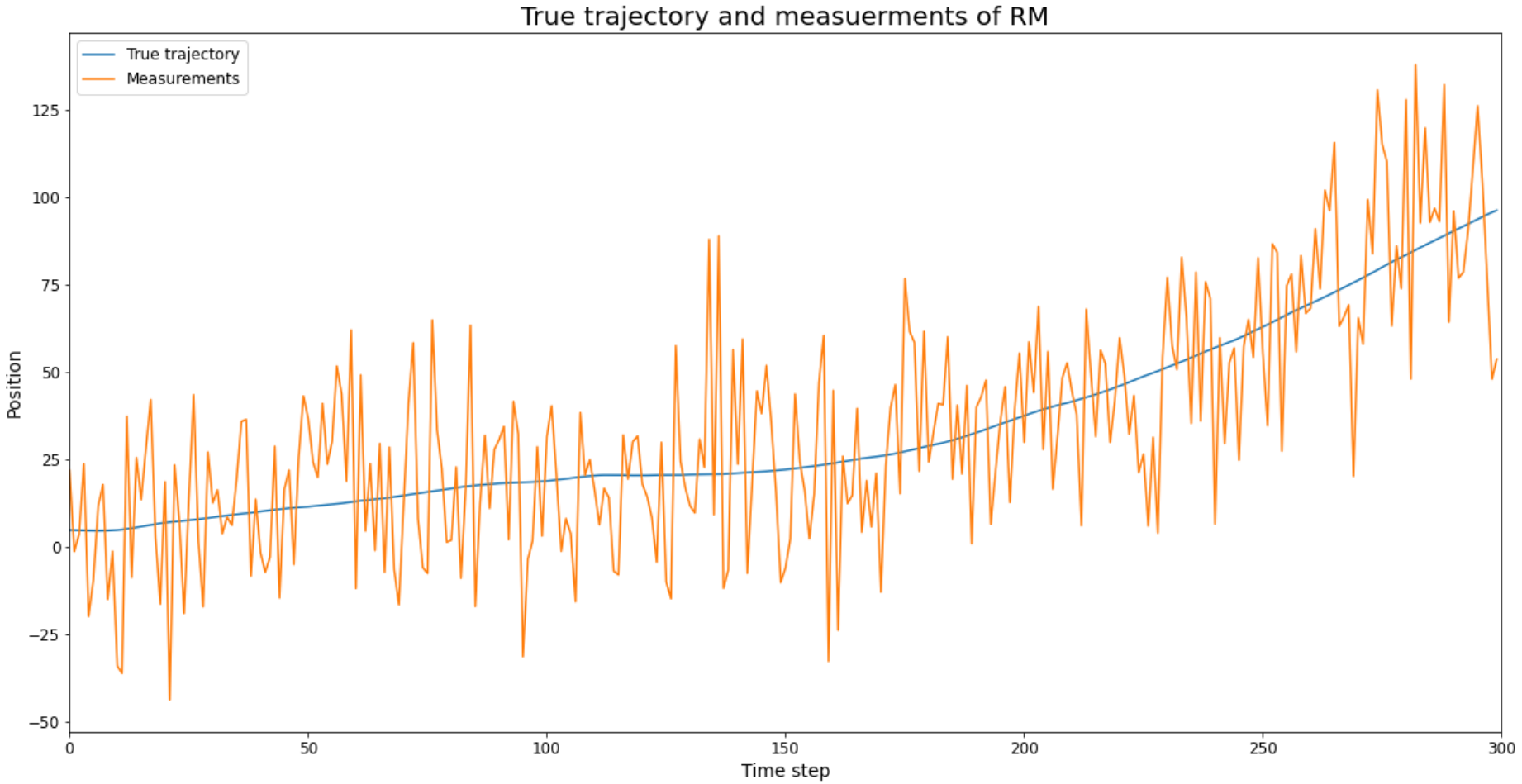
First trajectory

In [6]:
```
#Part 2
#First trajectory
size = 300
x0 = 5
v0 = 0
T = 0.1
sigma_a_2 = 10
sigma_eta_2 = 500
```

In [7]:
```
def motion(x_init, velocity_init):
    a = np.random.normal(0, np.sqrt(sigma_a_2))
    xi = x_init + velocity_init * T + a * (T ** 2) / 2
    vi = velocity_init + a * T
    zi = xi + np.random.normal(0, np.sqrt(sigma_eta_2))
    return [xi, vi, zi]
```

