Laboratory work 4 - Determining and removing drawbacks of exponential and running mean. Task 2

Nikolay Zherdev, Mikhail Kulbeda

Skoltech, 10.10.2018

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.

This laboratory work consists of two parts:

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 smoothing.

Part 1

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

Format:

Column 1: year

Column 2: month

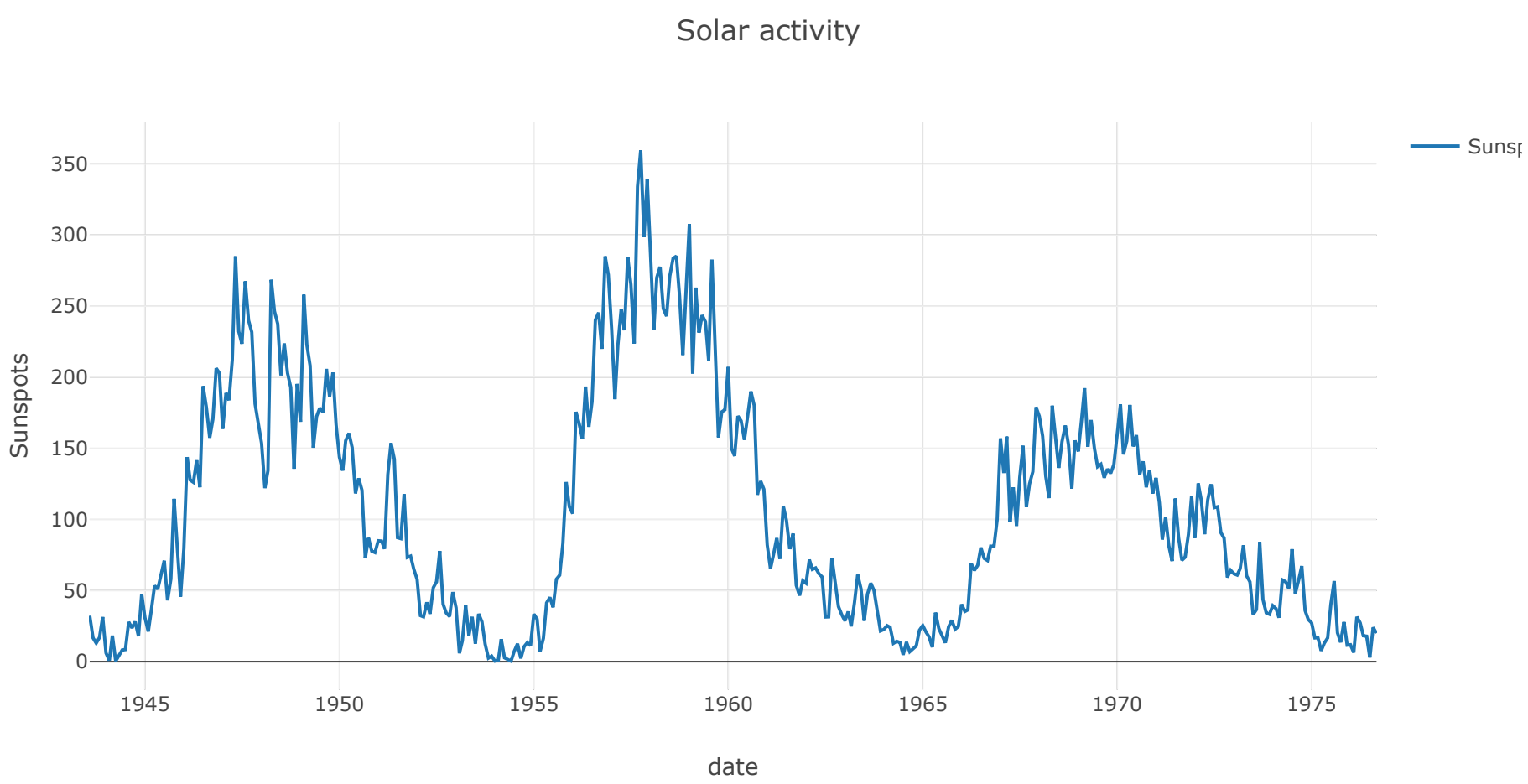
Column 3: monthly mean sunspot number

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go
%matplotlib inline
import matplotlib.pyplot as plt

from matplotlib.pyplot import figure


init_notebook_mode(connected=True)
```

In [3]:
```
data_path = '/Users/nickzherdev/Desktop/Experimental_Data_Processing/lab4/data_group2.txt'
df = pd.read_csv(data_path, delim_whitespace=True, header=None)
```

In [4]:
```
n = len(df[0])
day = [1 for i in range(n)] # this is just to make function pd.to_datetime work properly
                            # (it requirs day, month and year minimum)
pd_day = pd.Series(day)

dataframe = pd.DataFrame({
    'Year': df[0],
    'Month': df[1],
    'Sunspot': df[2],
    'Day': pd_day
})

# combining datetime to make smooth plot
date = pd.to_datetime(dataframe[['Year', 'Month', 'Day']])

#df.columns = ['Year', 'Month', 'Sunspot']
```

In [5]:
```
data = [
    go.Scatter(
        x = date,
        y = df[2],
        name='Sunspot'
    )
]

layout= go.Layout(
    title= 'Solar activity',
    xaxis= dict(
        title= 'date',
    ),
    yaxis=dict(
        title= 'Sunspots',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plo

