Assignment2

October 7, 2019

1 Assignment 2: Comparison of the exponential and running mean for random walk model

1.1 Team 6:

1. Angelina Prokopeva
2. Nikita Gorbadey
3. Mark Griguletskii
4. Stanislav Savushkin

03.10.2019, Skoltech

1.2 Working progress

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import plotly.graph_objects as go
        from IPython.display import Image
        import os

        if not os.path.exists("img"):
            os.mkdir("img")
```

Supplementary function for grapics display

```
In [2]: generate_report = True
        def plot(trace_num, x_data, y_data, xlable = 'xlable', ylable = 'ylable',
                      legend = 'legend', title = 'title', mode='lines'):
            plot.counter += 1
            fig_name = 'images/' + str(plot.counter) + '.jpg'
            fig = go.Figure()
            for i in range(trace_num):
                fig.add_trace(go.Scatter(x=x_data[i], y=y_data[i], mode=mode, name=legend[i], s
            fig.update_layout(
                title=go.layout.Title(
                    text=title,
                ),
```

```
                xaxis=go.layout.XAxis(
                    title=go.layout.xaxis.Title(
                        text=xlable
                    )
                ),
                yaxis=go.layout.YAxis(
                    title=go.layout.yaxis.Title(
                        text=ylable
                    )
                )
            )
            if generate_report is True:
                fig.write_image(fig_name)
                display(Image(fig_name))
            else:
                fig.show()
        plot.counter = 0
```

Supplementary function for exponential mean

```
In [3]: def exp_mean(data, alpha, init=0):
            out = np.empty(data.size)
            out[0] = init
            for i in range(1, data.size):
                out[i] = out[i-1] + alpha*(data[i]-out[i-1])
            return out
```

Supplementary function for running mean

```
In [4]: def run_mean(data, M, first_M, last_M):
            n = round((M-1)/2)
            out = np.empty(data.size)
            out[:n] = np.ones(n)*first_M
            out[-n:] = np.ones(n)*last_M
            for i in range(n, data.size - n):
                out[i] = 1/M*np.sum(data[i-n:i+n+1])
            return out
```

2 First part

2.1 Trajectories with 3000 points

```
In [5]: num = 3000
        sigma_w_given = 13**0.5
        sigma_n_given = 8**0.5
```

Generation a true trajectory using the random walk model

In [6]:
```
X = np.empty(num)
X[0] = 10

# normally distributed random noise with zero mathematical
# expectation and variance sigma^2 = 13
w = np.random.normal(loc=0, scale=sigma_w_given, size=num)

# random walk model
for i in range(1, num):
    X[i] = X[i-1] + w[i]
num_points = np.linspace(1, num, num=num)
```

In [7]:
```
plot(1, [num_points], [X], mode='lines', title='{} points random trajectory'.format(nu
     xlable = 'point', ylable = 'value', legend = ['true trajectory'])
```

Generating measurements of the process

In [8]:
```
# normally distributed random noise with zero mathematical expectation
# and variance sigma^2 = 8
n = np.random.normal(loc=0, scale=sigma_n_given, size=num)

# measurements generation
z = X + n
```

Variance calculation out of formulas from slides

In [9]:
```
# residuals calculation
v = np.empty(num)
p = np.empty(num)
v[0] = z[0]
for i in range(1, num):
    v[i] = z[i] - z[i-1]
p[0] = z[0]
p[1] = z[1]
for i in range(2, num):
    p[i] = z[i] - z[i-2]

# math expectation calculation
E_v = 1/(num-1) * np.sum(v[2:]**2)
E_p = 1/(num-2) * np.sum(p[3:]**2)

# variance calculation for random parameters
sigma_w = E_p - E_v
sigma_n = (2*E_v - E_p)/2
print('sigma_w = {}, sigma_n = {}'.format(sigma_w, sigma_n))
print('Referal values are 13 and 8 correspondingly.')
```

