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

THE REGRESSION METHODS OF TIME SERIES ANALYSIS

Laboratory Task no. 3

Yekaterinburg
2019

Ural Federal University
Institute of radioelectronics and information technologies

Contents

1. Introduction

On the previous task we have learned the functions for PSD estimations and how to use the statistical tests. In this task we should learn the basic curve fitting procedures, regression methods for trends and smoothing functions. As always, do not forget to plot figures and record down any valuable result that you received throughout this task.

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 regressive methods:

1) Import in to your code the following libraries:

```
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
import pandas as pd
from scipy import signal
import scipy.stats as stats
from statsmodels.tsa import api as tsa
%matplotlib inline
```

2) Create your time series from the column of the tables below, based on your student ID variant, as a numpy array (`np.array([...])`) with 24 points. Your time interval for the data is a linear array of 24 points between 0 and 1.

1	2	3	4	5	6	7	8
1,65	23,46	0,54	30,42	20,89	12,60	3,54	15,48
2,59	14,86	2,16	30,56	20,11	18,92	7,81	9,29
6,18	20,14	5,39	29,90	16,41	17,08	12,83	8,26
6,26	21,59	3,48	21,67	18,95	15,51	6,73	5,45
6,44	18,98	4,54	26,31	21,43	8,97	6,29	10,49
7,16	21,77	7,99	28,13	16,54	14,52	15,88	14,47
10,56	20,27	7,95	24,06	11,55	12,77	12,27	9,46
10,93	16,86	7,01	20,55	14,39	12,96	7,84	8,79
9,53	16,23	9,89	24,35	20,66	5,55	10,71	12,96
10,64	18,55	12,35	18,12	15,31	11,09	14,60	15,37
17,43	14,87	12,91	18,69	9,34	9,23	17,48	11,82
14,72	11,98	14,42	14,88	11,39	5,03	12,97	11,34
15,50	14,41	14,13	11,66	11,34	2,15	11,34	20,84
15,01	13,42	18,67	19,83	10,07	8,95	23,82	16,58
17,83	10,44	16,95	14,10	5,95	8,04	19,97	12,47
18,43	8,26	15,84	10,16	4,59	5,68	11,51	7,05
17,69	8,86	19,23	10,08	8,74	0,14	18,07	15,08
19,80	9,53	22,05	5,82	9,96	5,85	22,11	16,97
22,64	6,88	22,59	8,46	3,03	4,21	23,12	13,51
22,86	4,10	21,15	5,50	3,17	2,56	15,52	13,45
21,56	7,61	23,98	3,60	4,45	0,08	20,03	16,55
22,16	4,92	26,45	8,44	4,06	3,87	24,36	18,47
25,82	1,79	29,80	3,04	0,16	1,10	27,02	21,73
26,50	0,10	27,41	0,00	1,52	0,85	21,31	14,04

9	10	11	12	13	14	15	16
12,19	23,75	18,47	76,88	8,48	24,78	3,07	10,22
8,41	28,00	14,87	69,88	10,43	22,55	6,26	10,06
14,68	33,01	21,51	74,55	18,97	30,85	7,46	13,34
8,64	16,78	9,07	59,75	6,37	23,88	6,48	11,92
32,94	18,16	16,02	72,21	9,86	27,78	1,64	8,81
22,61	20,05	11,12	66,85	1,29	12,71	5,41	8,10
45,92	3,18	23,45	69,91	13,23	25,25	6,18	12,51
23,63	16,11	6,45	68,05	8,50	25,70	16,93	11,16
18,59	21,66	14,21	72,59	11,68	34,44	2,71	8,77
36,22	20,16	8,18	42,83	10,17	23,18	6,94	4,87
50,10	24,71	14,50	67,04	14,18	29,81	8,35	10,57
46,22	15,63	3,86	56,63	2,79	22,26	11,59	10,37
23,63	16,27	10,14	61,10	26,63	22,97	5,98	6,88
47,30	18,99	9,99	44,88	15,69	16,37	10,77	9,13
40,03	21,12	14,47	52,90	20,32	22,82	14,71	10,31
56,53	8,34	0,65	46,03	17,28	14,19	14,66	7,13
38,41	14,96	8,97	46,72	22,87	16,40	11,77	3,52
51,47	17,17	2,47	46,48	23,80	7,23	27,10	0,14
6,29	20,24	12,58	31,63	28,81	13,05	9,69	6,35
35,41	8,31	3,12	21,72	28,59	4,63	22,31	5,30
67,79	12,36	6,81	21,40	35,68	3,19	19,73	1,46
74,21	14,59	0,43	11,40	35,72	4,55	25,88	1,09
79,12	21,72	4,65	10,06	39,44	0,94	29,00	2,40
45,10	28,69	5,91	0,42	40,04	11,07	32,18	1,92