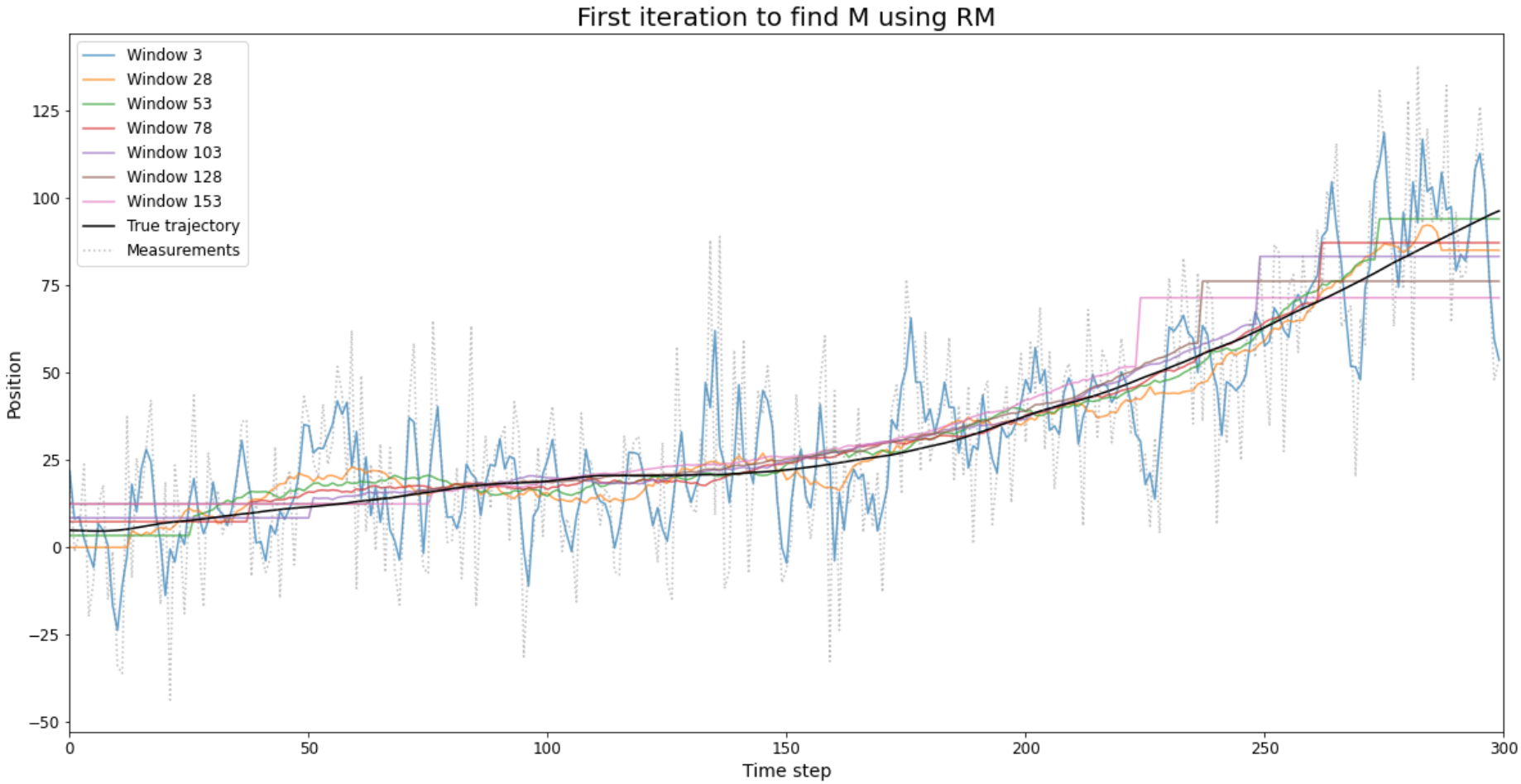
In [8]:
```
trajectory = []
measurements_2 = []

for i in range (size):
    if i == 0:
        motionn = motion(x0, v0)
        trajectory.append(motionn[0])
        velocity = motionn[1]
        measurements_2.append(motionn[2])
    else:
        motionn = motion(trajectory[i - 1], velocity)
        trajectory.append(motionn[0])
```

```
        velocity = motionn[1]
        measurements_2.append(motionn[2])

fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('True trajectory and measuerments of RM', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)
ax.plot(trajectory[:], label='True trajectory')
ax.plot(measurements_2[:], label='Measurements')
plt.xlim(0, 300)
ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 12)
plt.savefig('dont know')
```



In [9]:

```
#M definition with large options window
M = np.arange(3, 154, 25)
running_mean_M = np.zeros((len(M), 300))
beg_RM = np.zeros((len(M), 1))
end_RM = np.zeros((len(M), 1))
for i in range (len(M)):
    beg_RM[i] = sum(measurements_2[0:int((M[i] - 1) / 2)]) / ((M[i] - 1) / 2)
    end_RM[i] = sum(measurements_2[300 - int((M[i] - 1) / 2):300]) / ((M[i] - 1) / 2)

#Deviation
Deviation_M = np.zeros((len(M), 1))

for k in range(len(M)):
    for i in range(Points):
        if i <= int((M[k] - 1) / 2) - 1:
            running_mean_M[k, i] = beg_RM[k]
        elif i >= Points - int((M[k] - 1) / 2):
            running_mean_M[k, i] = end_RM[k]
        else:
            running_mean_M[k, i] = 1 / M[k] * sum(measurements_2[i - int((M[k] - 1) / 2):i + int((M[k] - 1) / 2) + 1])
    Deviation_M[k] = sum((measurements_2[:] - running_mean_M[k, :]) ** 2)

fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('First iteration to find M using RM', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)

for i in range(len(M)):
    ax.plot(running_mean_M[i, :], alpha = 0.7, label = 'Window '+ str(M[i]))
ax.plot(trajectory, 'k', markersize = 2, label = 'True trajectory')
ax.plot(measurements_2, ':', alpha = 0.5, label = 'Measurements')

ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 12)
plt.xlim(0, 300)
plt.savefig('RM_first')
a = min(Deviation_M)
```



