```
#Assignment 4
#Determining and removing drawbacks of exponential and running mean.
Task 2
#I. Comparison of the traditional 13-month running mean with the
forward-backward exponential smoothing for approximation of 11-year
sunspot cycle
#II. 3d surface filtration using forward-backward smoothin
#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
from mpl_toolkits.mplot3d import Axes3D
```

```
#Part 1
## Comparison of the traditional 13-month running mean with the
forward-backward exponential smoothing for approximation of 11-year
sunspot cycle.
```

```
#Step 1 - Download monthly mean sunspot number dataset
df = pd.read_csv("D:\Skoltech\Term 1B Courses\Experimental Data
Processing\Assignment 4\data_group5.csv")
```

```
df
```

```
     year  month  sunspot number
0    1843      1            22.2
1    1843      2             5.9
2    1843      3            13.9
3    1843      4            15.8
4    1843      5            35.1
..    ...    ...             ...
433  1879      2             0.9
434  1879      3             0.0
435  1879      4            10.4
436  1879      5             4.0
437  1879      6             8.0

[438 rows x 3 columns]
```

```
#unite year and month columns to new one named data for comforttable
plotting
df['data'] = pd.to_datetime((df['year'].astype ( str ) +
df['month'].astype ( str )), format="%Y%m")
df
```

```
     year  month  sunspot number       data
0    1843      1            22.2 1843-01-01
1    1843      2             5.9 1843-02-01
2    1843      3            13.9 1843-03-01
3    1843      4            15.8 1843-04-01
4    1843      5            35.1 1843-05-01
..    ...    ...             ...        ...
```

```
433  1879      2             0.9 1879-02-01
434  1879      3             0.0 1879-03-01
435  1879      4            10.4 1879-04-01
436  1879      5             4.0 1879-05-01
437  1879      6             8.0 1879-06-01

[438 rows x 4 columns]
```

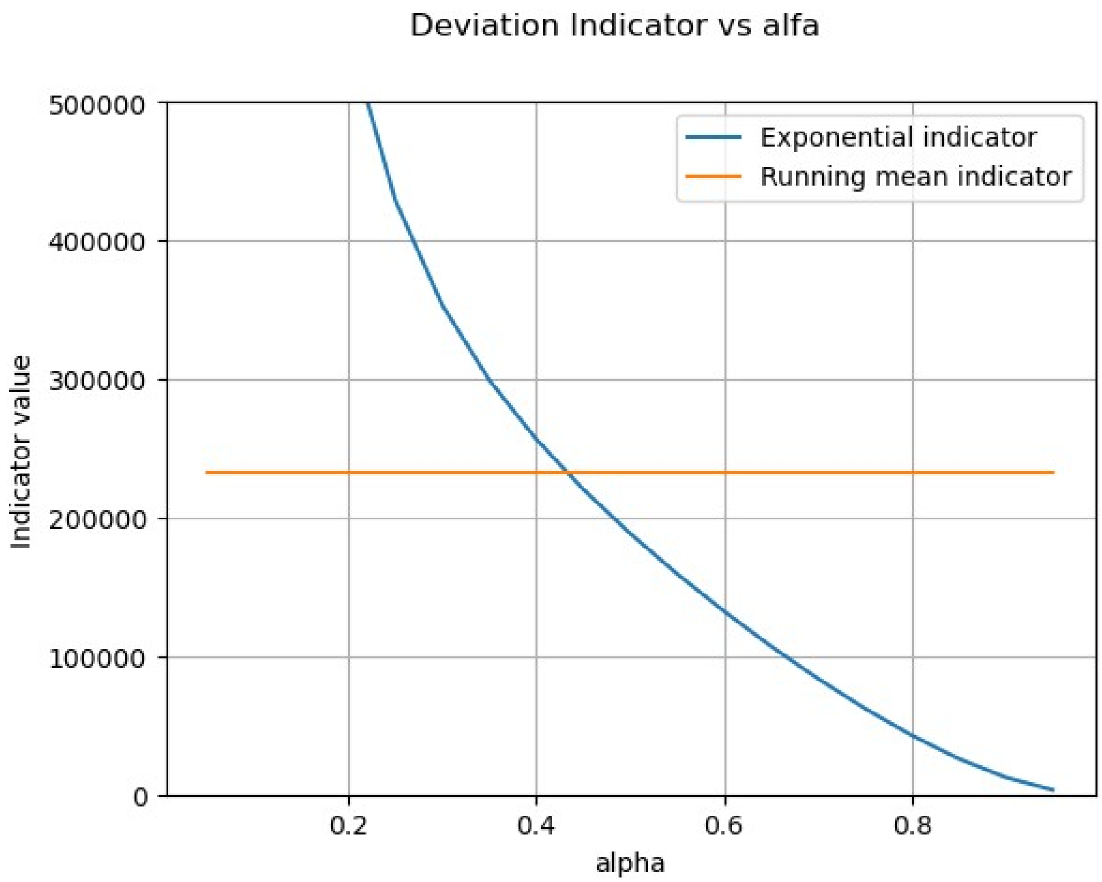
```
#Step 2 - make smoothing by 13-month Running mean for monthly
sunspot number
R = df['sunspot number'].values[:]
numrows = len(df['sunspot number'].values[:])
R1 = [[0] for _ in range(numrows)]
for i in range(0,numrows):
    if i in range(0,6):
        R1[i] = (R[i] + R[i+5] + R[i+4] + R[i+3] + R[i+2] +
R[i+1])/6
    if i in range(numrows-6,numrows):
        R1[i] = (R[i-5] + R[i-4] + R[i-3] + R[i-2] + R[i-1] +
R[i])/6
    if i in range(6,numrows-6):
        R1[i] = (R[i-6])/24 + (R[i-5] + R[i-4] + R[i-3] + R[i-2] +
R[i-1] + R[i] + R[i+5] + R[i+4] + R[i+3] + R[i+2] + R[i+1])/12 +
(R[i+6])/24

#np.shape(R1)
#print(R1)

#Step 3 - make forward-backward exponential smoothing

#we set functions for Forward exponential smoothing and Backward
exponential smoothing to have acsess to them in every future step

numrows = len(R)
#forward exponential smoothing
def Exp_smoothing(Z_f,n_f, alpha):
    Xsm_f = [0 for _ in range(n_f)]
    Xsm_f[0] = R[0]
    for i in range(1,n_f):
        Xsm_f[i] = Xsm_f[i-1] + alpha*(Z_f[i] - Xsm_f[i-1])
    return Xsm_f


#backward exponential smoothing
def Exp_back_smoothing(Xsm_f,n_f, alpha):
    Xsm_back = [0 for _ in range(n_f)]
    Xsm_back[numrows-1] = Xsm_f[numrows-1]
    for i in range(n_f-2,-1,-1):
        Xsm_back[i] = Xsm_back[i+1] + alfa*(Xsm_f[i] -
Xsm_back[i+1])
    return Xsm_back

#to make forward-backward exponential smoothing we need to set the
alfa, and we can do it based on  deviation and variability
```

```
indicators

#set the function for alfa
alfa_plot = [0 for _ in range(19)]
Xsm_array = [0 for _ in range(19)]
Xsm_array_f = [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_f[k] = Exp_smoothing(R,numrows,alfa)
    Xsm_array[k] = Exp_smoothing(Xsm_array_f[k],numrows,alfa)
    alfa_plot[k] = i/100
    k +=1

#Is there a smoothing constant alpha that provides better results
compared to 13-month running mean according to deviation and
variability indicators?

# set the functions for deviation and variability indicators for
exponential mean

#deviation indicator for Exponential mean
Id_em = [0 for _ in range(19)]
for i in range(19):
    for j in range(numrows):
        Id_em[i] += np.square(R[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(numrows-2):
        Iv_em[i] += (Xsm_array[i][j+2] - 2*Xsm_array[i][j+1] +
Xsm_array[i][j])**2

# deviation indicator Running Mean
Id_rm = np.sum(np.square(R-R1))

#variation indicators for Running Mean
Iv_rm = 0

for j in range(numrows-2):
    Iv_rm += (R1[j+2] - 2*R1[j+1] + R1[j])**2

print('deviation indicator Running Mean=',Id_rm)
print('variation indicators for Running Mean=',Iv_rm)
```

```
deviation indicator Running Mean= 232675.9597222222
variation indicators for Running Mean= 1607.834027777779
```