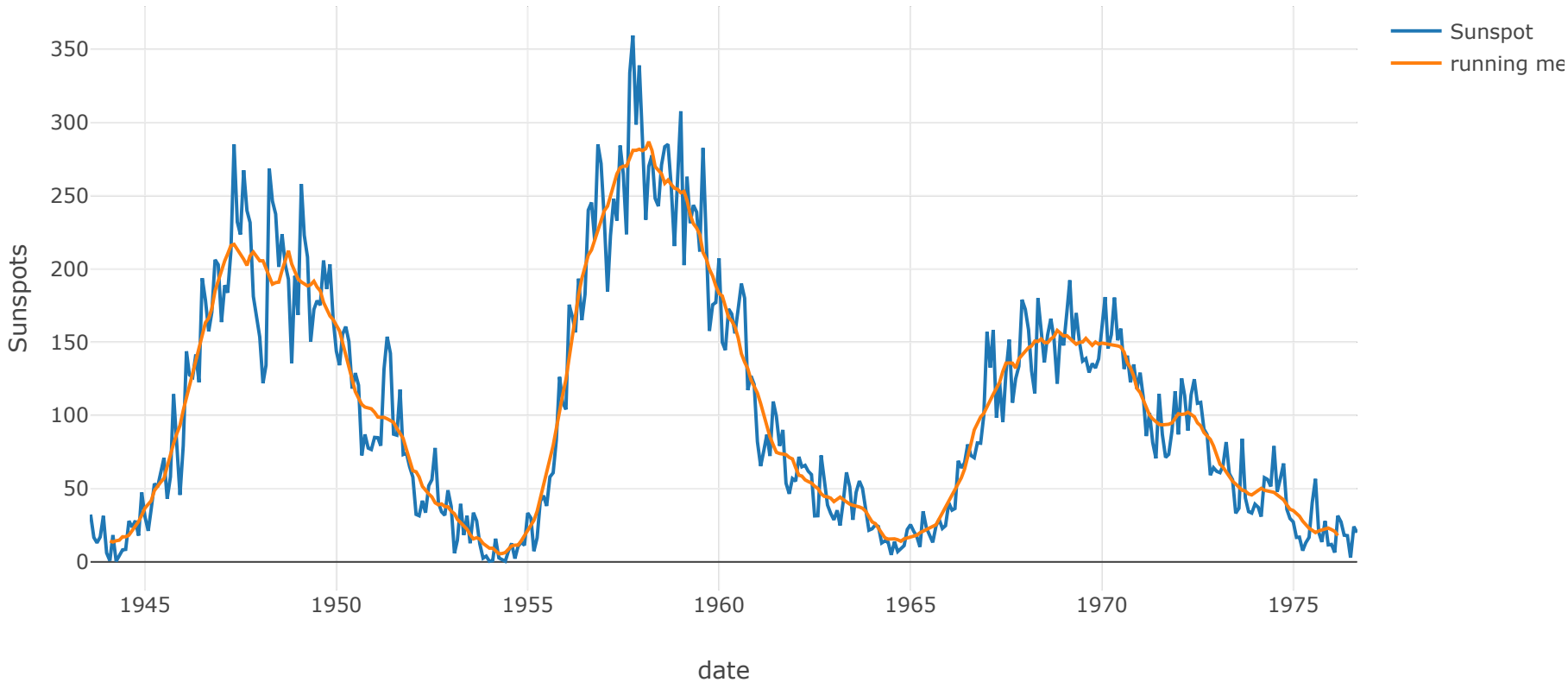
2. Make smoothing of monthly mean data by 13-month running mean.
13-month running mean $\bar{R}$

$$\bar{R} = \frac{1}{24}R_{i-6} + \frac{1}{12}(R_{i-5} + R_{i-4} + \cdots + R_{i-1} + R_i + R_{i+1} + \cdots + R_{i+5}) + \frac{1}{24}R_{i+6}$$

I couldn't manage to apply different weights to rolling function.

In [6]:
```
running_mean_sunspots = pd.Series(df[2]).rolling(window = 13, center = True).mean()
```