```
sigma_w = 12.714032425517495, sigma_n = 8.164112151234697
Referal values are 13 and 8 correspondingly.
```

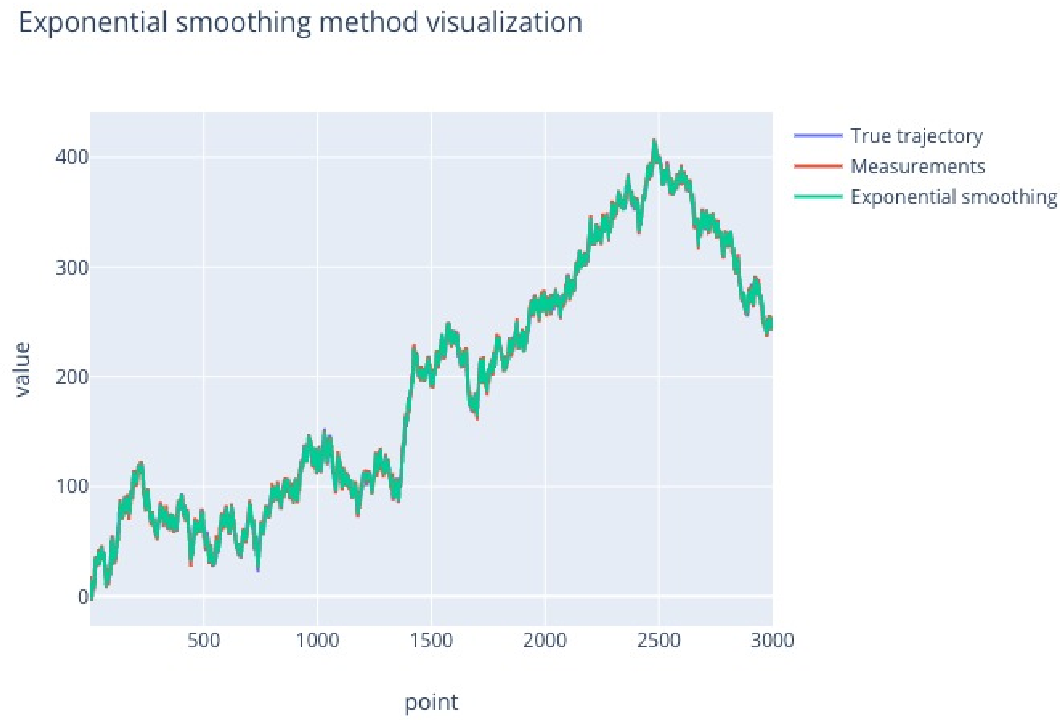
We can see that for 3000 points trajectory calculated variances are close to given ones
Now calculating optimal smoothing coefficient in exponential smoothing

In [10]:
```
ksi = sigma_w / sigma_n
alpha = (-ksi + (ksi**2 + 4*ksi)**0.5)/2
print('Exponential smoothing parameter = {}'.format(alpha))
```

```
Exponential smoothing parameter = 0.6922674367296673
```

In [11]:
```
X_smooth = exp_mean(z, alpha, z[0])
```

In [12]:
```
plot(3, [num_points, num_points, num_points], [X, z, X_smooth],
        legend=['True trajectory', 'Measurements', 'Exponential smoothing'],
        title='Exponential smoothing method visualization',
        xlable = 'point', ylable = 'value')
```



2.2 Now repeat the same for 300 points trajectory