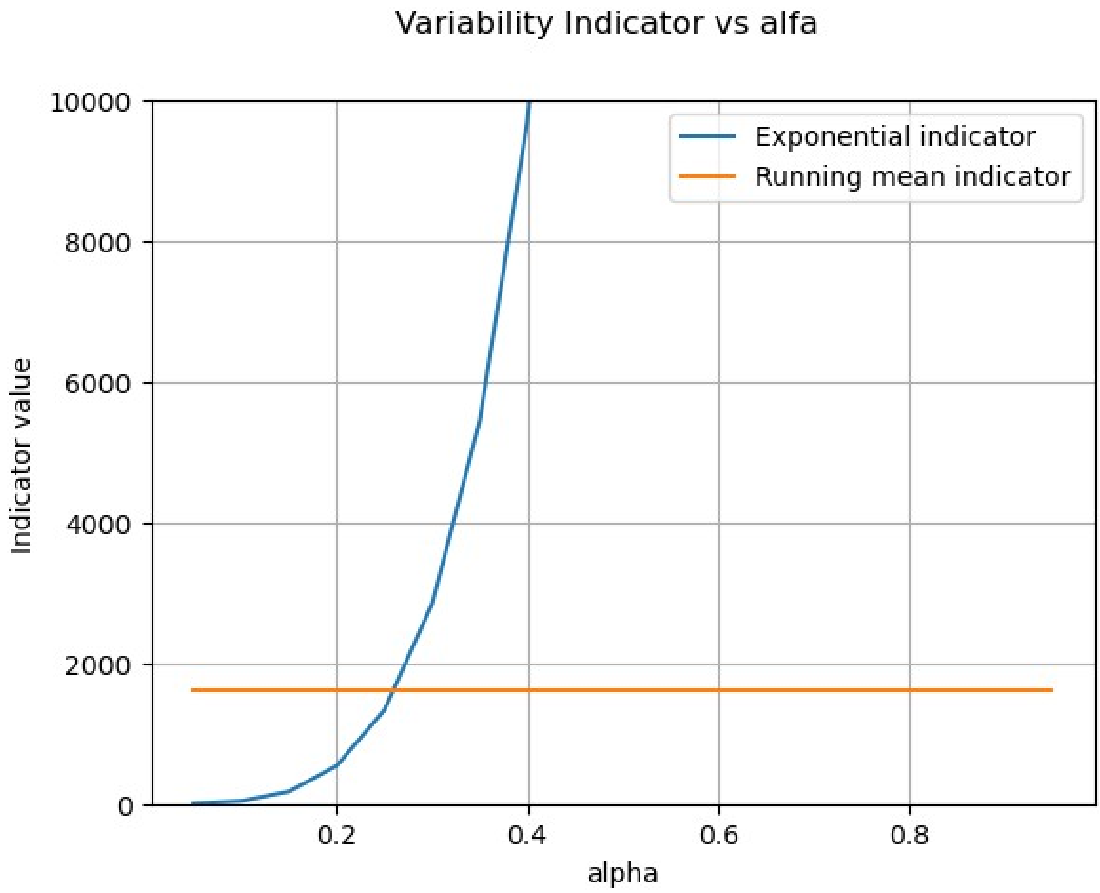
```
#plot the functions for to make conclusion abour choise of alfa
Id_rm_vector = [Id_rm for _ in range(19)]
y1 = Id_em
y2 = Id_rm_vector
x = alfa_plot
```

```python
plt.plot(x,y1,label = 'Exponential indicator')
plt.plot(x,y2,label = 'Running mean indicator')
plt.ylim(0,500000 )
plt.xlabel('alpha')
plt.ylabel('Indicator value')
plt.suptitle('Deviation Indicator vs alfa')
plt.legend()
plt.grid(True)
```