In [7]:
```
data = [
    go.Scatter(
        x = date,
        y = df[2],
        name='Sunspot'
    ),
    go.Scatter(
        x = date,
        y = running_mean_sunspots,
        name='running mean'
    ),
]

layout= go.Layout(
    title= 'Solar activity',
    xaxis= dict(
        title= 'date',
    ),
    yaxis=dict(
        title= 'Sunspots',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



3. Make forward-backward exponential smoothing of monthly mean sunspot number. Is there a smoothing constant α that provides better results compared to 13-month running mean according to deviation and variability indicators?

In [8]:
```
alpha = 0.25
```

In [9]:
```
def exp_smooth(z, alpha):
    X = np.zeros(z.shape)
    X[0] = z[0]
    for i in range(1, z.shape[0]):
        X[i] = X[i-1] + alpha*(z[i] - X[i-1])
    return X
```

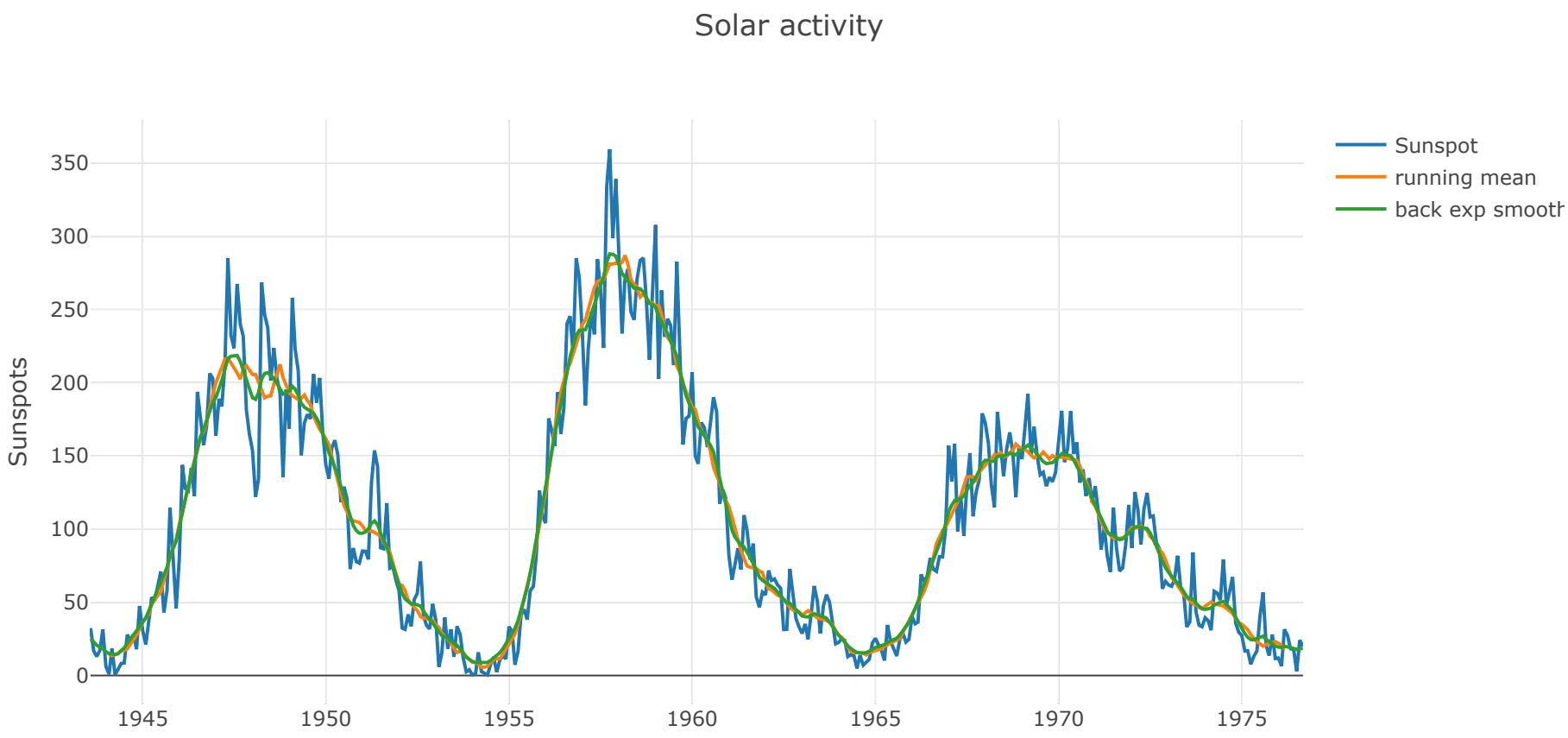
In [10]:
```
exp_smoothed = exp_smooth(df[2], alpha)
```

In [11]:
```
def back_exp_smooth(exp_smoothed, alpha):
    B = np.zeros(exp_smoothed.shape[0])
    B[-1] = exp_smoothed[-1]
    for i in reversed(range(exp_smoothed.shape[0]-1)):
        B[i] = B[i+1] + alpha*(exp_smoothed[i] - B[i+1])
    return B
```

In [12]:
```
def fb_smoothing(msr, alpha):
    exp_smoothed = exp_smooth(msr, alpha)
    return exp_smooth(exp_smoothed[::-1], alpha)[::-1]
```

In [13]:
```
back_exp_smoothed = back_exp_smooth(exp_smoothed, alpha)
```

In [14]:
```
data = [
    go.Scatter(
        x = date,
        y = df[2],
        name='Sunspot'
    ),
    go.Scatter(
        x = date,
        y = running_mean_sunspots,
        name='running mean'
    ),
#     go.Scatter(
#         x = date,
#         y = exp_smoothed,
#         name='exp smoothed'
#     ),
    go.Scatter(
        x = date,
        y = back_exp_smoothed,
        name='back exp smoothed'
    ),
]