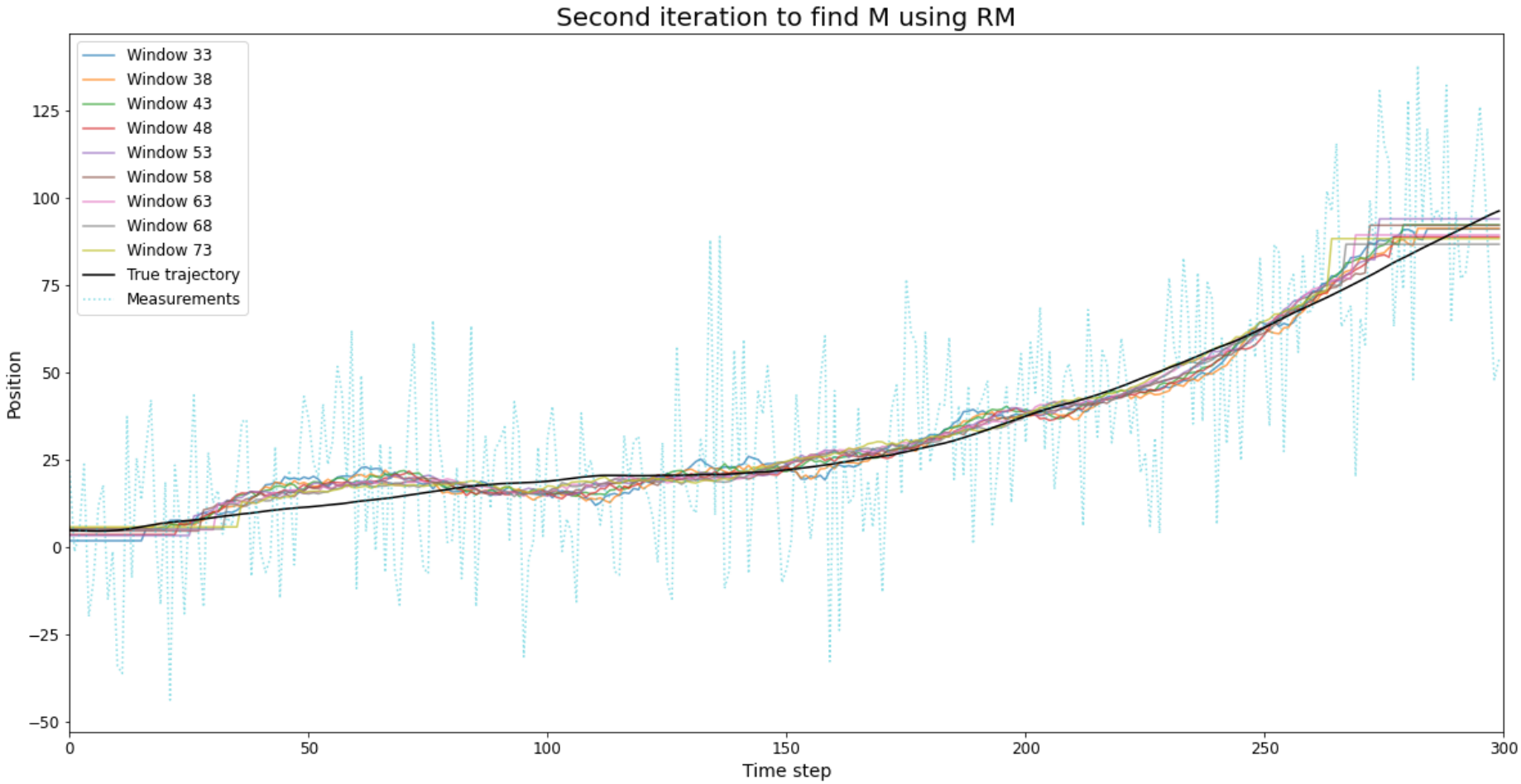
In [10]:
```
#Variability indicators and certain M determination
Var_M = np.zeros((len(M), 1))

for k in range(len(M)):
    for i in range(0, Points-2):
        Var_M[k] += (running_mean_M[k, i + 2] - 2 * running_mean_M[k, i + 1] + running_mean_M[k, i]) ** 2
M_precise = M[np.where(Var_M[:, 0] == np.min(Var_M))]
print(M[np.where(Var_M[:, 0] == np.min(Var_M))])
```

```
[53]
```

In [11]:
```
#M definition with small options window
M = np.arange(M_precise - 20, M_precise + 21, 5)
```

```
running_mean_M = np.zeros((len(M), 300))
beg_RM = np.zeros((len(M), 1))
end_RM = np.zeros((len(M), 1))
for i in range (len(M)):
    beg_RM[i] = sum(measurements_2[0:int((M[i] - 1) / 2)]) / ((M[i] - 1) / 2)
    end_RM[i] = sum(measurements_2[300 - int((M[i] - 1) / 2):300]) / ((M[i] - 1) / 2)

Deviation_M_final = np.zeros((len(M), 1))

for k in range(len(M)):
    for i in range(Points):
        if i <= int((M[k] - 1) / 2) - 1:
            running_mean_M[k, i] = beg_RM[k]
        elif i >= Points - int((M[k] - 1) / 2):
            running_mean_M[k, i] = end_RM[k]
        else:
            running_mean_M[k, i] = 1 / M[k] * sum(measurements_2[i - int((M[k] - 1) / 2):i + int((M[k] - 1) / 2) + 1])
    Deviation_M_final[k] = sum((measurements_2[:] - running_mean_M[k, :]) ** 2)

fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('Second iteration to find M using RM', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)

for i in range(len(M)):
    ax.plot(running_mean_M[i, :], alpha = 0.7, label = 'Window '+ str(M[i]))
ax.plot(trajectory, 'k', markersize = 2, label = 'True trajectory')
ax.plot(measurements_2, ':', alpha = 0.5, label = 'Measurements')

ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 12)
plt.xlim(0, 300)
plt.savefig('RM_second')

#Variability indicators and plots for precised M
Var_M_final = np.zeros((len(M), 1))

for k in range(len(M)):
    for i in range(0, Points-2):
        Var_M_final[k] += (running_mean_M[k, i+2] - 2*running_mean_M[k, i+1] + running_mean_M[k, i])**2

M_precise_final = M[np.where(Var_M_final[:, 0] == np.min(Var_M_final))]
print('The most appropriate window size M =', M_precise_final[0])
```

```
The most appropriate window size M = 48
```