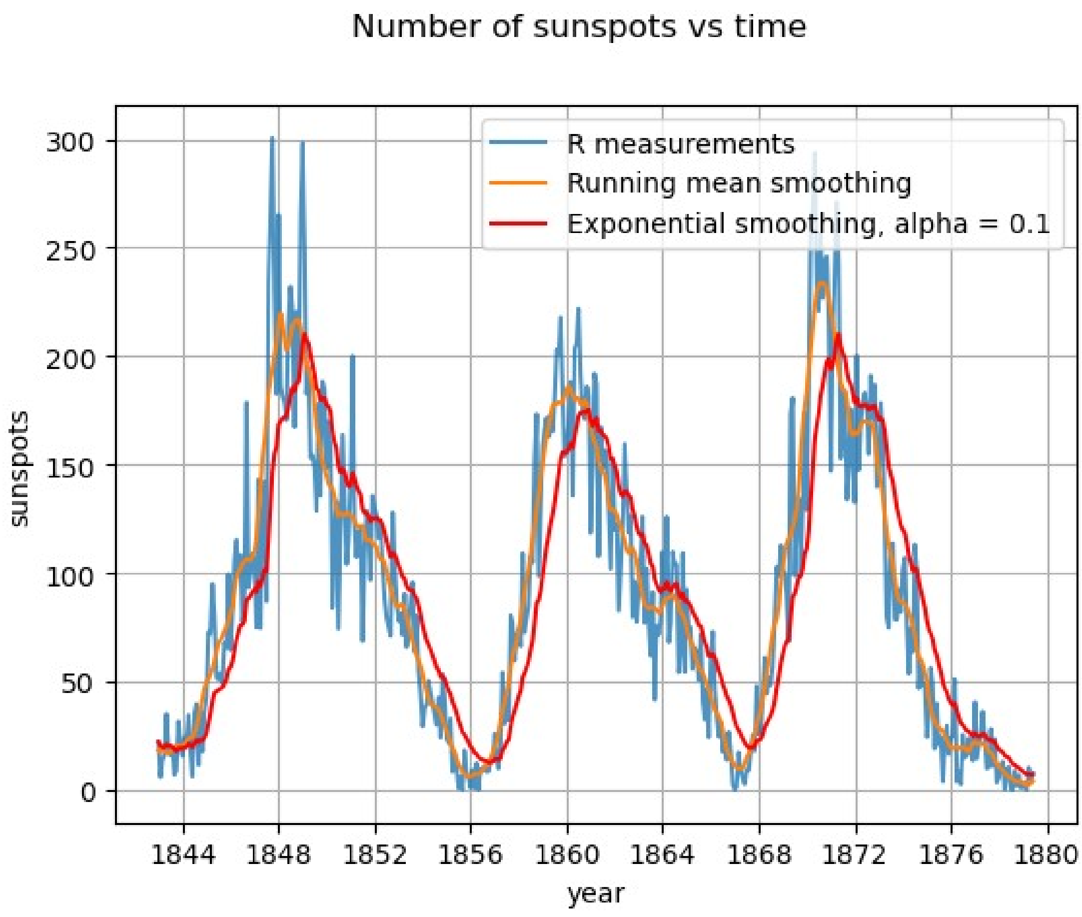
Conclusion: from the intersection on the plot we can consider that the level of filtration for exponential mean equal to level of filtration for running mean can be reached for alpha < 0.45

```python
#plot the functions for to make conclusion abour choise of alfa
Iv_rm_vector = [Iv_rm for _ in range(19)]
y1 = Iv_em
y2 = Iv_rm_vector
x = alfa_plot
plt.plot(x,y1,label = 'Exponential indicator')
plt.plot(x,y2,label = 'Running mean indicator')
plt.ylim(0,10000 )
plt.xlabel('alpha')
plt.ylabel('Indicator value')
plt.suptitle('Variability Indicator vs alfa')
plt.legend()
plt.grid(True)
```




```
#forward exponential smoothing with alpha1 = 0.1

alpha1 = 0.1

Xsm1 = Exp_smoothing(R,numrows,alpha1)

#backward exponential smoothing with alpha1 = 0.1

Xsm1_back = Exp_back_smoothing(Xsm1,numrows,alpha1)


#forward exponential smoothing with alfa2 = 0.2

alpha2 = 0.2

Xsm2 = Exp_smoothing(R,numrows,alpha2)

#backward exponential smoothing with alpha1 = 0.2

Xsm2_back = Exp_back_smoothing(Xsm2,numrows,alpha2)

#plot
y0 = R
y1 = R1
y2 = Xsm1_back
#y3 = Xsm1
```