```
In [13]: num = 300
         sigma_w_given = 13**0.5
         sigma_n_given = 8**0.5
```

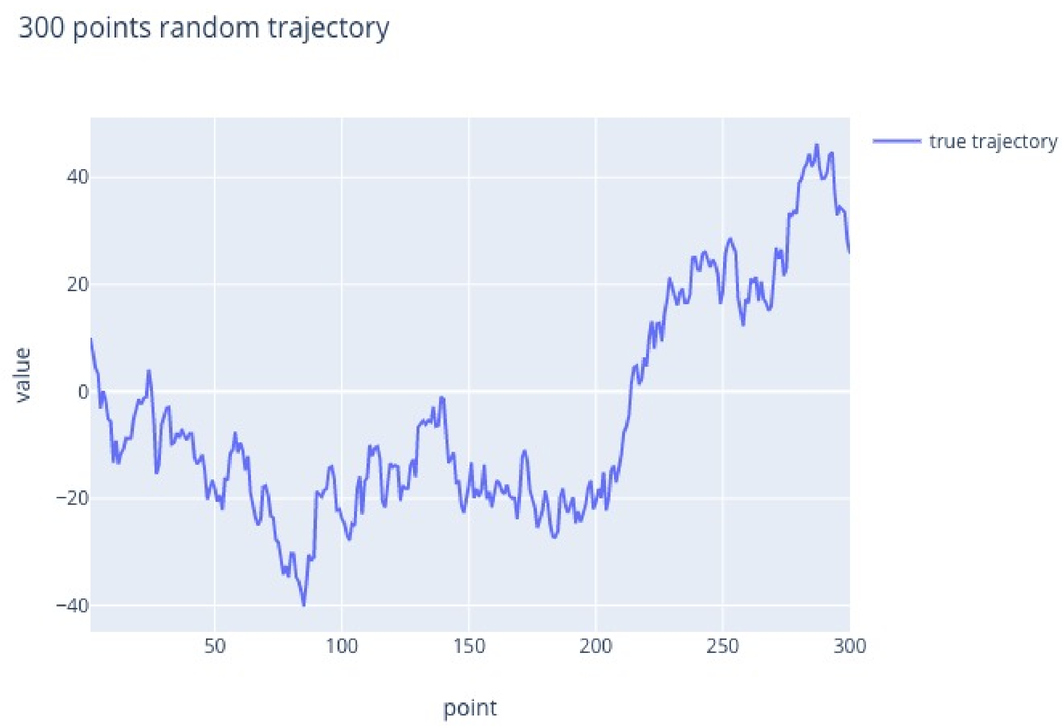
Generation a true trajectory using the random walk model

```
In [14]: X = np.empty(num)
         X[0] = 10

         # normally distributed random noise with zero mathematical
         # expectation and variance sigma^2 = 13
         w = np.random.normal(loc=0, scale=sigma_w_given, size=num)

         # random walk model
         for i in range(1, num):
             X[i] = X[i-1] + w[i]
         num_points = np.linspace(1, num, num=num)
```

```
In [15]: plot(1, [num_points], [X], mode='lines', title='{} points random trajectory'.format(nu
              xlable = 'point', ylable = 'value', legend = ['true trajectory'])
```



Generating measurements of the process

```
In [16]: # normally distributed random noise with zero mathematical expectation
         # and variance sigma^2 = 8
         n = np.random.normal(loc=0, scale=sigma_n_given, size=num)

         # measurements generation
         z = X + n
```

Variance calculation out of formulas from slides

```
In [17]: # residuals calculation
         v = np.empty(num)
         p = np.empty(num)
         v[0] = z[0]
         for i in range(1, num):
             v[i] = z[i] - z[i-1]
         p[0] = z[0]
         p[1] = z[1]
         for i in range(2, num):
             p[i] = z[i] - z[i-2]

         # math expectation calculation
```

```
E_v = 1/(num-1) * np.sum(v[2:]**2)
E_p = 1/(num-2) * np.sum(p[3:]**2)

# variance calculation for random parameters
sigma_w = E_p - E_v
sigma_n = (2*E_v - E_p)/2
print('sigma_w = {}, sigma_n = {}'.format(sigma_w, sigma_n))
print('Referal values are 13 and 8 correspondingly.')
```