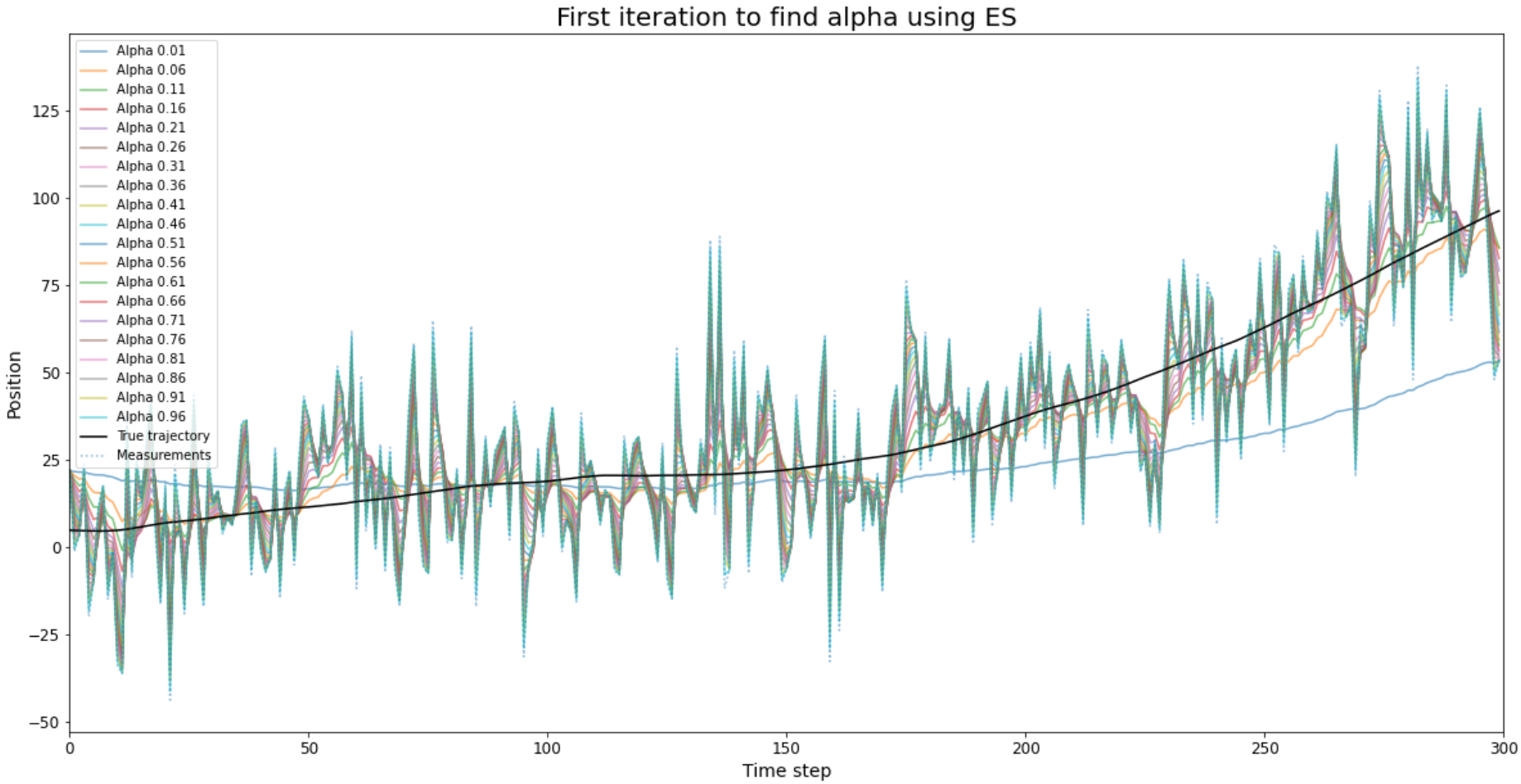


In [12]:
```python
#Alpha calculations with large options window
alpha = np.arange(0.01, 1.01, 0.05)
exp_smooth_a = np.zeros((len(alpha), 300))
Deviation_alpha = np.zeros((len(alpha), 1))

for k in range(len(alpha)):
    for n in range(size):
        if n == 0:
            exp_smooth_a[k, n] = measurements_2[0]
        else:
            exp_smooth_a[k, n] = exp_smooth_a[k, n - 1] + alpha[k] * (measurements_2[n] - exp_smooth_a[k, n - 1])
    Deviation_alpha[k] = sum((measurements_2[:] - exp_smooth_a[k, :]) ** 2)
```

```
#Variation calculating
Var_alpha = np.zeros((len(alpha), 1))

for k in range(len(alpha)):
    for i in range(0, Points - 2):
        Var_alpha[k] += (exp_smooth_a[k, i + 2] - 2 * exp_smooth_a[k, i + 1] + exp_smooth_a[k, i]) ** 2
```

In [13]:
```
#Normolized parameters and alpha determination
Var_alpha_norm = np.zeros((len(alpha), 1))
Deviation_alpha_norm = np.zeros((len(alpha), 1))
Compare_alpha = np.zeros((len(alpha), 1))

Var_alpha_norm[:] = Var_alpha[:] / sum(Var_alpha)
Deviation_alpha_norm[:] = Deviation_alpha[:] / sum(Deviation_alpha)
Compare_alpha [:] = Var_alpha_norm [:] + 0.05 * Deviation_alpha_norm [:]

alpha_precise = alpha[np.where(Compare_alpha[:, 0] == np.min(Compare_alpha))]
```

In [14]:
```
fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('First iteration to find alpha using ES', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)

for i in range(len(alpha)):
    ax.plot(exp_smooth_a[i, :], alpha = 0.6, label = 'Alpha '+ str(round(alpha[i], 2)))
ax.plot(trajectory, 'k', markersize = 2, label = 'True trajectory')
ax.plot(measurements_2, ':', alpha = 0.5, label = 'Measurements')

ax.tick_params(axis='both', labelsize=12)
plt.legend()
plt.xlim(0, 300)
plt.savefig('ES_first')
```



According to recieved results, the best smoothing occures when alpha is smaller than 0.2 that's why the range 0 < alpha < 0.2 should be considered

