Ural Federal University
named after the first President of Russia B.N.Yeltsin
Institute of radioelectronics and information technologies

# SINGULAR SPECTRUM ANALYSIS AND DECOMPOSITION

## Laboratory Task no. 5

Yekaterinburg
2019

Ural Federal University
Institute of radioelectronics and information technologies

# Contents

## 1. Introduction

The purpose of this laboratory task is an implementation of the Singular Spectrum Analysis (SSA) algorithm, and also its approbation on model time series (TS) with the purpose to learn and seize skills at the choice of its parameters, grouping indices and reconstructions of the diagrams helping to carry out the analysis technique of the TS by the SSA method. The SSA method is an adaptive method, and therefore, it is very effective for the analysis and decomposition of a set of different time series including non-stationary.

## 2. Laboratory Task

The final result of laboratory task is presented as the report in form of *Jupyter*-notebook (or something similar), with all the needed commands for a complete Run of cells. Now, let's begin with time series data analysis:

1) Import into your code the following libraries:

```
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
import h5py
%matplotlib inline
```

2) The Singular Spectrum Analysis method consists of two steps – **decomposition** and **grouping**. Correspondingly, we will create two functions in Python for these steps.
3) Let's define the first function for decomposition:

```
def SSA_modes(F, L):
```

which takes two input parameters – the data *F* and the length of window *L*.

4) Inside function let's initialize the matrix $\boldsymbol{X}$ with dimensions $L$ x $K$.

```
N = len(F)
K = N - L + 1
X = np.empty((L, K))
```

5) Now, on your own, fill the elements of this matrix $\boldsymbol{X}$ with points from data $F = f(t) = \{f(t_0), ..., f(t_{N-1})\}$ by embedding it with window $L$:

$$X = (x_{ij})_{i,j=1}^{L,K} = \begin{pmatrix} f_0 & f_1 & f_2 & \cdots & f_{K-1} \\ f_1 & f_2 & f_3 & \cdots & f_K \\ f_2 & f_3 & f_4 & \cdots & f_{K+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{L-1} & f_L & f_{L+1} & \cdots & f_{N-1} \end{pmatrix}$$

6) The second step of embedding procedure is to create a new matrix $S = XX^T$ from this matrix $\boldsymbol{X}$:

```
S = np.dot(X, X.T)
```

7) Then this matrix is decomposed through Singular Value Decomposition (SVD) into its eigen-values and sub-matrices:

```
U, A, _ = np.linalg.svd(S)
```

where $\boldsymbol{U}$ – matrix of eigen-vectors, $\boldsymbol{A}$ – an array of ordered eigen-values $(\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_L \geq 0)$. The third output of function is ignored and is not needed to us.

8) We will also require matrix of trajectory vectors $V = X^T U$. Estimate it on your own correspondingly.

9) Overall, this function **SSA_modes(F, L)** must return 3 outputs: array of eigen-values $\boldsymbol{A}$, matrix of eigen-vectors $\boldsymbol{U}$ and matrix of trajectory vectors $\boldsymbol{V}$.

10) Let's test the performance and correctness of this function through simple example:

**ts = np.array([3, 2, 1, 2, 3, 2, 1, 2, 3, 2, 1, 2, 3]) #** test data

**A, U, V = SSA_modes(ts, 3) #** decomposition by SSA with length = 3

**print(A) #** eigen-values

**print(U) #** eigen-vectors

**print(V) #** trajectory vectors

11) Correct function will return the following results:

For ***A***: `[129.66842566  12.          3.33157434]`

For ***U***:
```
[[-5.78869570e-01  7.07106781e-01  4.06091149e-01]
 [-5.74299610e-01  4.14039445e-16 -8.18645196e-01]
 [-5.78869570e-01 -7.07106781e-01  4.06091149e-01]]
```