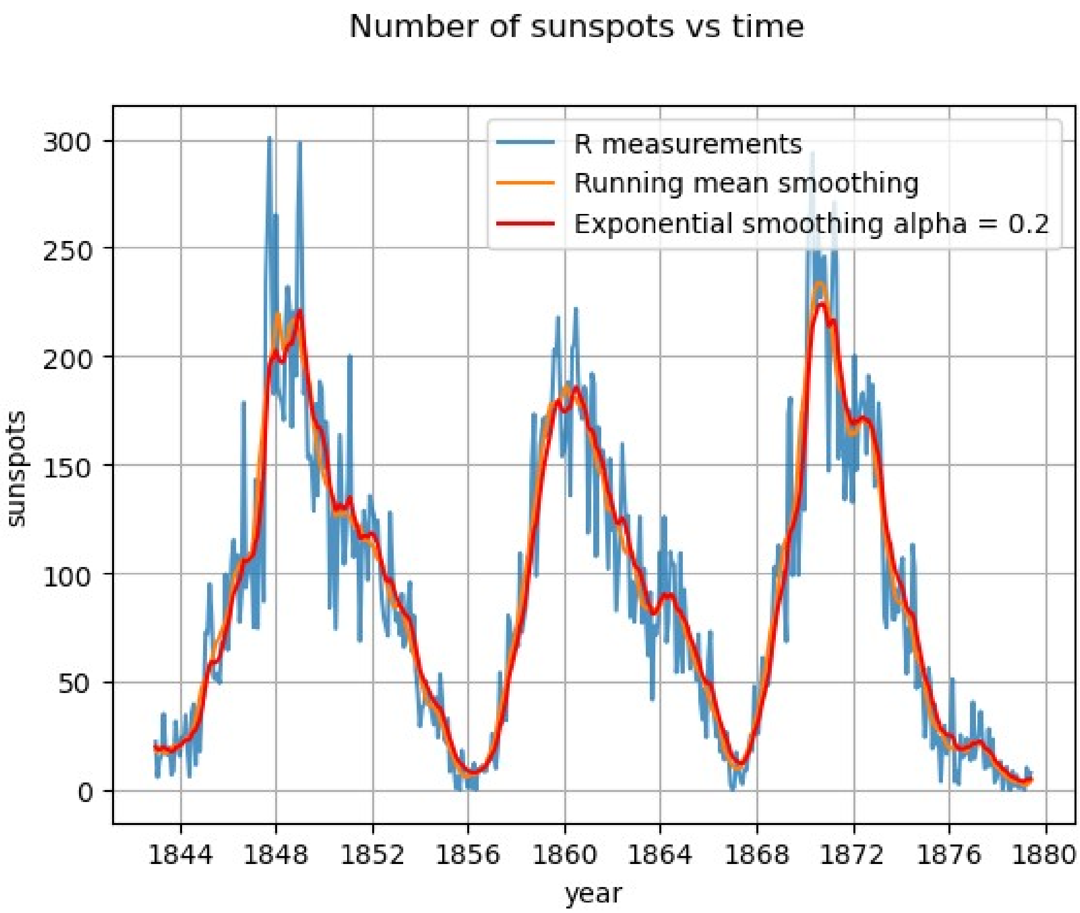
```
x = df['data'].values[:]
plt.plot(x,y0,label = 'R measurements', alpha=0.8)
plt.plot(x,y1,label = 'Running mean smoothing')
plt.plot(x,y2,label = 'Exponential smoothing, alpha = 0.1',
color='r')
#plt.plot(x,y3,label = 'Forward Exponential smoothing, alpha = 0.1',
color='b')
plt.xlabel('year')
plt.ylabel('sunspots')
plt.suptitle('Number of sunspots vs time')
plt.legend()
plt.grid(True)
```



Conclusion: alpha1 provides high level of smoothing but causes the shift in our

```
#plot
y0 = R
y1 = R1
y2 = Xsm2_back
x = df['data'].values[:]
plt.plot(x,y0,label = 'R measurements', alpha = 0.8)
plt.plot(x,y1,label = 'Running mean smoothing')
plt.plot(x,y2,label = 'Exponential smoothing alpha = 0.2',
color='r')
plt.xlabel('year')
plt.ylabel('sunspots')
```

```
plt.suptitle('Number of sunspots vs time')
plt.legend()
plt.grid(True)
```



Conclusion: alpha2= 0.2 is optimal choice for smoothing

Part 1 Conclusion: after analizing and plotting the data, we can consider that exponential mean smoothing with alpha=0.2 can provide results as good as 13-mounth running mean provides and maybe even better for more precise ajustments of alpha

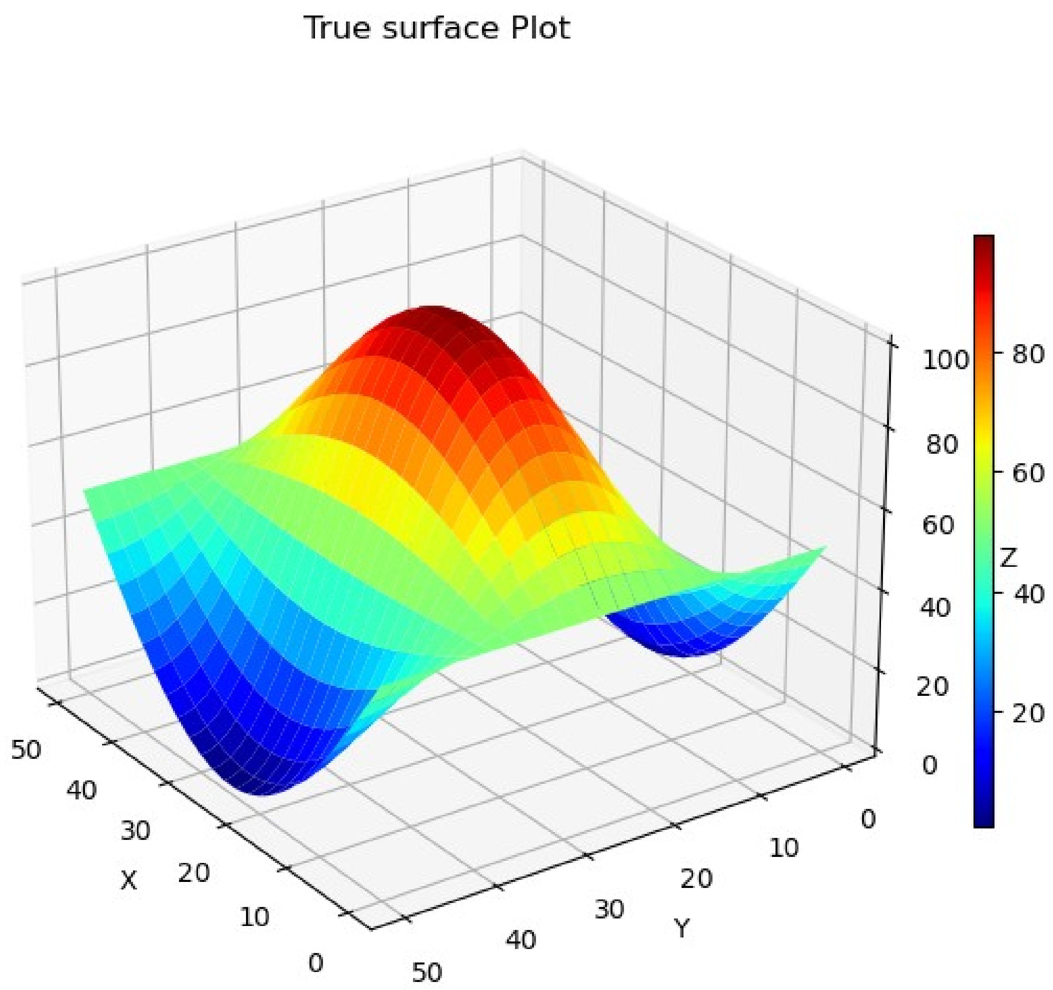
```
#Part 2

#1. noisy_surface.txt(available mesurements to work with)
#true_surface.txt(true surface to compare the estimation results)

#2. Plot noisy and true surface for visualization purposes.To plot
3d surfaces in matlab, there is a command “mesh”.
#You can assign a colormap for the plot, i.e., “colormap jet”,
“set(gca,'colormap','jet')”
#The plot should be accompanied with the “colorbar".

#Step 1 - download surface data

noisy = np.loadtxt("noisy_surface.txt")
n = len(noisy)
```

```
true = np.loadtxt("true_surface.txt")
x = np.loadtxt("true_surface.txt")
y = np.loadtxt("true_surface.txt")
```

