Ural Federal University

named after the first President of Russia B.N.Yeltsin

Institute of radioelectronics and information technologies

TIME SERIES FORECAST

Laboratory Task no. 8

Yekaterinburg

2019

Contents

1. Introduction 3

2. Laboratory Task 3

3. Report requirements 10

1. Introduction

In most practical cases the forecast of time series estimated after its analysis and creation of its model (as function of time). We already know some of those methods and will actively use them for the forecast of time series data. For a better approach, the forecast for this task would be in form of **retrospective forecast** (meaning forecast on already known points as if they were unknown to us) on some real data time series called FORT.

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 and forecast:

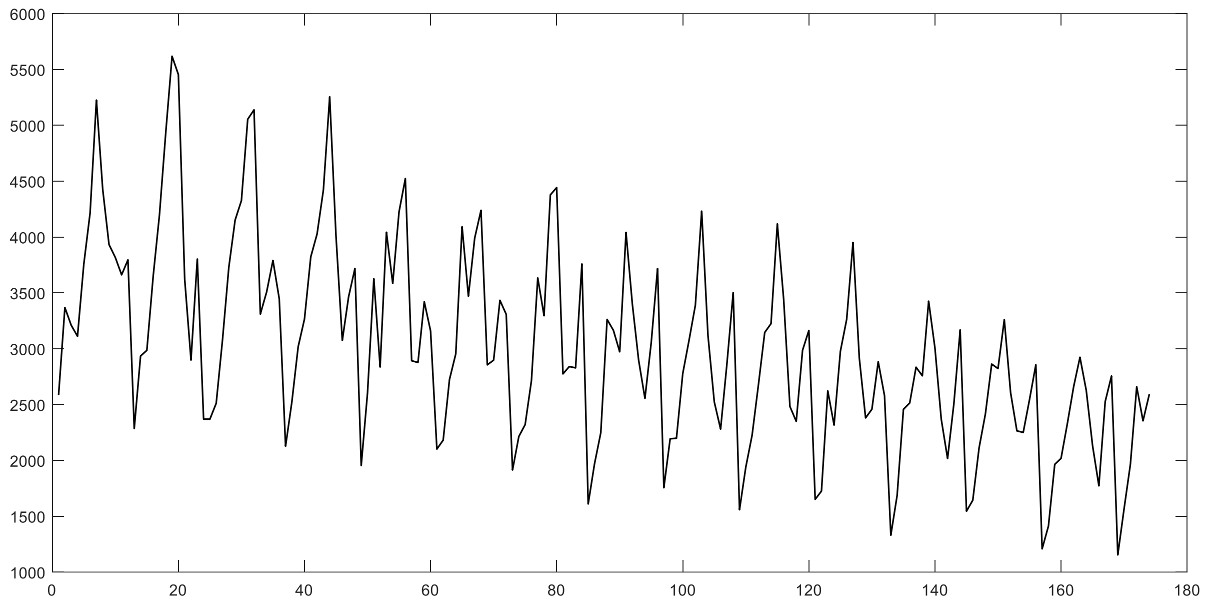
1) Import into your code the following libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy.stats as stats
from statsmodels.tsa import api as tsa
from statsmodels.tsa.arima_model import ARIMA
%matplotlib inline
```

2) Load the data from *mat*-file – short time series FORT of 174 points:

```
file = h5py.File('Fort.mat', 'r')
data = file.get('Fort')
Fort = np.array(data)
plt.figure(figsize = (10, 5))
plt.plot(Fort, 'k')
plt.show()
```

3) The time series is presented below. There is a clear periodicity in data with descending trend. The data is short for fast *Python* calculations.



4) The **retrospective forecast** (meaning forecast on already known observations as if they were unknown to us) for all next methods is created for the last 24 points of data. In order to achieve this let's create a truncated version of time series called ***Z***:

```
Z = Fort[:len(Fort)-24+1] # truncate the last forecasted 24 points
t=np.arange(0, len(Z), 1) # time-span for time series
t=t.reshape(-1,1)
plt.figure(figsize = (10, 5))
plt.plot(Fort, 'k') # initial data
plt.plot(t, Z, 'b') # truncated data
plt.show()
```

5) Let's begin with the simplest methods for the forecast. First, let's create the linear trend of data with methods from Laboratory Task 03. This regression curve can be used as the simplified forecast scheme on continued time-span.

6) Let's start with time series forecast with regressions models. For example, regression forecast with **sklearn**:

```
t=np.arange(0, len(Z), 1) # time-span of truncated data
t=t.reshape(-1,1)
t0=np.arange(0, len(Fort), 1) # time-span of full time series
t0=t0.reshape(-1,1)
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(t, Z) # regression model
plt.figure(figsize = (10, 5))
plt.plot(t, Z, 'k')
plt.plot(t0, reg.predict(t0), 'r') # forecast and predict future observations
plt.plot(t0[-24:], Fort[-24:], 'b') # real observations to compare
plt.show()
```

7) Using the library **statsmodels**:

```
import statsmodels.api as sm
x_ = sm.add_constant(t)
smm = sm.OLS(Z, x_)
res = smm.fit() # create the linear regression model
print(res.params)
plt.figure(figsize = (10, 5))
plt.plot(t, Z, 'k')
plt.plot(t0, res.predict(sm.add_constant(t0)), 'r') # forecast
plt.plot(t0[-24:], Fort[-24:], 'b') # real observations to compare
plt.show()
```

8) With polynomial curves from **polyfit**:

```
bb = np.polyfit(t.reshape(1,-1)[0], Z.reshape(1,-1)[0], 1)
p = np.poly1d(bb) # polynomial class
plt.figure(figsize = (10, 5))
plt.plot(t, Z, 'k')
plt.plot(t0, p(t0), 'r') # data estimated on full time-span
plt.plot(t0[-24:], Fort[-24:], 'b')
plt.show()
```

9) With regression methods for functions from **scipy.optimize**:

```
def func(t, b0, b1):
    return b0 + b1 * t

from scipy.optimize import curve_fit
popt, pcov = curve_fit(func, t.reshape(1,-1)[0], Z.reshape(1,-1)[0])
plt.figure(figsize = (10, 5))
plt.plot(t, Z, 'k')
plt.plot(t0, t0*popt[1]+popt[0], 'r') # here is the forecast
plt.plot(t0[-24:], Fort[-24:], 'b')
plt.show()
```

10) Now, on your own, similarly estimate the **trends of second and third order** (quadratic and cubic) and create the regressive forecasts with it. See Lab Task 03 for examples and additional help.

11) The quality of the forecast can be measured visually, but it is always better to rely on numbers to estimate **the accuracy of the forecast**. Estimate the accuracy of your every forecast by creating the following *Python* functions (where M – is a number of forecasted points):

Mean error of the forecast: $\overline{\Delta}^* = \frac{\sum_{i=1}^{M} \Delta_i^*}{M} = \frac{\sum_{i=1}^{M} \left| y(t_i) - y_i \right|}{M}$

STD of the forecast error: $\sigma_e = \sqrt{\frac{\sum_{i=1}^{M} \left(y(t_i) - y_i \right)^2}{M}}$

Mean error of the forecast approximation: $\bar{\varepsilon} = \frac{1}{M} \sum_{i=1}^{M} \frac{\left| y(t_i) - y_i \right|}{y(t_i)} \cdot 100\%$

Coefficient of disparity 1: $KH_1 = \sqrt{\frac{\sum_{i=1}^{M} \left(y_i - y(t_i) \right)^2}{\sum_{i=1}^{M} y(t_i)^2}}$

Coefficient of disparity 2: $KH_2 = \sqrt{\frac{\sum_{i=1}^{M} \left(y_i - y(t_i) \right)^2}{\sum_{i=1}^{M} \left(\bar{y} - y(t_i) \right)^2}}$

Apply all of these functions to your forecasted points against real observations and receive the numerical values of the forecast accuracy.

12) Let's estimate the confidence intervals of our forecast:

$$\tau_B(t) = \tau(t) + \delta(t),$$
$$\tau_H(t) = \tau(t) - \delta(t),$$

One needs to estimate the value of $\delta(t)$. For a **trend of first order** this value can be calculated as:

$$\delta_{p=1}(t_l) = 1.96 \cdot S \cdot \sqrt{1 + \frac{1}{N} + \frac{\left(\tau(t_l) - \bar{\tau}\right)^2}{\sum_{i=1}^{N}\left(\tau_i - \bar{\tau}\right)^2}}$$

where $S = \sqrt{\frac{\sum_{i=1}^{N} e_i^2}{N-2}}$, e_i – residual data or errors of the forecast, meaning the difference between forecasted points and real observations.

13) Also, estimate the confidence intervals for the **trends of second and third order**, but with a simplified empirical estimation:

$$\delta(t) = 1.96 \cdot \sqrt{\frac{\sum_{i=1}^{M}\left(y(t_i) - y_i\right)^2}{M}}$$

14) The method of regressive functions from library **scipy.optimize** allows one to establish **any type of function**. Just for your interest, try to describe your own function in this method, that will provide some meaningful forecast of presented data. Check its accuracy accordingly and estimate its empirical confidence intervals.

15) Now let’s create a forecast with another type of learned models called **ARIMA** models. First of all, you should note, that the creation of ARIMA model in initial pure form requires the data **to be centered to zero** (i.e. its expectation value is constant and close to zero; trend-stationary data form). Our practical real data obviously do not satisfy this condition. What to do? Obviously, reform the data into trend-stationary form: **estimate the linear trend** of data (see linear regression of first order), and **subtract** this linear trend from time series, reforming the data into zero-mean form.

16) According to learned technique from Laboratory Task 04, estimate the optimal model of ARIMA(p, d, q) of some order (**every parameter is estimated by students themselves**) for truncated data ***Z*** with subtracted trend. For example, how you search for some optimal model:

```
arimaz = ARIMA(Z_minus_trend, order = (p, d, q))
model_fit = arimaz.fit(disp = False) # fit the ARIMA model to data
print(model_fit.summary())
```

17) If everything has been done correctly, the plot for the forecast even with complete confidence intervals is pretty simple in *Python*:

```
model_fit.plot_predict(0, len(Fort))
```

18) Still, the forecast should be compared against the initial observations, especially the last 24 points in case of our retrospective forecast. In this case, the actual plot of the forecast against real data can be reconstructed like this:

```
plt.figure(figsize = (10, 5))
model_fit.plot_predict(0, len(Fort)) # forecast of ARIMA model
plt.plot(t0, Fort-(trend_as_func_of_t0), 'r') # Initial data minus trend
plt.show()
```

19) In order to estimate the numerical values of the forecast accuracy one requires the exact values of the forecast. There is a function **predict** for this:
model_fit.predict(len(Z), len(Fort)) # output is the array of the forecasted points

20) Use these points from output array in order to estimate the accuracy of the forecast with ARIMA model with your functions from point 11.

21) At last, try to estimate the optimal ARIMA model for initial truncated time series **without subtracting the trend from it** (no trend-stationary form). Note the differences in the work of *Python* functions in this non-zero mean case, also check the accuracy of fit and forecast, the values of AIC, BIC and so on.

22) Similarly, for non-trend-stationary form of ARIMA model plot the forecast against initial time series and estimate its accuracy with functions from point 11. Compare the results with initial ARIMA model.

23) Complete the report and the course of Time Series Data Analysis.

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.