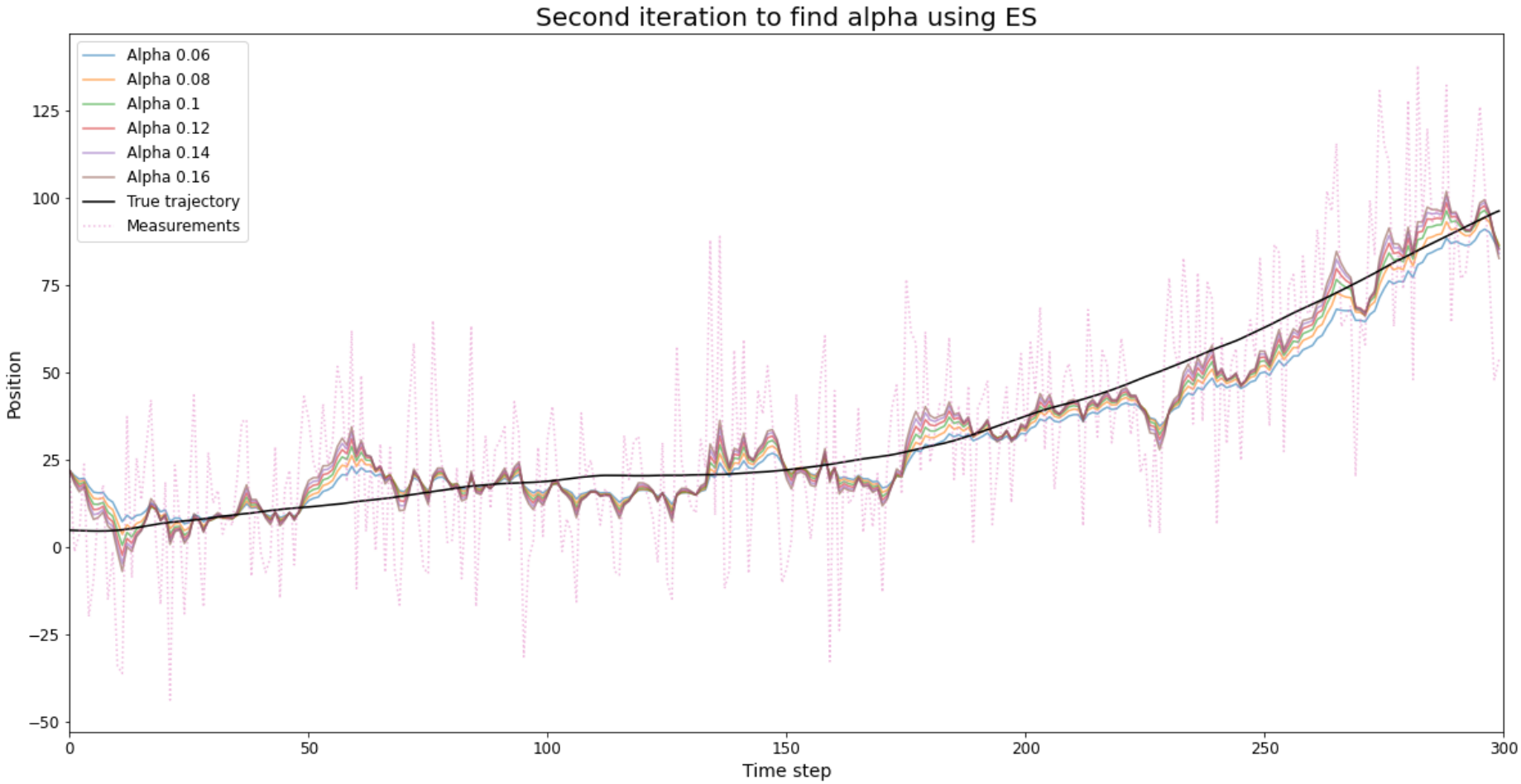
In [15]:
```
#Alpha with small options window
if (alpha_precise-0.05) >= 0:
    alpha = np.arange(alpha_precise - 0.05 , alpha_precise + 0.06, 0.02)
else:
    alpha = np.arange(0, alpha_precise + 0.06, 0.02)
exp_smooth_a_final = np.zeros((len(alpha), 300))
Deviation_alpha_final = np.zeros((len(alpha), 1))
Var_alpha_final = np.zeros((len(alpha), 1))
```

```
for k in range(len(alpha)):
    for n in range(size):
        if n == 0:
            exp_smooth_a_final[k, n] = measurements_2[0]
        else:
            exp_smooth_a_final[k, n] = exp_smooth_a_final[k, n - 1] + alpha[k] * (measurements_2[n] - exp_smooth_a_final[k, n - 1
    Deviation_alpha_final[k] = sum((measurements_2[:] - exp_smooth_a_final[k, :]) ** 2)

#Variation calculating
for k in range(len(alpha)):
    for i in range(0, Points-2):
        Var_alpha_final[k] += (exp_smooth_a_final[k, i + 2] - 2 * exp_smooth_a_final[k, i + 1] + exp_smooth_a_final[k, i]) ** 2

Var_alpha_final_norm = np.zeros((len(alpha), 1))
Deviation_alpha_final_norm = np.zeros((len(alpha), 1))
Compare_alpha_final = np.zeros((len(alpha), 1))

Var_alpha_final_norm[:] = Var_alpha_final[:] / sum(Var_alpha_final)
Deviation_alpha_final_norm[:] = Deviation_alpha_final[:] / sum(Deviation_alpha_final)
Compare_alpha_final [:] = 0.25 * Var_alpha_final_norm [:] + Deviation_alpha_final_norm [:]
```

In [16]:

```
#Plotting new alpha
fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('Second iteration to find alpha using ES', fontsize = 20)
ax.set_ylabel('Position', fontsize = 14)
ax.set_xlabel('Time step', fontsize = 14)

for i in range(len(alpha)):
    ax.plot(exp_smooth_a_final[i, :], alpha = 0.6, label = 'Alpha '+ str(round(alpha[i], 2)))
ax.plot(trajectory, 'k', markersize = 2, label = 'True trajectory')
ax.plot(measurements_2, ':', alpha = 0.5, label = 'Measurements')

ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 12)
plt.xlim(0, 300)
plt.savefig('ES_second')

alpha_final = alpha[np.where(Compare_alpha_final[:, 0] == np.min(Compare_alpha_final))]
print('The most appropriate alpha =', round(alpha_final[0], 2))
```