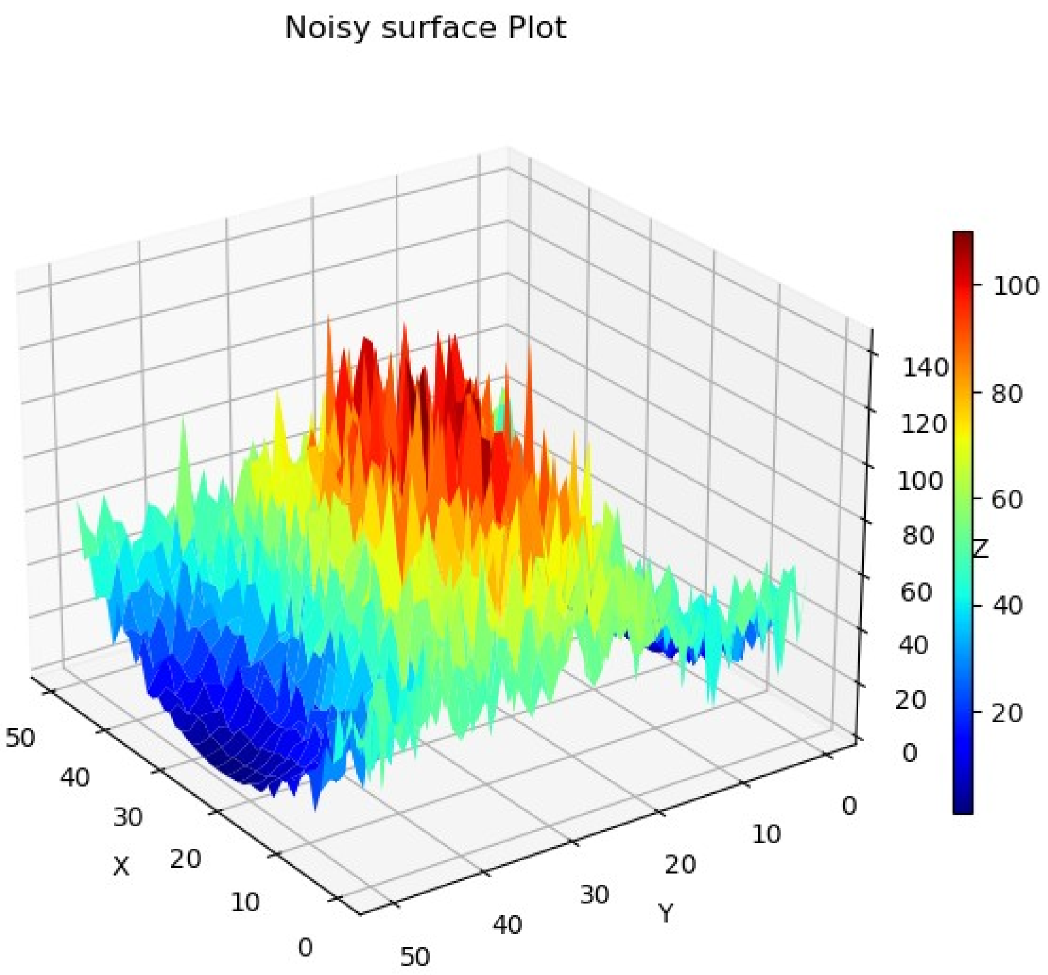
```
noisy
```

```
array([[61.26  , 33.4491, 60.0272, ..., 54.7422, 55.5336, 64.1533],
       [38.4392, 56.0842, 24.2585, ..., 28.1391, 50.7476, 45.8256],
       [59.4336, 55.5091, 52.3045, ..., 66.1099, 47.8726, 43.1854],
       ...,
       [60.9133, 67.1038, 49.7542, ..., 54.8217, 46.8984, 47.3424],
       [57.8051, 44.5932, 66.0002, ..., 34.3572, 39.3617, 44.418 ],
       [56.4043, 54.3355, 58.5813, ..., 49.8374, 55.3364, 65.9077]])
```

```
#Step 2 - Plot noisy and true surface for visualization purposes
for i in range(n):
    for j in range(n):
        x[i][j] = j
        y[i][j] = j
x = x.T

#plot true surface
fig = plt.figure(figsize=(8,8))
ax3d = fig.add_subplot(111, projection='3d')

#plot = ax3d.plot_wireframe(x,y,true, cmap = 'jet')
plot = ax3d.plot_surface(x,y,true, cmap = 'jet')
ax3d.set_title('True surface Plot')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('  Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание шкалы градиента
ax3d.view_init(25, 145)

plt.show()
```




```
#Conclusion: nicely looking surface of true values

#plot noisy surface
fig = plt.figure(figsize=(8,8))
ax3d = plt.axes(projection="3d")

my_cmap = plt.get_cmap('jet')

#plot = ax3d.plot_wireframe(x,y,noisy, cmap = my_cmap)
plot = ax3d.plot_surface(x,y,noisy, cmap = 'jet')
ax3d.set_title('Noisy surface Plot')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание
шкалы градиента
ax3d.view_init(25, 145)
plt.show()
```