```
sigma_w = 10.563972499182377, sigma_n = 7.683436528455495
Referal values are 13 and 8 correspondingly.
```

We can see that for 300 points trajectory calculated variances are not as close to referal ones as for 3000 points trajectory. We imply that exponential smoothing method works better for larger datasets, keeping other parameters the same.

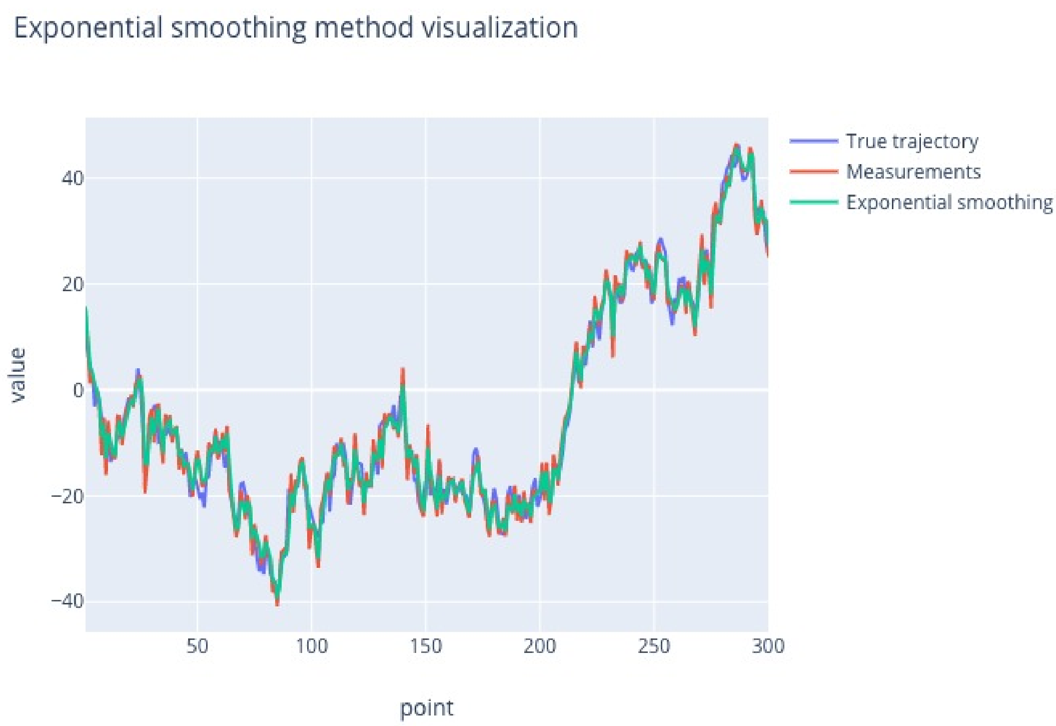
Now calculating optimal smoothing coefficient in exponential smoothing

```
In [18]: ksi = sigma_w / sigma_n
         alpha = (-ksi + (ksi**2 + 4*ksi)**0.5)/2
         print('Exponential smoothing parameter = {}'.format(alpha))
```

```
Exponential smoothing parameter = 0.671773370018545
```

```
In [19]: X_smooth = exp_mean(z, alpha, z[0])

In [20]: plot(3, [num_points, num_points, num_points], [X, z, X_smooth],
              legend=['True trajectory', 'Measurements', 'Exponential smoothing'],
              title='Exponential smoothing method visualization',
              xlable = 'point', ylable = 'value')
```



2.3 Second part

Comparison of methodical errors of exponential and running mean.
Trajectory parameters

In [21]:
```
num = 300
sigma_w = 28
sigma_n = 97
num_points = np.linspace(1, num, num=num)
```

Generation of a true trajectory and its measurements z using the random walk model

In [22]:
```
X = np.empty(num)
X[0] = 10

# normally distributed random noise with zero mathematical
# expectation and variance sigma^2 = 13
w = np.random.normal(loc=0, scale=sigma_w, size=num)

# random walk model
for i in range(1, num):
    X[i] = X[i-1] + w[i]
```

```
# measurements generation
n = np.random.normal(loc=0, scale=sigma_n, size=num)
z = X + n

# optimal smoothing coefficient  determination
ksi = sigma_w**2/sigma_n**2
alpha = (-ksi + (ksi**2 + 4*ksi)**0.5)/2
print('Exponential smoothing parameter alpha = {}'.format(alpha))
```

```
Exponential smoothing parameter alpha = 0.24998861233121078
```

Calculation window size M such as theoretical variance of running mean method is equel to exponential smoothing one

```
In [23]: M = round((2-alpha)/alpha)
         print('Running mean window size M = {}'.format(M))
```

```
Running mean window size M = 7
```

Theoratical variance of running mean and exponential smoothing methods

```
In [24]: sigma2_rm = sigma_n**2/M
         sigma2_es = sigma_n**2*alpha/(2-alpha)
         print('RM variance = {}, ES variance = {}'.format(sigma2_rm, sigma2_es))
```