layout= go.Layout(
    title= 'Solar activity',
    xaxis= dict(
        title= 'date',
    ),
    yaxis=dict(
        title= 'Sunspots',
    ),
    showlegend= True
)
fig= go.Figure(data=data, layout=layout)
iplot(fig)
```



In [15]:
```
def exp_smooth_variability(sequence):
    return np.sum((sequence[2:] - 2*sequence[1:-1] + sequence[:-2])**2)
```

In [16]:
```
def running_mean_variability(sequence, M):
    X_j2 = sequence[int(M/2)+2:-int(M/2)]
    X_j1 = sequence[int(M/2)+1:-(int(M/2)+1)]
    X_j = sequence[int(M/2):-(int(M/2)+2)]
    RM_variability = np.sum((np.add(np.subtract(X_j2, 2*X_j1), X_j))**2)
    return RM_variability
```

In [17]:
```
def deviation_indicator(measurments, estimations):
    error = measurments - estimations
    return np.mean(error**2)
```

In [18]:
```
running_mean_dev_ind = deviation_indicator(df[2], running_mean_sunspots)
back_exp_smooth_dev_ind = deviation_indicator(df[2], back_exp_smoothed)
```

In [19]:
```
running_mean_dev_ind, back_exp_smooth_dev_ind
```

Out[19]: (512.1549184474354, 387.12564371933894)

In [20]:
```
RM_variability = running_mean_variability(running_mean_sunspots, 13)
BES_variability = exp_smooth_variability(back_exp_smoothed)
```

In [21]:
```
RM_variability, BES_variability
```

Out[21]: (3085.5690532544404, 1069.5911783559081)

Yes, smoothing constant α=0.25 provides better results compared to 13-month running mean according to deviation and variability indicators.

Part 2

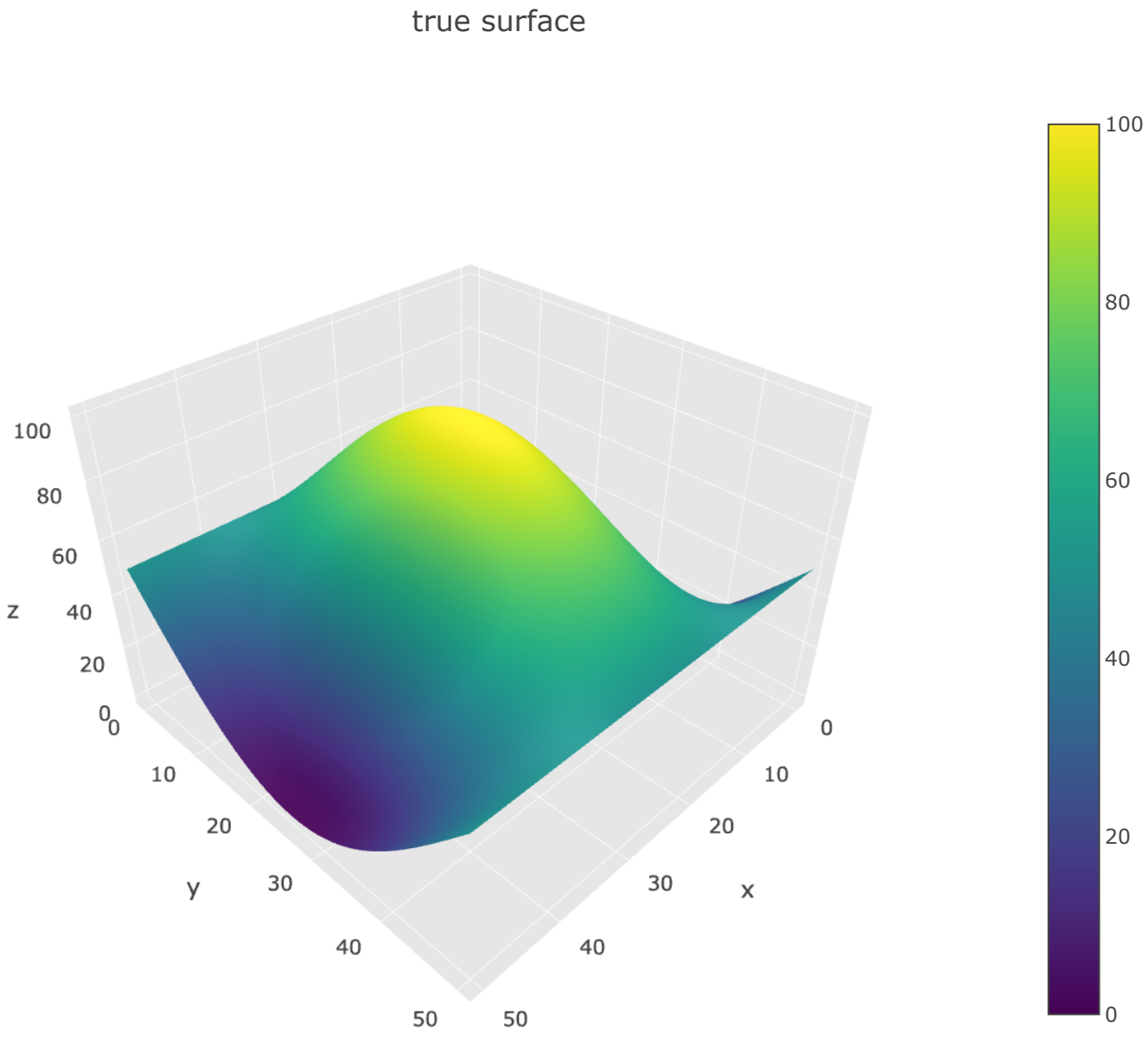
3d surface filtration using forward-backward smoothing.

The goal of this task is to reconstruct the 3D surface (A) on the basis noisy measurements of the surface (B) in conditions of uncertainty.

Plot noisy and true surface for visualization purposes.

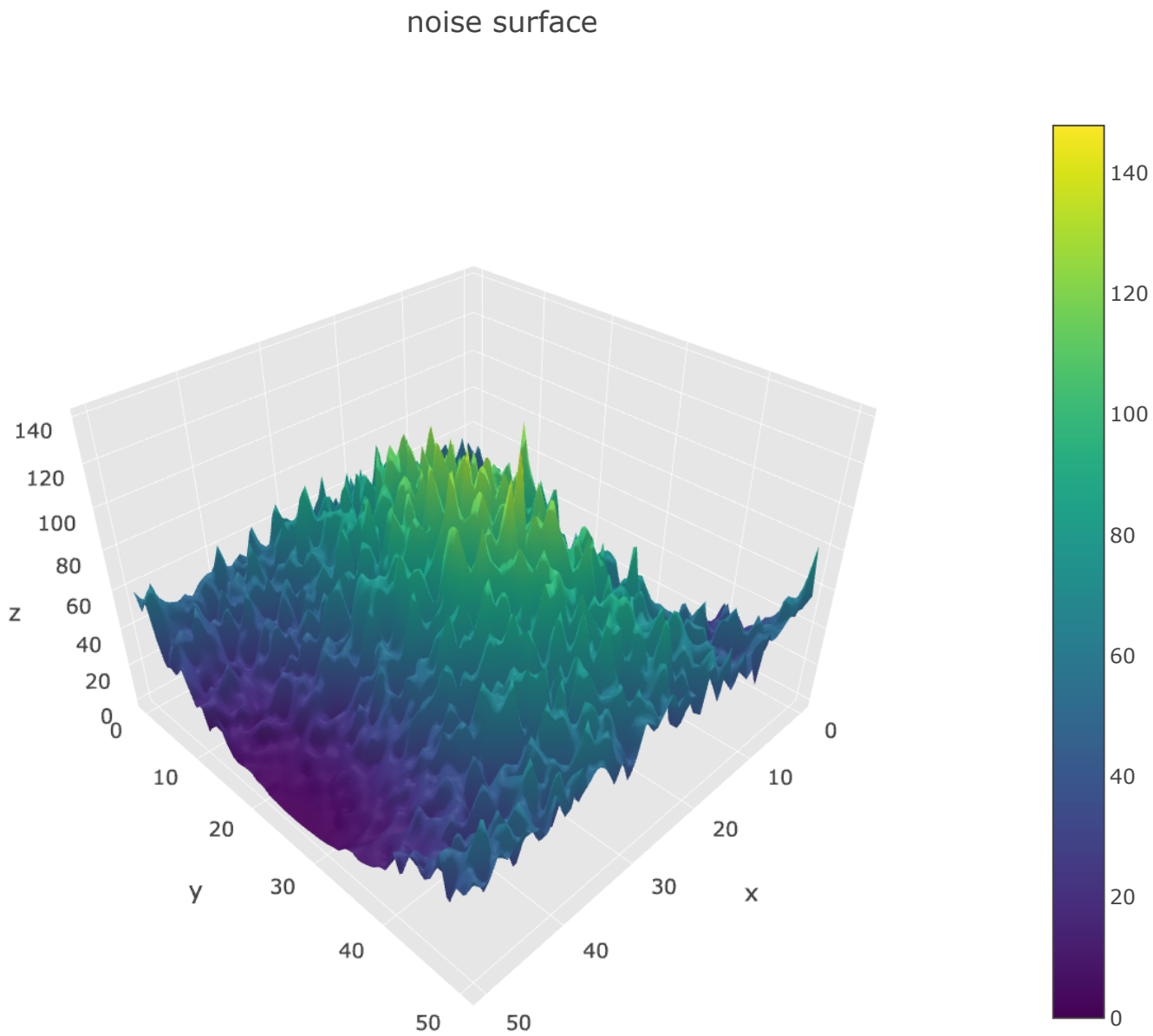
In [22]:
```
true_path = '/Users/nickzherdev/Desktop/Experimental_Data_Processing/lab4/true_surface.txt'
noise_path = '/Users/nickzherdev/Desktop/Experimental_Data_Processing/lab4/noisy_surface.txt'