17	18	19	20	21	22	23	24
11,54	0,54	6,86	11,43	10,41	4,89	15,45	5,93
0,80	4,33	3,91	7,60	7,70	3,10	11,94	3,88
12,76	3,73	6,66	12,15	10,39	5,19	11,93	5,08
11,18	5,18	6,38	10,39	10,73	1,02	18,66	5,98
8,90	2,50	8,35	11,44	12,31	6,25	12,69	7,77
8,49	3,72	6,16	10,94	9,58	5,06	10,01	6,67
11,38	4,78	7,68	13,54	11,53	5,96	8,81	6,55
10,93	5,72	7,12	11,87	11,55	6,27	10,86	6,27
9,40	3,69	8,61	13,35	13,98	6,56	11,49	8,23
9,30	4,80	5,87	11,72	10,07	6,43	10,78	6,61
12,43	6,35	7,76	13,58	11,44	6,45	10,38	7,40
11,03	6,89	7,07	10,56	11,00	6,26	13,07	7,48
10,88	6,38	8,37	11,04	11,16	7,00	10,81	8,08
11,33	5,93	8,69	8,96	9,49	4,51	12,73	7,00
13,86	9,17	6,83	11,38	10,41	5,93	12,11	6,16
14,98	9,31	6,17	9,26	9,15	6,53	15,74	5,73
12,66	4,07	6,98	9,38	8,48	6,98	17,71	7,23
12,98	9,47	3,84	8,04	5,41	8,96	15,31	3,86
18,09	12,28	4,75	10,98	6,44	5,78	11,15	5,63
17,49	13,32	4,05	7,95	6,15	5,87	18,12	5,66
14,97	9,87	4,88	7,67	3,17	6,21	20,81	5,71
14,42	12,73	0,51	4,69	0,47	3,33	19,90	2,62
21,29	16,73	2,60	7,16	1,80	5,21	19,15	3,89
20,66	17,05	0,90	4,17	1,26	4,63	22,43	3,44

3) Plot on figure your time series data.
4) Let's create the regressive **linear model** of trend for it: $\tau(t) = \beta_0 + \beta_1 t$.
5) To find those coefficients you will need to solve the equation $y = X\beta$.
6) For a linear trend: $X = \begin{vmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_N \end{vmatrix}, y = \begin{vmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{vmatrix}, \beta = \begin{vmatrix} \beta_0 \\ \beta_1 \end{vmatrix}$.
7) To solve this linear problem, you can use Python function:

```
B = np.linalg.lstsq(X,Y)
```

8) After this function you will receive your coefficients for trend in first element:

```
B = B[0]    # take the coefficients β
print(B)
plt.figure(figsize = (10, 5))
plt.plot(t, Y)                      # plot time series data
plt.plot(t, B[0] + B[1] * t , 'r')  # plot the trend for it
plt.show()
```

9) The method of matrices and linear solving is a basic method for the **regression** methods. But in Python there is a whole lot of other, more efficient and complex methods.