Conclusion: high level of noise destroys understanding of data nature

```
#Step 3 - Determine the variance of deviation of noisy surface from
the true one.

var_noisy = np.reshape(noisy, (1,np.product(noisy.shape)))
var_true = np.reshape(true, (1,np.product(true.shape)))
s1 = 0
for i in range(n**2):
    s1 += (var_noisy[0][i] - var_true[0][i] )**2
s1 = s1/n**2
print('Variance of deviation of noisy surface from the true one
=',s1)
```

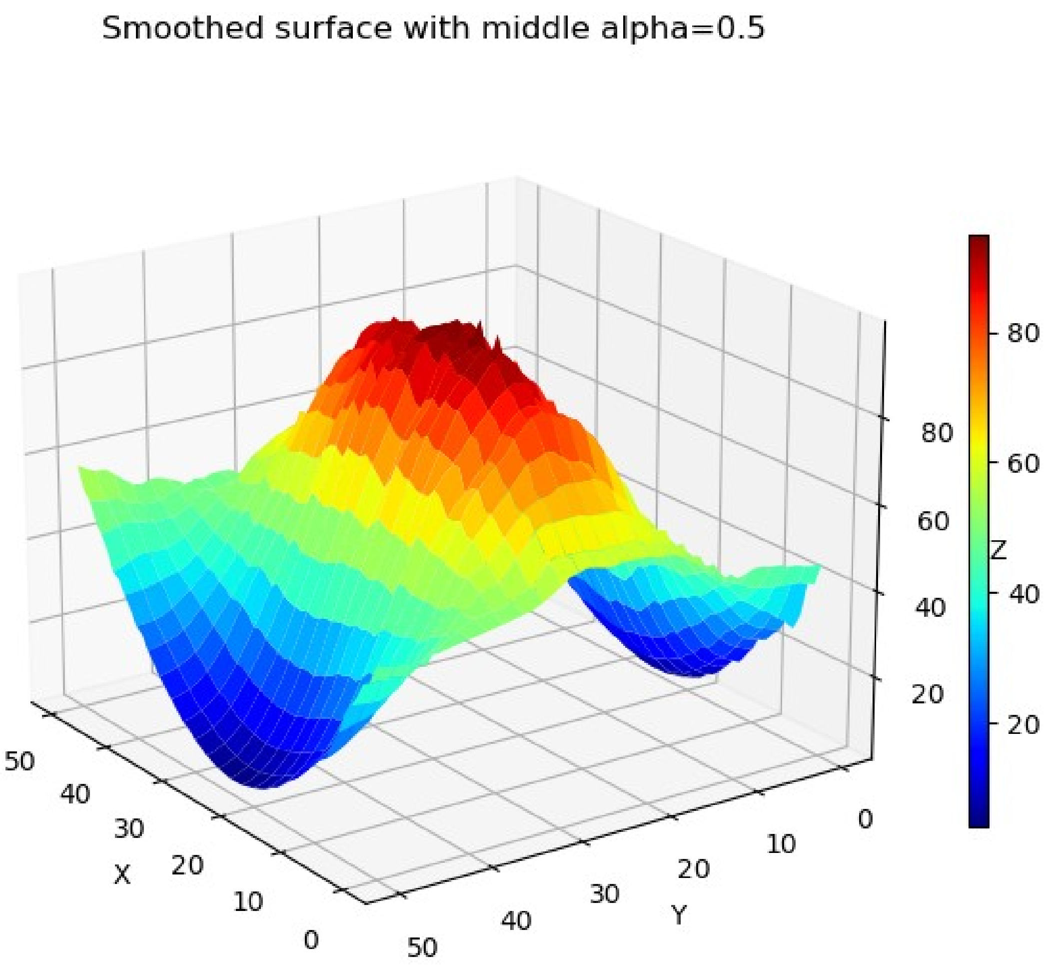
```
Variance of deviation of noisy surface from the true one =
122.82584136052297
```

By following the procedure we have determined the variation of deviation of noisy surface from the true one which is 122.82584136052297

```
#Step 4 - Apply forward-backward exponential smoothings
#The smoothing constant can be A = 0.335
#There should be 4 steps in forward-backward smoothing of a surface.
```

```
#Step 1
# Forward exponential smoothing of rows
alfa = 0.335
def F_smoothing(a,n):
    Xsm = [[0]*n for _ in range(n)]
    for i in range(n):
        Xsm[i][0] = a[i][0]
    for i in range(n):
        for j in range(1,n):
            Xsm[i][j] = Xsm[i][j-1] + alfa*(a[i][j] - Xsm[i][j-1])
    return Xsm

#Step 2
#backward exponential smoothing
def B_smoothing(a,n):
    Xsm_back = [[0]*n for _ in range(n)]
    for i in range(n):
        Xsm_back[i][n-1] = a[i][n-1]
    for i in range(n): # 50 -> 0
        for j in range(n-2,-1,-1):  #49 -> 0
            Xsm_back[i][j] = Xsm_back[i][j+1] + alfa*(a[i][j] -
Xsm_back[i][j+1])
    return Xsm_back

Xsm_F_row = F_smoothing(noisy, n)
Xsm_B_row = B_smoothing(Xsm_F_row, n)

#fig = plt.figure(figsize=(8,8))
#ax3d = plt.axes(projection="3d")

#my_cmap = plt.get_cmap('jet')

#plot = ax3d.plot_surface(x,y,np.array(Xsm_B_row), cmap = 'jet')
#ax3d.set_title(' surface Plot')
#ax3d.set_xlabel('X')
#ax3d.set_ylabel('Y')
#ax3d.set_zlabel('Z')
#fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание
шкалы градиента
#ax3d.view_init(30, 145)
#plt.show()

#Step 3
# Forward exponential smoothing of columns
def F_column_smoothing(a,n):
    Xsm_c = [[0]*n for _ in range(n)]
    for i in range(n):
        Xsm_c[n-1][i] = a[n-1][i]
    for j in range(n):
        for i in range(n-2,-1,-1):
            Xsm_c[i][j] = Xsm_c[i+1][j] + alfa*(a[i][j] - Xsm_c[i+1]
[j])
    return Xsm_c
```