df_true = pd.read_csv(true_path, delim_whitespace=True, header=None)
df_noise = pd.read_csv(noise_path, delim_whitespace=True, header=None)
```

In [23]:
```
data = [
    go.Surface(
        z=df_true.values,
        colorscale='Viridis'
    )


]
layout = go.Layout(
    title='true surface',
    width=800,
    height=700,
    autosize=False,
    margin=dict(
        l=65,
        r=50,
        b=65,
        t=90
    ),
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plot.ly »

In [24]:
```
data = [
    go.Surface(
        z=df_noise.values,
        colorscale='Viridis'
    )


]
layout = go.Layout(
    title='noise surface',
    width=800,
    height=700,
    autosize=False,
    margin=dict(
        l=65,
        r=50,
        b=65,
        t=90
    ),
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plot.ly »

3. Determine the variance of deviation of noisy surface from the true one.
Hint: You may reshape the matrix (difference between the noisy and true surface) into one array ("reshape command") and then determine the variance of obtained array.

In [25]:
```
np_df_noise = np.array(df_noise).flatten()
np_df_true = np.array(df_true).flatten()
```

In [26]:
```
error = np_df_noise - np_df_true
var = np.var(error, ddof = 1)
```

In [27]:
```
var
```

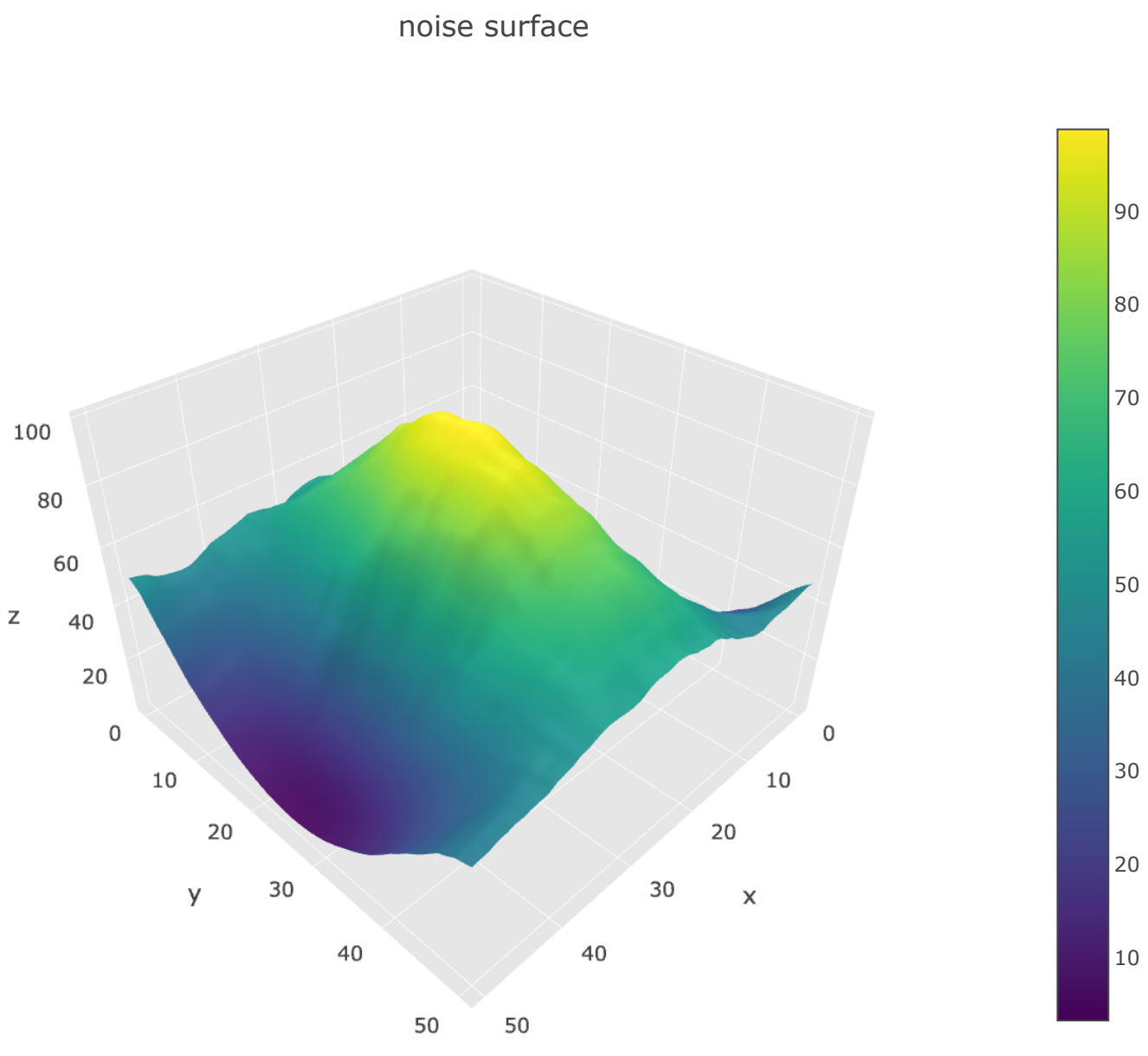
Out[27]:
```
122.05477678749108
```

4. Apply forward-backward exponential smoothing to filter noisy surface measurements.
The smoothing constant can be $\alpha = 0.335$
Hint: There should be 4 steps in forward-backward smoothing of a surface.
Step 1: Forward exponential smoothing of rows (from left to right).
Step 2: Backward exponential smoothing of results obtained at step 1 (from right to left).
Step 3: Forward exponential smoothing of results obtained at step 2 along the columns (from bottom to top).
Step 4: Backward exponential smoothing of results obtained at step 3 along the columns (from top to bottom).

In [28]:
```
noizy_data = np.loadtxt('/Users/nickzherdev/Desktop/Experimental_Data_Processing/lab4/noisy_surface.txt')
```

In [29]:
```
def smooth_matrix(_matrix, alpha, smoother=fb_smoothing):

    matrix = _matrix.copy()

    for i in range(len(matrix[1,:])):
        matrix[:,i] = smoother(matrix[:,i], alpha)

    for i in range(len(matrix[:,1])):
        matrix[i,:] = smoother(matrix[i,:], alpha)

    return matrix
```

In [30]:
```
smoothed_fb = smooth_matrix(noizy_data, 0.335)

data = [
    go.Surface(
        z = smoothed_fb,
        colorscale='Viridis'
    )


]
layout = go.Layout(
    title='noise surface',
    width=800,
    height=700,
    autosize=False,
    margin=dict(
        l=65,
        r=50,
        b=65,
        t=90
    ),
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



Export to plot.ly »

5. Compare visually the obtained estimation results and true surface.

6. Determine the variance of deviation of smoothed surface from the true one. Compare the variance with that from item 3.