10) For example, with polynomial curves of **numpy**:

```
bb = np.polyfit(t, Y, 1)          # polynomial of first order = linear
plt.figure(figsize = (10, 5))
plt.plot(t, Y)
plt.plot(t, bb[1] + bb[0]*t, 'r') # Note! Coef. β in reverse order
plt.show()
```

A better approach is to use function **poly1d** to create your trend:

```
p = np.poly1d(bb)       # create a polynomial class with your coef.
plt.figure(figsize = (10, 5))
plt.plot(t, Y)
# estimate a polynomial curve in corresponding time points
plt.plot(t, p(t), 'g')
plt.show()
```

11) Another way by using linear regression from **scipy.stats**:

```
out = stats.linregress(t, Y)
print(out)          # print coefficients and statistic of regression
plt.figure(figsize = (10, 5))
plt.plot(t, Y)              # plot the data and the trend
plt.plot(t, out.intercept + out.slope*t, 'r')
plt.show()
```

12) Based on fitting curves from **scipy.optimize**:

```
def func(t, b0, b1):      # define the form of trend
    return b0 + b1 * t    # linear trend with two parameters from time

from scipy.optimize import curve_fit
popt, pcov = curve_fit(func, t, Y)    # fit the data to our function
print(popt)       # here are the coefficients b0 & b1 of trend
print(pcov)       # here is the covariance matrix of fit errors
```

13) Linear Regression with **sklearn** library:

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(t.reshape(-1,1), Y)
print(reg.coef_)          # here is the linear coefficient b1
print(reg.intercept_)     # and here is the displacement b0
print(reg.score(t.reshape(-1,1), Y))
                # and here is the value of R^2 score, close to 1.0 is the better
```

14) Also another way by using **statsmodel**:

```
import statsmodels.api as sm
x_ = sm.add_constant(t.reshape(-1,1)) # linear model τ(t) = β0 + β1t
smm = sm.OLS(Y, x_)   # solve by Ordinary Least Squares = OLS method
res = smm.fit()       # fit the parameters of model to data
print(res.params)     # receive the results
```

15) The last example is the closest one to the *machine learning* methods, because we only provide the form of final solution (linear model of trend), the method to solve it (OLS) and receive the final result from calculations.

16) Make sure, that all of those methods provide the same results.

17) Now, try to estimate the trend of **second order (quadratic)** and **third order (cubic)**, with methods learned earlier. Make sure, that all of those methods provide the same results, although, obviously, not all of those methods are applicable for a higher order of trend.

18) Estimate the trend with **exponential** model: $\tau(t) = \beta_0 e^{\beta_1 t}$.

19) All of those trends (linear, quadratic, cubic, exponential) plot on the same figure with the initial time series data.

20) Now, let's try to estimate the trend of data by **smoothing**. For this, insert in to your Python notebook this script with function:

```
def smooth(x, window_len):
    if window_len<3:
        return x
    s=np.r_[2*x[0]-x[window_len-1::-1], x, 2*x[-1]-x[-1:-window_len:-1]]
    w=np.ones(window_len, 'd')
    y=np.convolve(w/w.sum(), s, mode='same')
    return y[window_len:-window_len+1]
```

21) Use this function to smooth your data, for example, on 3 points:

```
Smoothed_data = smooth(Y, 3)      # smoothing by 3 points
```

22) Plot the smoothed trends on the same plot by smoothing with **3**, **7** and **11** points of data, on the same figure as initial time series data. Compare them with each other.

23) Create your own function to smooth data by **3** points:

$$\tau_j = 1/3\left(y_{j-1} + y_j + y_{j+1}\right)$$

24) Create your own function to smooth data by **7** points, similarly. Compare the results with function smooth(). There might be some discrepancies in the estimated values on the edges of time-span.

25) For the last, estimate the exponential trend $\tau_j = (1-\alpha)\tau_{j-1} + \alpha y_j,\ \ j = 1,2,...,N$.

26) The value α you need to choose on your own: just remember, that this value is larger than 0 and smaller than 1. Try to find the best fit for your variant data.

27) Plot this exponential trend on the same figure, as time series.

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.