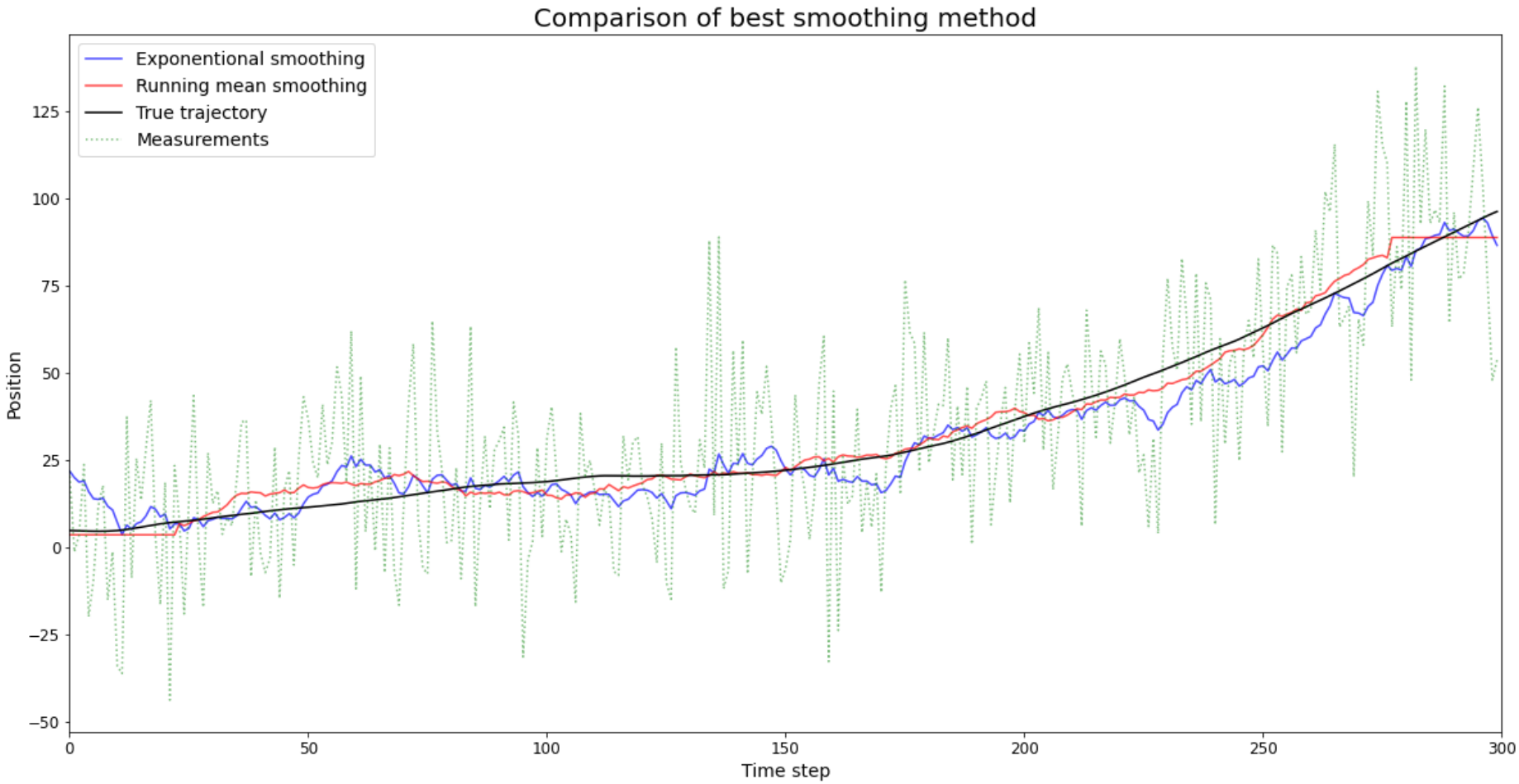
```
The most appropriate alpha = 0.08
```



In [17]:
```
#Best smoothing method smoothing
print('Running mean deviation =', Deviation_M_final[np.where(Var_M_final[:, 0] == np.min(Var_M_final))][0][0])
print('Exponentional smoothing deviation =', Deviation_alpha_final[np.where(Compare_alpha_final[:, 0]
                                                                            == np.min(Compare_alpha_final))][0][0])

print('Running mean variability indicator =', Var_M_final[np.where(Var_M_final[:, 0] == np.min(Var_M_final))][0][0])
print('Exponentional smoothing variability indicator =', Var_alpha_final[np.where(Var_alpha_final[:, 0]
                                                                                  == np.min(Var_alpha_final))][0][0])

fig, ax = plt.subplots(figsize = (20,10))
ax.set_title('Comparison of best smoothing method', fontsize = 20)
ax.set_xlabel('Time step', fontsize = 14)
ax.set_ylabel('Position', fontsize = 14)
```

```
ax.plot(exp_smooth_a_final[(np.where(Compare_alpha_final[:, 0] == np.min(Compare_alpha_final)))[0][0], :],
        'b', markersize = 2, alpha = 0.7, label = 'Exponentional smoothing')
ax.plot(running_mean_M[np.where(Var_M_final[:, 0] == np.min(Var_M_final))[0][0], :], 'r', markersize = 2,
        alpha = 0.7, label = 'Running mean smoothing')
ax.plot(trajectory, 'k', markersize = 3, label = 'True trajectory')
ax.plot(measurements_2, ':g', alpha = 0.5, label = 'Measurements')

ax.tick_params(axis='both', labelsize=12)
plt.legend(fontsize = 14)
plt.xlim(0, 300)
plt.savefig('comparison')
```

```
Running mean deviation = 148737.40021970574
Exponentional smoothing deviation = 140913.76581930774
Running mean variability indicator = 311.01151824918213
Exponentional smoothing variability indicator = 1168.9219350282438
```