7. Try greater and smaller values of smoothing coefficient α and explain the affect on estimation results.

8. General conclusions.

Visually - forward-backward exponential smoothing did a very good job.

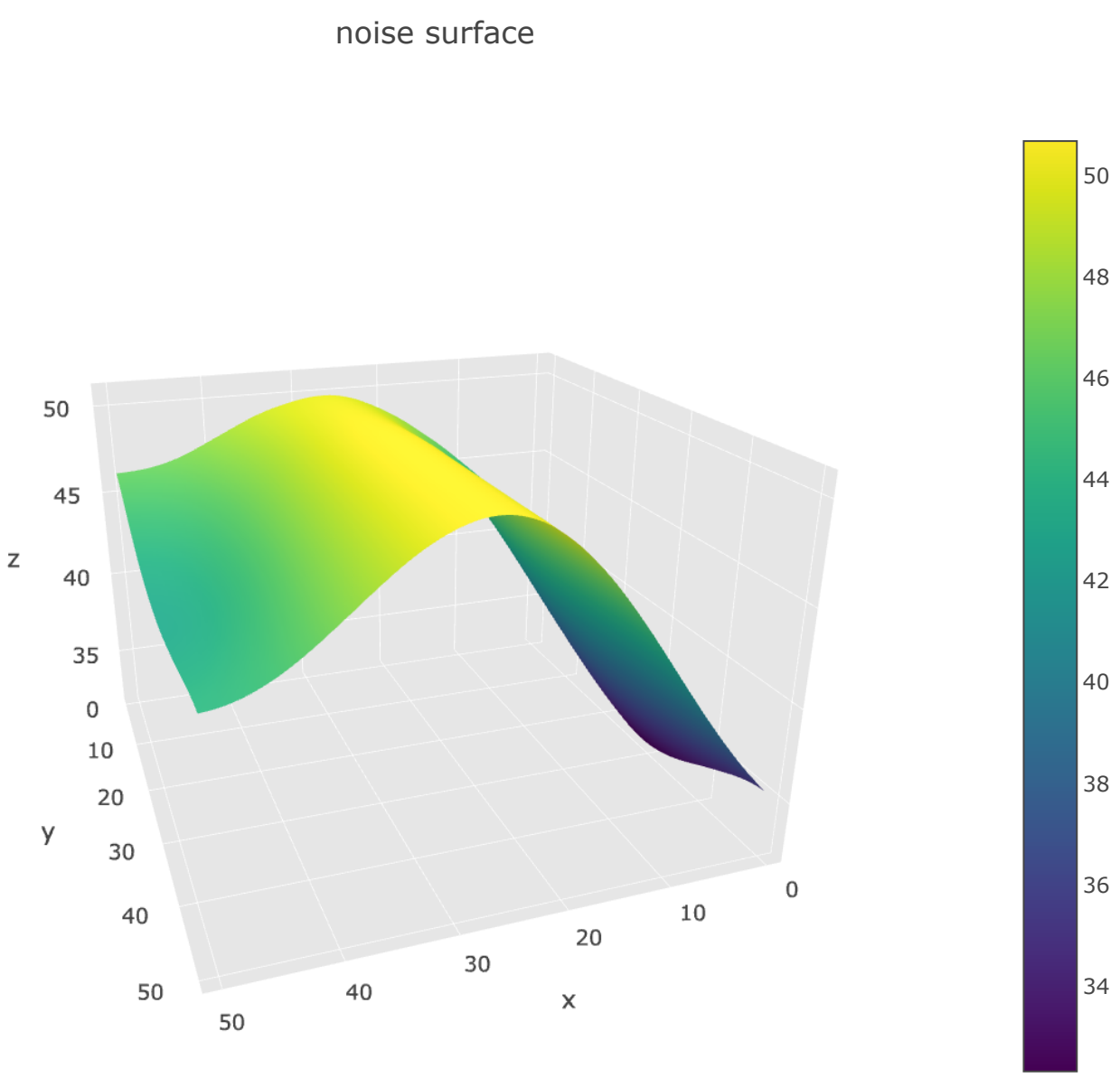
In [33]:
```
estimated = np.array(smoothed_fb).flatten()
np_df_true = np.array(df_true).flatten()
error = estimated - np_df_true
var = np.var(error, ddof = 1)
var
```

Out[33]: 6.7947501069405565

Variance of error between smoothed surface and the true one is 6.79, which is much smaller than in item 3 (122.05).