```
RM variance = 1344.142857142857, ES variance = 1344.0728843243012
```

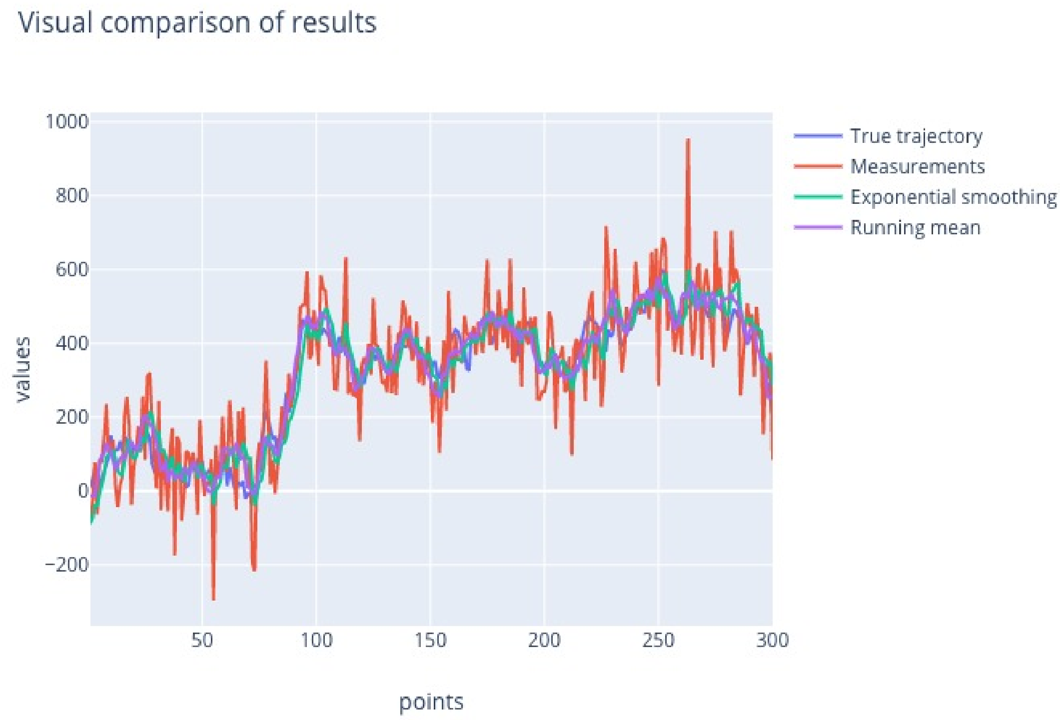
We cans see that theoretical variances are about the same.
Performing exponential smoothing methos

```
In [25]: X_ES = exp_mean(z, alpha, z[0])
```

Performing running mean method

```
In [26]: X_RM = run_mean(z, M, np.mean(z[:round((M-1)/2)]), np.mean(z[-round((M-1)/2):]))
```

```
In [27]: plot(4, [num_points, num_points, num_points, num_points], [X, z, X_ES, X_RM],
              legend=['True trajectory', 'Measurements', 'Exponential smoothing', 'Running mea
              title='Visual comparison of results', xlable='points', ylable='values')
```



Finaly, lets determine the variance of deviation of smoothed data from the true one.

```
In [28]: RM_err = X_RM - X
         ES_err = X_ES - X
         RM_var = np.var(RM_err, ddof = 1)
         ES_var = np.var(ES_err, ddof = 1)

         print("Running mean variance", float("{:.2f}".format(RM_var)))
         print("Exp smoothing variance", float("{:.2f}".format(ES_var)))
```

```
Running mean variance 1736.57
Exp smoothing variance 2323.54
```

2.4 Conclusion

Bases on visual analysis, there is a tendency of shifting in Forward Exponential Smoothing method. The running mean method has much less shifting error. By calculating variances of both methods we obtained exact result: running mean performs more accurately.

From first part we noticed very important thing: exponential smoothing performs better for larger datasets, keeping other parameters the same

In this lab, we learned about two methods of data processing: running mean and exponential smoothing. The second method has an advantage over the first: the averaging takes into account all previous points with different weights varying exponentially, while the first method takes into account measurements only by window size. For different signals, the same window size will have a different effect on the quality of averaging and noise elimination. However, the disadvantage of exponential smoothing is the shift relative to the actual trajectory and measurements. This methodical error can lead to delay or advance of a signal. The running mean method has almost no such methodological error. By changing the smoothing factor, it is possible to adapt such a filter to different input signals: smoother or frequently changing.