Second trajectory

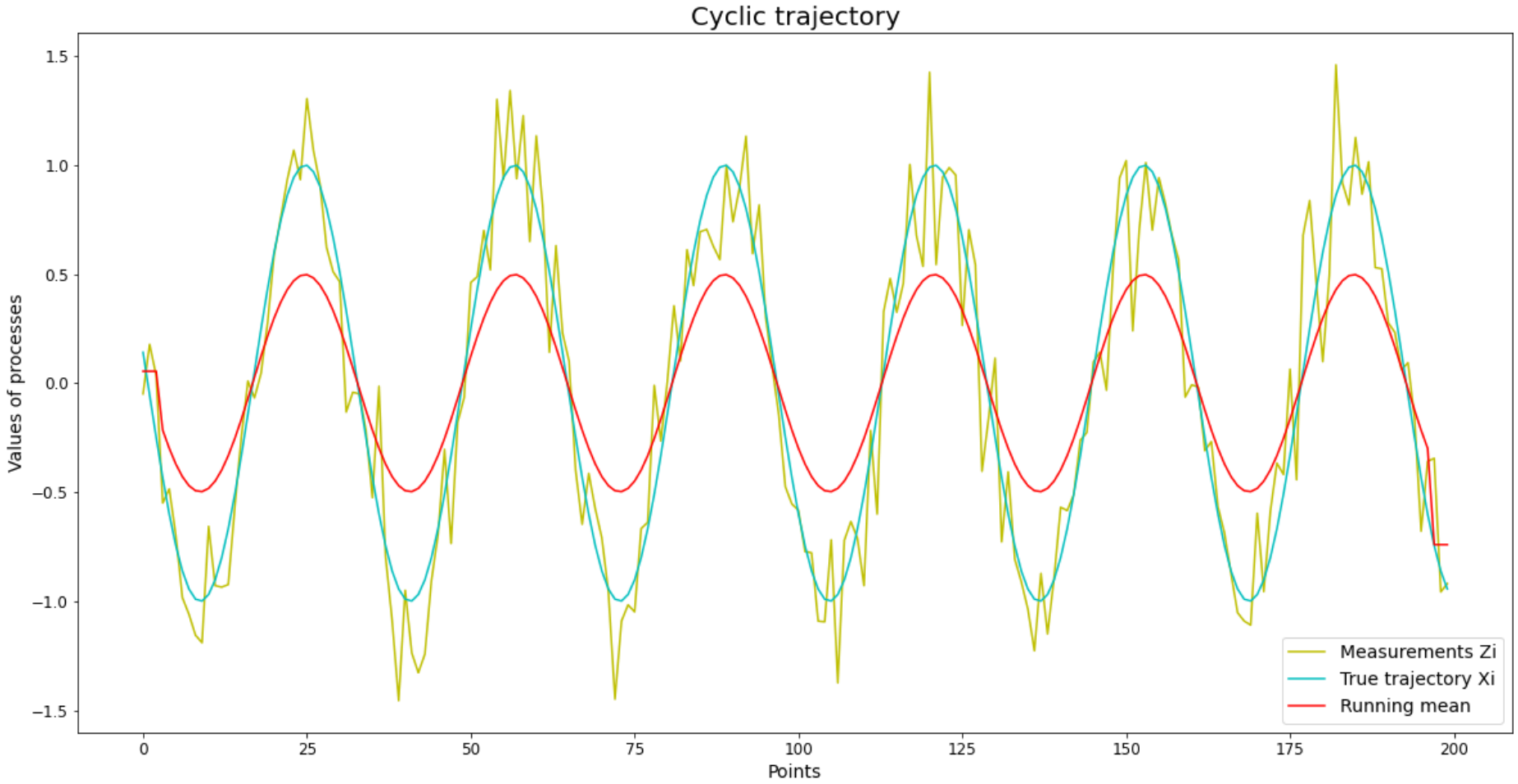
In [18]:
```
#Size of trajectory
n = 200

#Intialization of arrays
x3 = np.zeros((n, 1))
z3 = np.zeros((n, 1))
A = np.zeros((n, 1))

A[0]= 1
```

```
M = 15
# The period of the true trajectory
T = 32
M_half = int(M/2)

#Variance of noise
sigma_w = 0.08**2
sigma_eta2 = 0.05

w = np.random.normal(0, np.sqrt(sigma_w), n - 1)
eta2 = np.random.normal(0, np.sqrt(sigma_eta2), n)
run_mean = np.zeros((n, 1))

# Generation of A
for i in range(1,len(A)-1):
    A[i] = A[i - 1] + w[i]

# Generation of true trajectory Xi
for i in range(len(x3)):
    x3[i] = A[0]= 1*np.sin((np.pi * 2 * (i) / T) + 3)

# Generation of measurements Zi of the process Xi
for i in range(len(z3)):
    z3[i] = x3[i] + eta2[i]
```

In [19]:
```
#Running Mean
M_rm = 13
RM = np.zeros((n, 1))
beg_RM = np.sum(z3[:3]) / len(z3[:3])
end_RM = np.sum(z3[len(z3) - 3:]) / len(z3[len(x3) - 3:])
for i in range(len(z3)):
    if i <= 2:
        RM[i] = beg_RM
    elif i >= len(z3) - 3:
        RM[i] = end_RM
    else:
        RM[i] = np.sum(x3[i - 3: i + 4]) / M_rm
```

In [20]:
```
fig, ax = plt.subplots(figsize=(20, 10))
ax.set_title("Cyclic trajectory", fontsize = 20)
ax.set_ylabel("Values of processes", fontsize = 14)
ax.set_xlabel("Points", fontsize = 14)
ax.plot(z3, label = 'Measurements Zi', color = "y")
```

```
ax.plot(x3, label = 'True trajectory Xi', color = "c")
ax.plot(RM, label = 'Running mean', color = "r")
ax.legend(fontsize='14')
ax.tick_params(axis='both', labelsize=12)
plt.savefig('cyclic trajectory')
```