For ***V***:
```
[[-3.46407750e+00  1.41421356e+00 -1.29257973e-02]
 [-2.88977789e+00  0.00000000e+00  8.05719399e-01]
 [-3.46407750e+00 -1.41421356e+00 -1.29257973e-02]
 [-4.03837711e+00  8.88178420e-16 -8.31570994e-01]
 [-3.46407750e+00  1.41421356e+00 -1.29257973e-02]
 [-2.88977789e+00  0.00000000e+00  8.05719399e-01]
 [-3.46407750e+00 -1.41421356e+00 -1.29257973e-02]
 [-4.03837711e+00  8.88178420e-16 -8.31570994e-01]
 [-3.46407750e+00  1.41421356e+00 -1.29257973e-02]
 [-2.88977789e+00  0.00000000e+00  8.05719399e-01]
 [-3.46407750e+00 -1.41421356e+00 -1.29257973e-02]]
```

12) Next, the function for **grouping** of decomposed data is required. Let's name this function **SSA_group**. This function will have the following input parameters: array of eigen-values ***A***, matrix of eigen-vectors ***U***, matrix of trajectory vectors ***V***, length of initial data ***N*** and an array of grouping indices ***I***. The output is an array of recreated data for some component of initial time series.

13) This function will, obviously, when called, receive the following already found parameters ***A, U, V, N=len(F)***. The only new and unknown parameter is the array ***I***. Additionally; the window length ***L*** can be estimated through the length of array of eigen-values: **L = len(A)**. Also, the variable $K = N - L + 1$ is required.

14) Let the parameter ***I*** be represented in the form of an array on indices/numbers of components that we want to group together. This form simplifies the next step for us:

```
V = V.transpose()
Z = np.dot(U[:, I], V[I, :]) # equal statement Z = X_{I_1} + ... + X_{I_m}
```

15) Next is the diagonal averaging of this ***Z***. Let's reconstruct the component $G = g_0, ..., g_{N-1}$ with the length of initial data. First, provide a memory for this data: **G = np.zeros(N)**. Also, these two variables $L^* = \min(L, K)$, $K^* = \max(L, K)$ are required.

16) Now create a code on Python for this formula (diagonal averaging):

$$g_k = \begin{cases} \dfrac{1}{k+1}\sum_{m=0}^{k} Z^*_{m,k-m} & 0 \le k < L^* - 1, \\ \dfrac{1}{L^*}\sum_{m=0}^{L^*-1} Z^*_{m,k-m} & L^* - 1 \le k < K^*, \\ \dfrac{1}{N-k}\sum_{m=k-K^*+1}^{N-K^*} Z^*_{m,k-m} & K^* \le k < N+1. \end{cases}$$

17) At last, we have two functions for the SSA algorithm – to decompose data **SSA_modes**, and to reconstruct data **SSA_group**. Let's test the grouping with the following test, the data and decomposition is provided from pt. 11. It is pretty simple, let's group all of the components (all 3) and we should definitely receive our initial array of data:

```
ts1 = SSA_group(A, U, V, len(ts), [0, 1, 2])
print(ts1)
```

```
[3. 2. 1. 2. 3. 2. 1. 2. 3. 2. 1. 2. 3.]
```

18) If the previous step is correct, then try to create the three independent SSA components of this data: component with index [0] is received as **SSA_group(A, U, V, len(ts), [0])**, component with index [1], component with index [2], in the same way; and also their pair combinations (grouping indices [0, 1], [0, 2], [1, 2]). Plot all of these components on different figures against initial data.

19) Note the main characteristics of those components for different numbers. First, 0-component **contains the mean value** (expectation value) of data and is pretty close to a form of trend. Thus, 1-component and 2-component have mean values close to zero. Second, 1-component and 2-component have the same period/frequency. That's because in SSA method every **period** data will mainly consist of **pair of components**. Third, amplitude of 1-component is higher than 2-component, because array of eigen-values and corresponding components is **ordered in descending order**, thus, with increasing number of component its power or energy part of initial data is also decreasing.