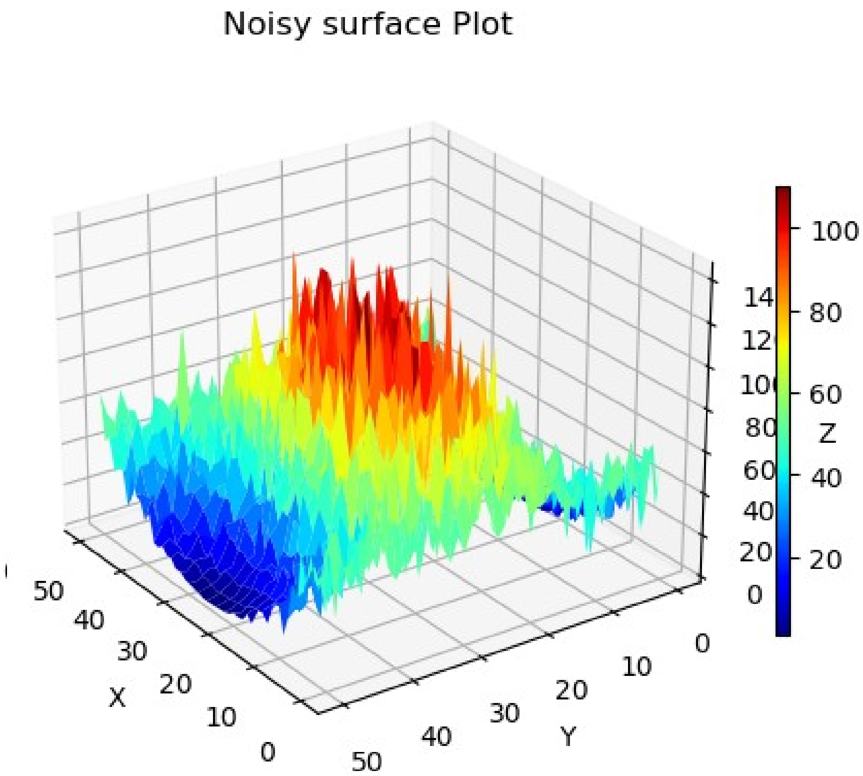
```
#Step 4
#backward exponential smoothing of columns
def B_column_smoothing(Xsm,n):
    Xsm_back_c = [[0]*n for _ in range(n)]
    for i in range(n):
        Xsm_back_c[0][i] = Xsm[0][i]
    for j in range(n):
        for i in range(1,n):
            Xsm_back_c[i][j] = Xsm_back_c[i-1][j] + alfa*(Xsm[i][j]
- Xsm_back_c[i-1][j])
    return Xsm_back_c

Xsm_F_column = F_column_smoothing(Xsm_B_row, n)
Xsm_B_column = B_column_smoothing(Xsm_F_column, n)

fig = plt.figure(figsize=(8,8))
ax3d = plt.axes(projection="3d")

my_cmap = plt.get_cmap('jet')

#plot = ax3d.plot_wireframe(x,y,np.array(Xsm_F_column), cmap =
'jet')
plot = ax3d.plot_surface(x,y,np.array(Xsm_F_column), cmap = 'jet')
ax3d.set_title('Smoothed surface with middle alpha=0.5')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание
шкалы градиента
ax3d.view_init(20, 145)
plt.show()
```

