In [1]:

```
# Assignment 2
# Comparison of the exponential and running mean for random walk model.
# Team 2:
#     Ekaterina Karmanova
#     Timur Chikichev
#     Iaroslav Okunevich
#     Nikita Mikhailovskiy
#
# Skoltech, 03.10.2019
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

In [3]:

```
#Part 1
#1.1
# Array with σw2 and ση2
sigmasTrue=[np.sqrt(9),np.sqrt(12)]
#w-normally distributed random noise with zero mathematical expectation and variance =
9.
w3000= np.random.normal(0, sigmasTrue[0], 3000)
w300= np.random.normal(0, sigmasTrue[0], 300)
#TrueTrajectory() - generate true trajectory.
def TrueTrajectory(size,w):
        array=np.zeros((size))
        array[0]=10
        for i in range(1,size):
                array[i]=array[i-1]+w[i]
        return array

x3000=TrueTrajectory(3000,w3000)
x300=TrueTrajectory(300,w300)
```

In [4]:

```
#1.2
#ny-normally distributed random noise with zero mathematical expectation and variance =
12.
ny3000= np.random.normal(0, sigmasTrue[1], 3000)
ny300= np.random.normal(0, sigmasTrue[1], 300)
#Measurements() - generate measurements.
def Measurements(x,ny):
        z=np.zeros(len(x))
        for i in range(0, len(x)):
            z[i]=x[i]+ny[i]
        return z
z3000=Measurements(x3000,ny3000)
z300=Measurements(x300,ny300)
```

In [5]:

```
#2
#FindSigma() - identify σw2 and ση2 using formulas (5) and (6).
def FindSigma(w,n):
    count = len(w)
    #Finding E[v^2]
    Ev2 = 0
    for k in range (1,count):
        Ev2 += (w[k]+n[k]-n[k-1])**2
    Ev2 /= (count-1)
    #Finding E[r^2]
    Er2 = 0
    for k in range (2,count):
        Er2 += (w[k]+w[k-1]+n[k]-n[k-2])**2
    Er2 /= (count-2)
    #Calculating sigmas
    sigmas=np.zeros(2)
    sigmas[0] = Er2 - Ev2
    sigmas[1] = Ev2 - Er2/2
    return sigmas
sigmas3000=FindSigma(w3000,ny3000)
print('σw2 and ση2 using formulas (3) and (4) for 3000 points=',sigmas3000)
sigmas300=FindSigma(w300,ny300)
print('σw2 and ση2 using formulas (3) and (4) for 300 points=',sigmas300)
```

```
σw2 and ση2 using formulas (3) and (4) for 3000 points= [ 9.99725154 11.94
585732]
σw2 and ση2 using formulas (3) and (4) for 300 points= [11.73757831 11.633
66229]
```

In [6]:

```
#Compare the accuracy of estimation.
ErrorSigma3000 =[abs((np.sqrt(sigmas3000[0])-sigmasTrue[0])/sigmasTrue[0]),abs((np.sqrt
(sigmas3000[1])-sigmasTrue[1])/sigmasTrue[1])]
ErrorSigma300 =[abs((np.sqrt(sigmas300[0])-sigmasTrue[0])/sigmasTrue[0]),abs((np.sqrt(s
igmas300[1])-sigmasTrue[1])/sigmasTrue[1])]
print('Error of σw2={}% ση2={}% for 3000 points'.format(int(ErrorSigma3000[0]*100),int(
ErrorSigma3000[1]*100)))
print('Error of σw2={}% ση2={}% for 300 points'.format(int(ErrorSigma300[0]*100),int(Er
rorSigma300[1]*100)))
```