20) Also note, that with higher value of parameter window ***L*** we will have more components, meaning the previous mentioned characteristics would be split among many components. But the overall regularities stay the same.

21) Now let's use the prepared method of SSA to a model data, for research and learning purposes.

22) First, create the model time series with 2 periods with noise:

```
t = np.linspace(0, 1, 1024)
f1 = 10
f2 = 50
F=np.sin(2*np.pi*f1*t)+np.sin(2*np.pi*f2*t)+0.2*rand.randn(len(t))
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k') # black line for data
plt.plot(t, np.sin(2*np.pi*f1*t), 'b') # blue line for 1st periodic
plt.plot(t, np.sin(2*np.pi*f2*t), 'r') # red line for 2nd periodic
plt.show()
```

23) Decompose the data ***F*** into these periodic components by choosing the window length ***L*** and grouping indices ***I*** with best accuracy on your own. Plot your found components in the same manner on figure.

24) Now, the next model data of noised time series with exponential trend:

```
t = np.linspace(0, 4, 4096)
F = np.exp(-0.4*np.pi*t) + 0.5*rand.randn(len(t))
```

25) By using the SSA method recreate the component of exponential trend and compare it against **np.exp(-0.4*np.pi*t)**. The window length ***L*** and grouping indices ***I*** choose on your own. Plot your found trend against real data on some figure.

26) Create in Python the following time series data with 4 harmonic periods and noise and then decompose it into 4 meaningful components with the similar periods:

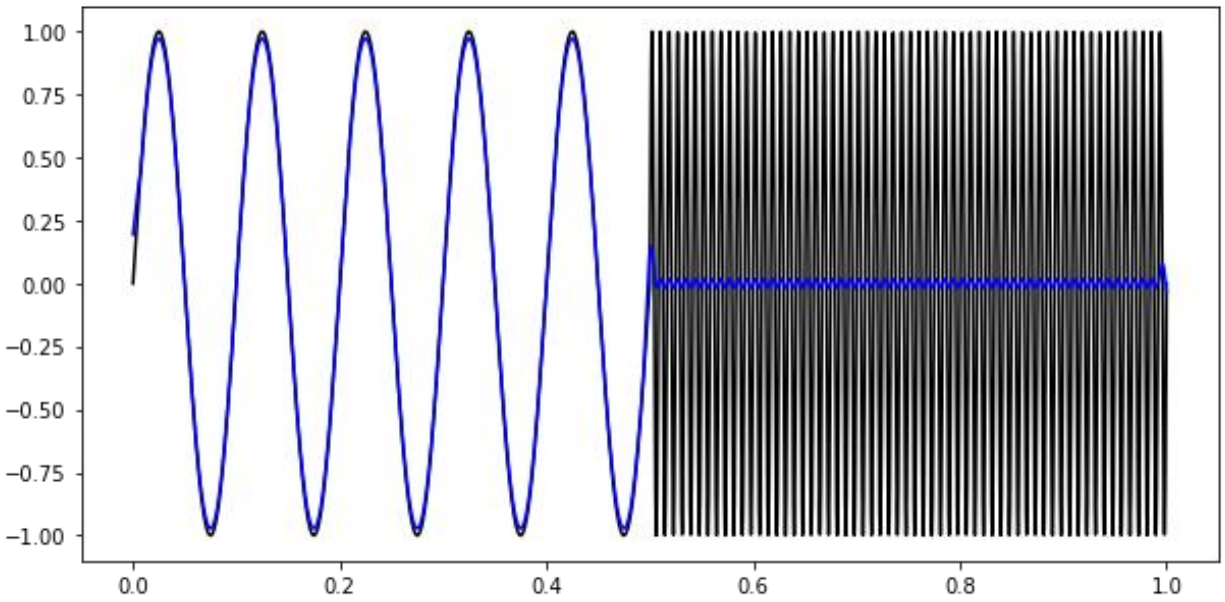
$$u(t) = \sin\left[2\pi t(f_1)\right] + \sin\left[2\pi t(f_2)\right] + \sin\left[2\pi t(f_3)\right] + \sin\left[2\pi t(f_4)\right] + \xi(t)$$