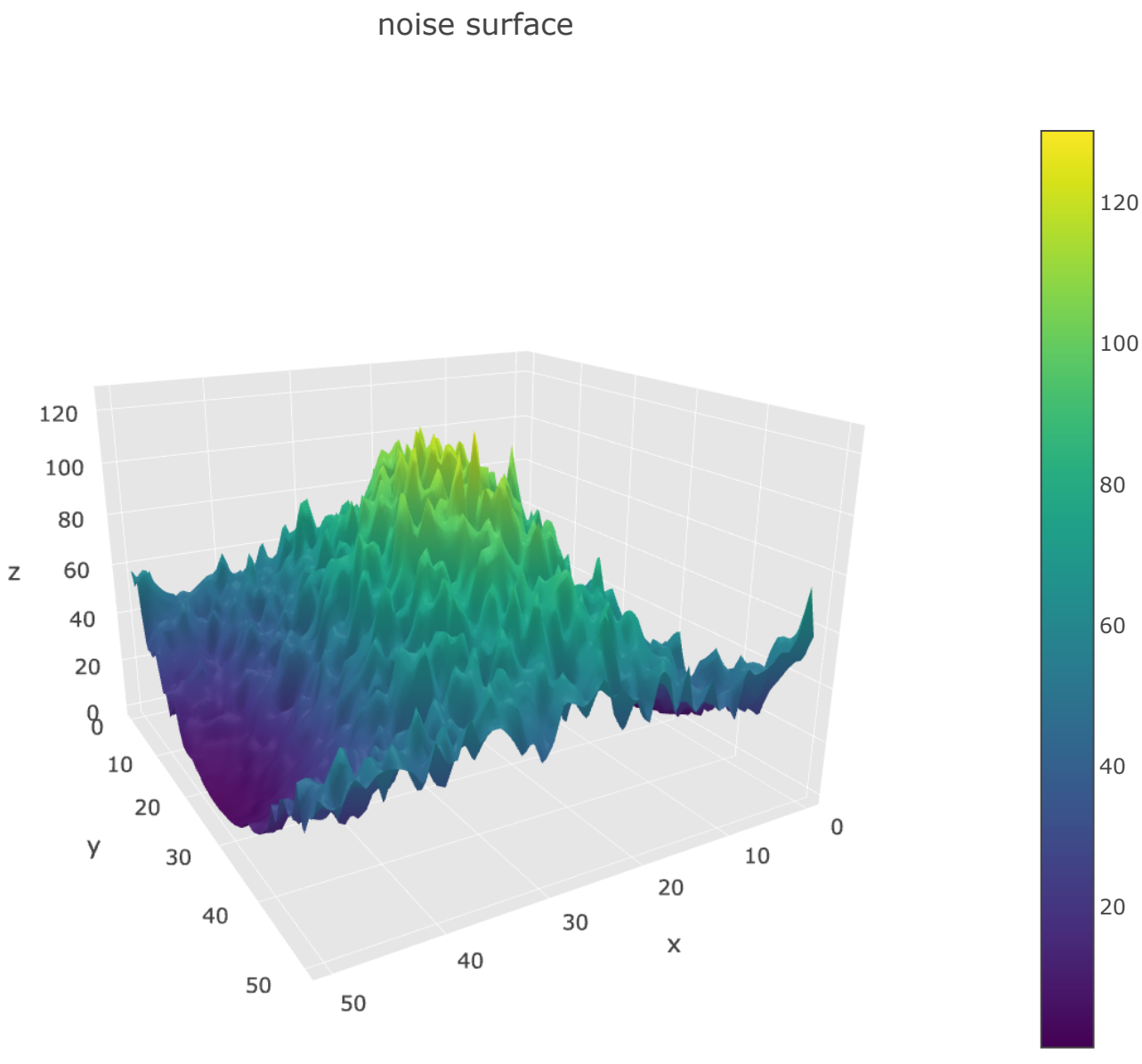
Now we will compare smoothing effects of small alpha and big alpha.

In [36]:
```
smoothed_fb_005 = smooth_matrix(noizy_data, 0.05)

data = [
    go.Surface(
        z = smoothed_fb_005,
        colorscale='Viridis'
    ),

]
layout = go.Layout(
    title='noise surface',
    width=800,
    height=700,
    autosize=False,
    margin=dict(
        l=65,
        r=50,
        b=65,
        t=90
    ),
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



As we can see from graph, smoothing with alpha = 0.05 produces very "conservative" surface that is not responsible for new measurments.

In [38]:
```
smoothed_fb_09 = smooth_matrix(noizy_data, 0.9)

data = [
    go.Surface(
        z = smoothed_fb_09,
        colorscale='Viridis'
    )

]
layout = go.Layout(
    title='noise surface',
    width=800,
    height=700,
    autosize=False,
    margin=dict(
        l=65,
        r=50,
        b=65,
        t=90
    ),
    scene=dict(
        xaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        yaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        zaxis=dict(
            gridcolor='rgb(255, 255, 255)',
            zerolinecolor='rgb(255, 255, 255)',
            showbackground=True,
            backgroundcolor='rgb(230, 230,230)'
        ),
        aspectratio = dict( x=1, y=1, z=0.7 ),
        aspectmode = 'manual'
    )
)
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```



On the contrary, big alpha = 0.9 is very responsible for new measurements and doesn't follow the "history" of estimates, so the produced surface is very

noisy.

General conclusions

Main drawback of forward exponential mean method, the shift, was eliminated in backward exponential mean method. The backward method worked better for processing sunspot number data, but not with all smoothing constants. Optimal smoothing constant α should be determined for the method to perform better. Deviation and variability indicators can be used as criteria for a good evaluation method. Also, for finding optimal smoothing constant. Backward exponential mean method can be applied to smooth surface measurements.