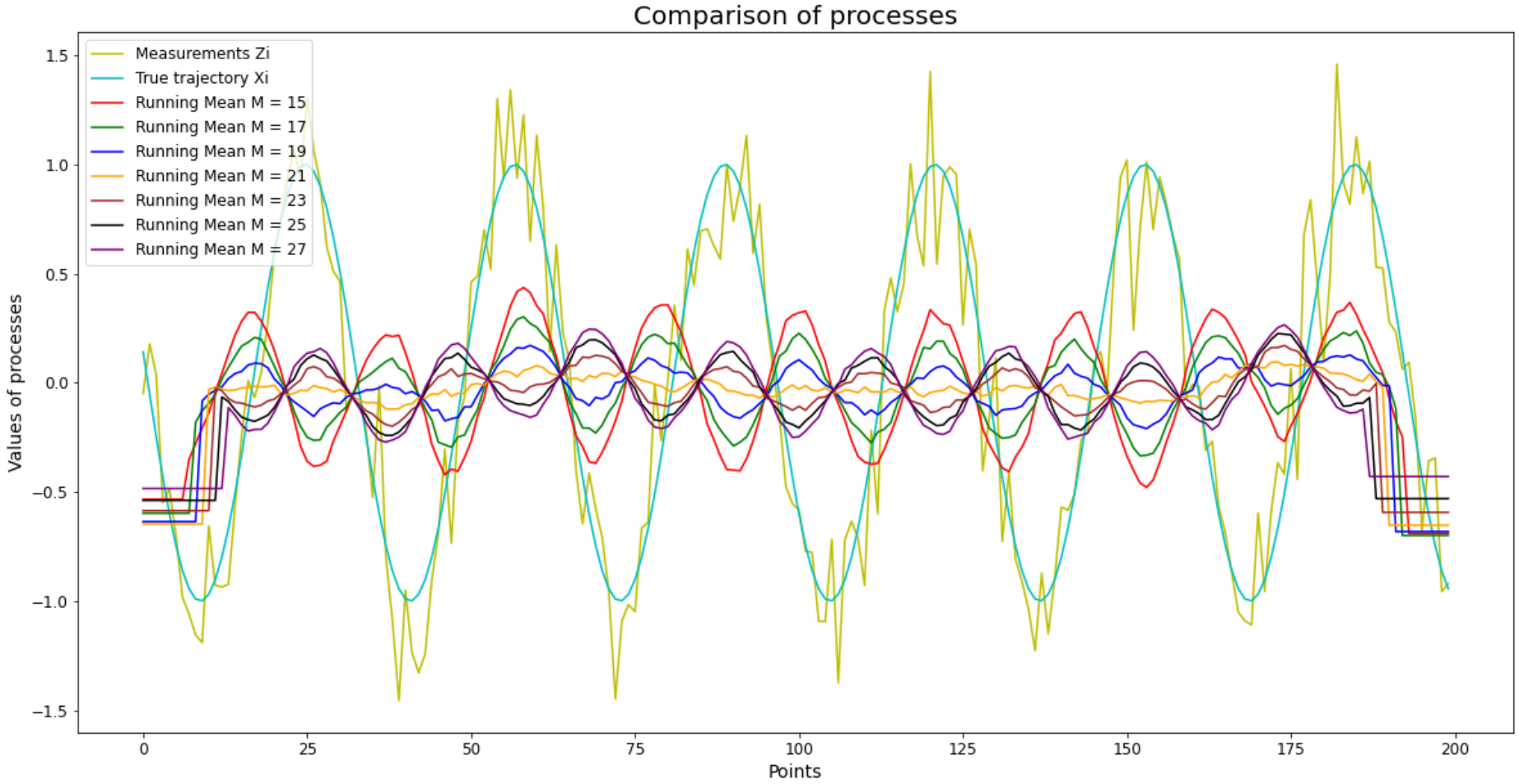


In [21]:
```
#Intialization of arrays
x7 = np.zeros((n, 1))
z7 = np.zeros((n, 1))

M7 = 15
# The period of the true trajectory
T7 = 21
```

```
M_half7 = int(M7/2)

# Generation of true trajectory Xi
for i in range(len(x3)):
    x7[i] = A[0]= 1 * np.sin((np.pi * 2 * (i) / T7) + 3)

# Generation of measurements Zi of the process Xi
for i in range(len(z3)):
    z7[i] = x7[i] + eta2[i]
```

In [22]:
```
# definition Running Mean function
def running_mean(M_half, z3):
    beg_mean = np.sum(z3[:M_half]) / len(z3[:M_half])
    end_mean = np.sum(z3[len(z3)-M_half:]) / len(z3[len(z3)-M_half:])

    for i in range(len(z3)):
        if i <= M_half - 1:
            run_mean[i] = beg_mean
        elif i >= len(z3) - M_half:
            run_mean[i] = end_mean
        else:
            run_mean[i] = np.sum(z3[i - M_half:i + M_half + 1]) / len(z3[i - M_half:i + M_half + 1])
    return run_mean
```

In [23]:
```
fig, ax = plt.subplots(figsize=(20, 10))
ax.set_title("Comparison of processes", fontsize = 20)
ax.set_ylabel("Values of processes", fontsize = 14)
ax.set_xlabel("Points", fontsize = 14)
ax.plot(z3, label = "Measurements Zi", color = "y")
ax.plot(x3, label = "True trajectory Xi", color = "c")
ax.plot(running_mean(M_half, z7), label = 'Running Mean M = 15', color = 'red')
ax.plot(running_mean(M_half + 1, z7), label = 'Running Mean M = 17', color = 'green')
ax.plot(running_mean(M_half + 2, z7), label = 'Running Mean M = 19', color = 'blue')
ax.plot(running_mean(M_half + 3, z7), label = 'Running Mean M = 21', color = 'orange')
ax.plot(running_mean(M_half + 4, z7), label = 'Running Mean M = 23', color = 'brown')
ax.plot(running_mean(M_half + 5, z7), label = 'Running Mean M = 25', color = 'black')
ax.plot(running_mean(M_half + 6, z7), label = 'Running Mean M = 27', color = 'purple')
ax.tick_params(labelsize = 12)
ax.legend(fontsize = 12)
plt.savefig('comparison cyclic')
```


Comparison of processes
Measurements Zi
True trajectory Xi
Running Mean M = 15
Running Mean M = 17
Running Mean M = 19
Running Mean M = 21
Running Mean M = 23
Running Mean M = 25
Running Mean M = 27
Values of processes
Points
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
25
50
75
100
125
150
175
200