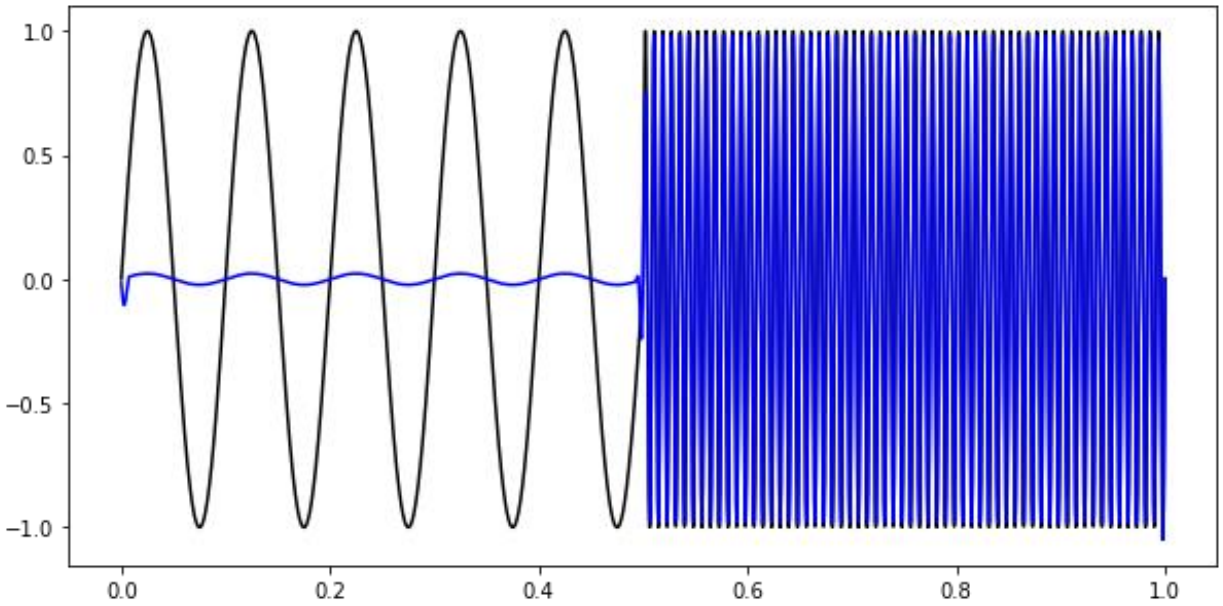
27) Plot the found components on figures against data and real periods/frequencies $f_1$, $f_2$, $f_3$, $f_4$.

28) Create the model of data with frequency break:

```
t = np.linspace(0, 1, 4096)
x2 = np.zeros(4096)
for i in range(0, len(t)//2):
   x2[i] = np.sin(2*np.pi*10*t[i])
for i in range(len(t)//2, len(t)):
   x2[i] = np.sin(2*np.pi*120*t[i])
plt.figure(figsize = (10, 5))
plt.plot(t, x2)
plt.show()
```

29) Try to decompose this data with SSA method into two components with different periods and on different time spans, also plot those components against data. The final result should be similar to the figures shown on the next page:




30) At last, load the time series with some real data:

```
file = h5py.File('lab5.mat','r')
data = file.get('EEG')
eeg = np.array(data).T
```

31) Yield the periodic component from this data with SSA method. Or maybe try to decompose data into several meaningful components. Plot found components on figures against data.

## 3. Report requirements

Report in Jupyter-notebook should include: number of laboratory task, name of student, student ID or variant number, student group number and the complete task (code, plots and figures) with student comments in the report.