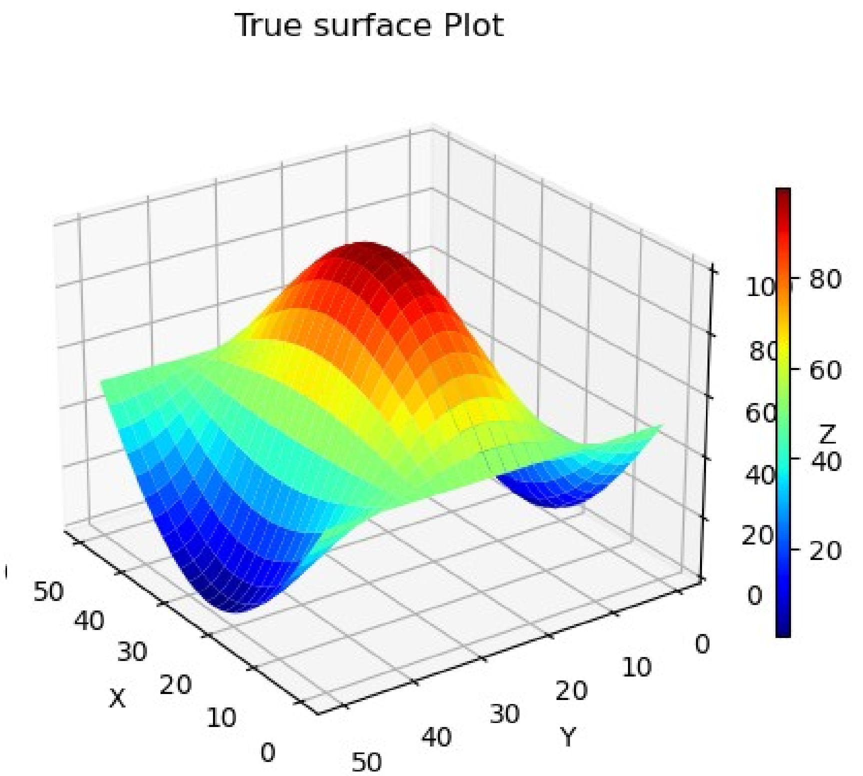


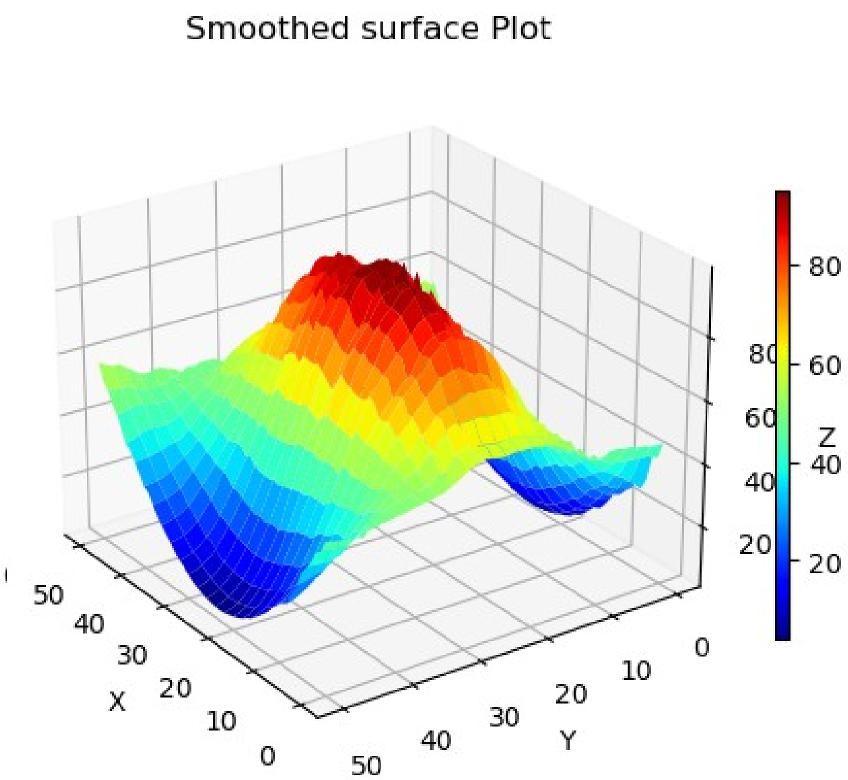
Conclusion: very nice transformation of row and noisy surface to a pleasant looking one, but now we need to prove effectivness of transformation numericaly by calculating variance

```
fig = plt.figure(figsize=(6,6))
fig.add_subplot(1, 2, 1, projection='3d')
ax3d = plt.axes(projection="3d")

my_cmap = plt.get_cmap('jet')


plot = ax3d.plot_surface(x,y,noisy, cmap = 'jet')
ax3d.set_title('Noisy surface Plot')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание шкалы градиента
ax3d.view_init(25, 145)



#-----------------------------------------------------
```

```
fig = plt.figure(figsize=(6,6))
fig.add_subplot(1, 2, 1, projection='3d')
ax3d = plt.axes(projection="3d")

#plot = ax3d.plot_wireframe(x,y,true, cmap = 'jet')
plot = ax3d.plot_surface(x,y,true, cmap = 'jet')
ax3d.set_title('True surface Plot')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30) # создание
шкалы градиента
ax3d.view_init(25, 145)



#------------------------------------------------------



fig = plt.figure(figsize=(6,6))
fig.add_subplot(1, 2, 1, projection='3d')
ax3d = plt.axes(projection="3d")

my_cmap = plt.get_cmap('jet')

#plot = ax3d.plot_wireframe(x,y,np.array(Xsm_F_column), cmap =
'jet')
plot = ax3d.plot_surface(x,y,np.array(Xsm_F_column), cmap = 'jet')
ax3d.set_title('Smoothed surface Plot')
ax3d.set_xlabel('X')
ax3d.set_ylabel('Y')
ax3d.set_zlabel('Z')
fig.colorbar(plot, ax = ax3d, shrink = 0.5, aspect = 30)
ax3d.view_init(25, 145)
plt.show()

plt.show()
```


Noisy surface Plot
X
Y
Z


True surface Plot
X
Y
Z



Conclusion: this series of plots nicely depicts how noisy surface differs from true one, and how we change it and make it close to true with smoothing

```
#Step 6 - determine the variance of deviation of smoothed surface
from the true one.
# Compare the variance with that from item 3.

#calculate new variance of deviation of smoothed surface from the
true one
index = 0
var_smoothed = [0 for _ in range(n**2)]
for i in range(n):
    for j in range(n):
        var_smoothed[index] = Xsm_B_column[i][j]
        index += 1
var_true = np.reshape(true, (1,np.product(true.shape)))
s2 = 0
for i in range(n**2):
    s2 += (var_smoothed[i] - var_true[0][i] )**2
s2 = s2/n**2
print('Variance of deviation of smooth surface from the true one
=',s2)
```

```
Variance of deviation of smooth surface from the true one =
7.920949927804128
```

```
#after our manipulations the variance value decreased significally
#compare variance of noisy and smooth surface
dif = s1/s2
print('Comparison of variance:\n',dif)
```

```
Comparison of variance:
 15.50645345318742
```

Conclusion: after our manipulations the variance value decreased significally - in 15.5 times in compaeison with noisy/true variance.It deffinately means that the method we applyied was powerful and could recover the noisy dataset very close to the true one

#Step 7 - trying different values of alpha and explore how it affects on estimation results

Observation results are following:

#small values of alfa (alpha=0.05) give us high level of smoothing, however it causes distortion of surface image.png

#middle values of alfa (alpha=0.5) give some level of smoothing, but in this case its not enough to read surface clearly image-2.png

#big values of alfa (alpha=0.8) provede almost zero filtration image-3.png

#Step 8 - Personal Learning log

Yaroslav: from this assignment I lerned the true magic of recovering data almost from scratch. I was really amazed how such simple method as exponential smoothing could rebuild so noisy data set in almost identical to the true one.

Lisa: I learned how to create 3d plots and work with 2d arrays processing them with familiar method of forward-backward exponential smoothing. First part of assignment was useful to train exponential smoothing on 1d arrays again.

Selamawit Asfaw: Such a way of recovering data coluld be useful in my future learning activities, so I made a note about it. It was pleasant to see how methods we learned for one dimension datasets can be applyied on more complicated sources and deepicted so nicely on plots, made by Lisa.