```
Error of σw2=5% ση2=0% for 3000 points
Error of σw2=14% ση2=1% for 300 points
```

In [7]:

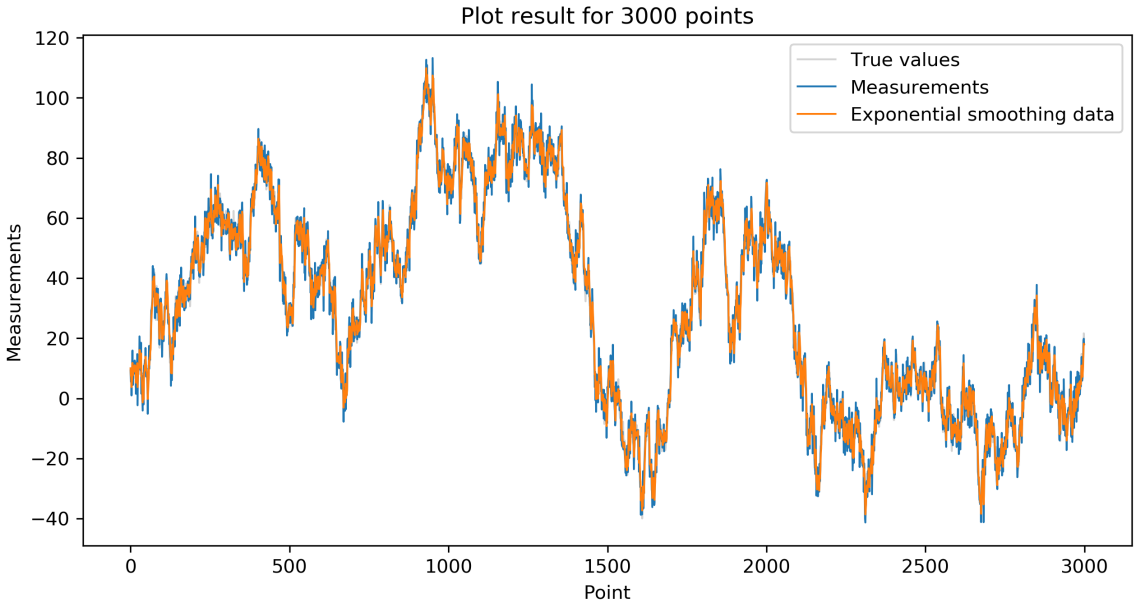
```
#3
#Alfa() - determine optimal smoothing coefficient in exponential smoothing.
def Alfa(sigma):
    psi=sigma[0]**2/sigma[1]**2
    print(psi)
    return (-psi+np.sqrt(psi**2+4*psi))/2
alf=Alfa(sigmasTrue)
print('Optimal smoothing coefficient=',alf)
```

```
0.7500000000000001
Optimal smoothing coefficient= 0.5687293044088437
```

In [8]:

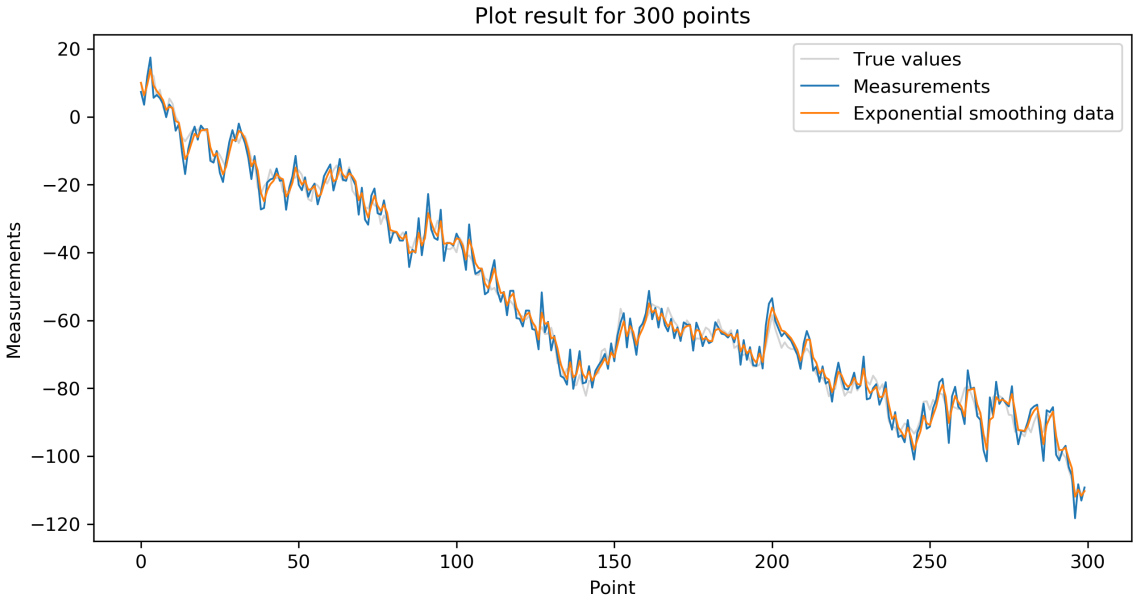
```
#4
#Expsmoothing() - exponential smoothing.
def Expsmoothing(alfa,z):
    array=np.zeros(len(z))
    array[0]=10
    for i in range(1,len(z)):
                array[i]=array[i-1]+alfa*(z[i]-array[i-1])
    return array

exp3000=Expsmoothing(alf,z3000)
figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot result for 3000 points')
plt.ylabel('Measurements')
plt.xlabel('Point')
plt.plot(x3000,label='True values',lw=1, c='lightgrey')
plt.plot(z3000,label='Measurements',lw=1)
plt.plot(exp3000,label='Exponential smoothing data',lw=1)
plt.legend()
plt.show()
```



In [9]:

```
exp300=Expsmoothing(alf,z300)
figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot result for 300 points')
plt.ylabel('Measurements')
plt.xlabel('Point')
plt.plot(x300,label='True values',c='lightgrey',lw=1)
plt.plot(z300,label='Measurements',lw=1)
plt.plot(exp300,label='Exponential smoothing data',lw=1)
plt.legend()
plt.show()
```



In [10]:

```
#Part 2
#1
sigmasTrue=[28,97]
w300= np.random.normal(0, sigmasTrue[0], 300)
x300=TrueTrajectory(300,w300)
```

In [11]:

```
#2
ny300= np.random.normal(0, sigmasTrue[1], 300)
z300=Measurements(x300,ny300)
```

In [12]:

```
#3
alf=Alfa(sigmasTrue)
print('Optimal smoothing coefficient=',alf)
```

```
0.08332447656499097
Optimal smoothing coefficient= 0.24998861233121078
```

In [13]:

```
#4
#FindWind()- determine the window size.
def FindWind(alfa):
    return round((2-alfa)/alfa)
wind=FindWind(alf)
print('Window size=',wind)
```

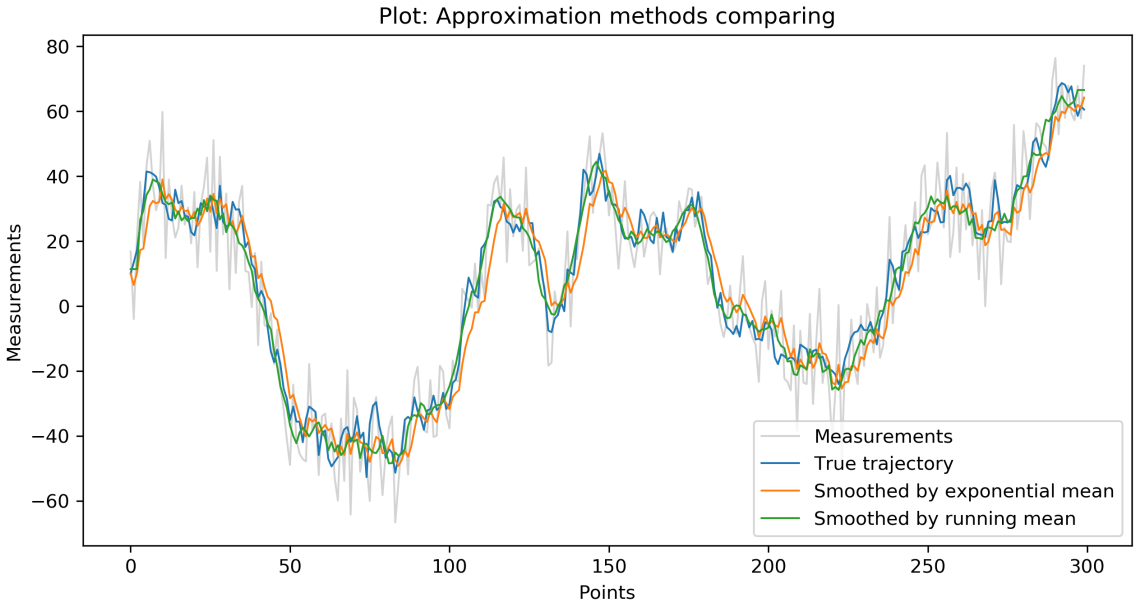
```
Window size= 7.0
```

In [14]:

```
#5
#RunningMean()- running mean smoothing.
def RunningMean(z,m):
    x=np.zeros(len(z))
    sumFirst=0
    sumLast=0
    step=int((m-1)/2)
    for i in range(0,step):
        sumFirst+=z[i]
        sumLast+=z[len(z)-i-1]
    for i in range(step,len(z)-step):
        for j in range(i-step,i+step+1):
            x[i]+=z[j]
        x[i]/=m
    for i in range(0,step):
        x[i]=sumFirst/step
        x[len(x)-i-1]=sumLast/step
    return x

r300=RunningMean(z300,wind)
exp300=Expsmoothing(alf,z300)

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Approximation methods comparing')
plt.ylabel('Measurements')
plt.xlabel('Points')
plt.plot(z300,label='Measurements',c='lightgrey',lw=1)
plt.plot(x300,label='True trajectory',lw=1)
plt.plot(exp300,label='Smoothed by exponential mean',lw=1)
plt.plot(r300,label='Smoothed by running mean',lw=1)
plt.legend()
plt.show()
```


Plot: Approximation methods comparing
Measurements
Points
80
60
40
20
0
−20
−40
−60
0
50
100
150
200
250
300
Measurements
True trajectory
Smoothed by exponential mean
Smoothed by running mean