OLS analysis of serially correlated dependent and independent variables

Sandeep Kiran Gudla

May 25, 2019

We know that OLS gives the best linear unbiased estimators when all the Gauss Markov Assumptions are met. One such assumption is random sampling. Let us see how the estimators behave when this assumption is violated. One of the way to violate this error is creating serial dependencies on dependent and independent variables.

Let us begin by generating dependent and independent variables using the following data generating processes.

$$x_t = x_{t-1,i} + u_{t,i}, u_{t,i} \sim iid\mathcal{N}(0, \sigma^2)$$

for t = 1,2,....T and $i = 1, 2, ...n$.

Let's use a similar process to generate another feature, say $\{y_t\}$

$$y_t = y_{t-1} + e_t, e_t \sim iid\mathcal{N}(0, \sigma^2)$$

for $t = 1, 2, ...T$

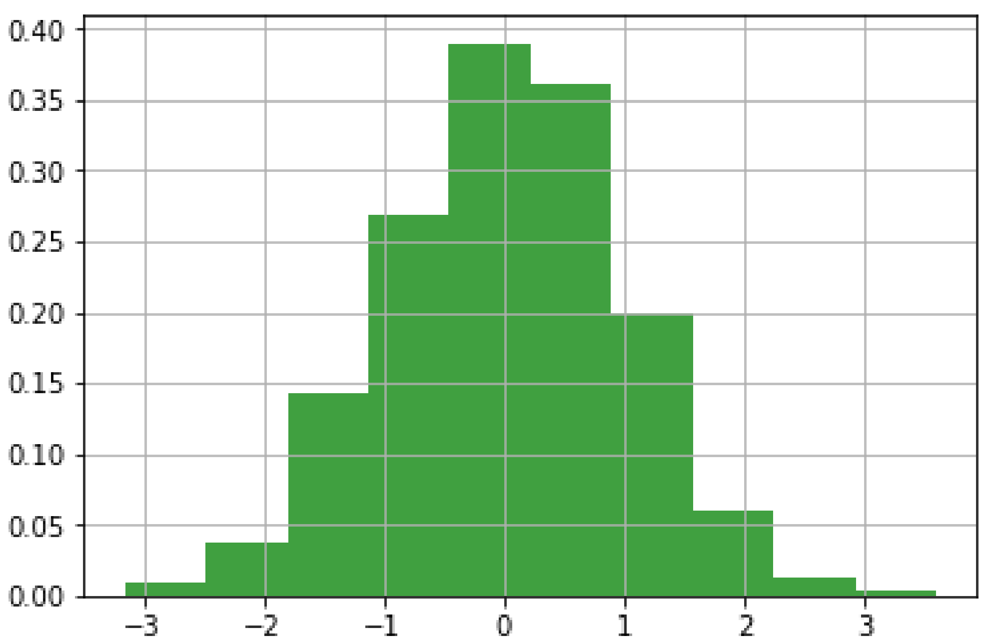
```
#importing the necessary packages
import numpy as np
import pandas as pd
import random
from statsmodels import api as sm
import matplotlib.pyplot as plt
import scipy as sc
```

Let's generate $x_{t,i}$ for $t = 1$ and $i = 1$, x$_1 = x_0 + u_{1,i}, u_{1,i} \sim iid\mathcal{N}(0, \sigma^2)$ $and y_t$ for $t = 1$, y$_1 = y_0 + e_1, e_1 \sim iid\mathcal{N}(0, \sigma^2)$

```
t=1
e=[]
y=[]
u=[]
x=[]
#initial y gen
y.append(random.normalvariate(4,1))
#initial x gen
x.append(random.normalvariate(10,1))
#generate errors
simn = 1000
```

```
random.seed(2)
for isim in range(0,simn):
    #error e gen
    e.append(random.normalvariate(0,1))
    #error u gen
    u.append(random.normalvariate(0,1))
for isim in range(1,simn):
    #generate y
    y.append(y[isim-t]+e[isim])
    #generate x
    x.append(x[isim-t]+u[isim])
x=np.array(x).reshape(-1,1)
y=np.array(y)
ones = np.ones((len(x),1))
x = np.hstack((ones,x))
```

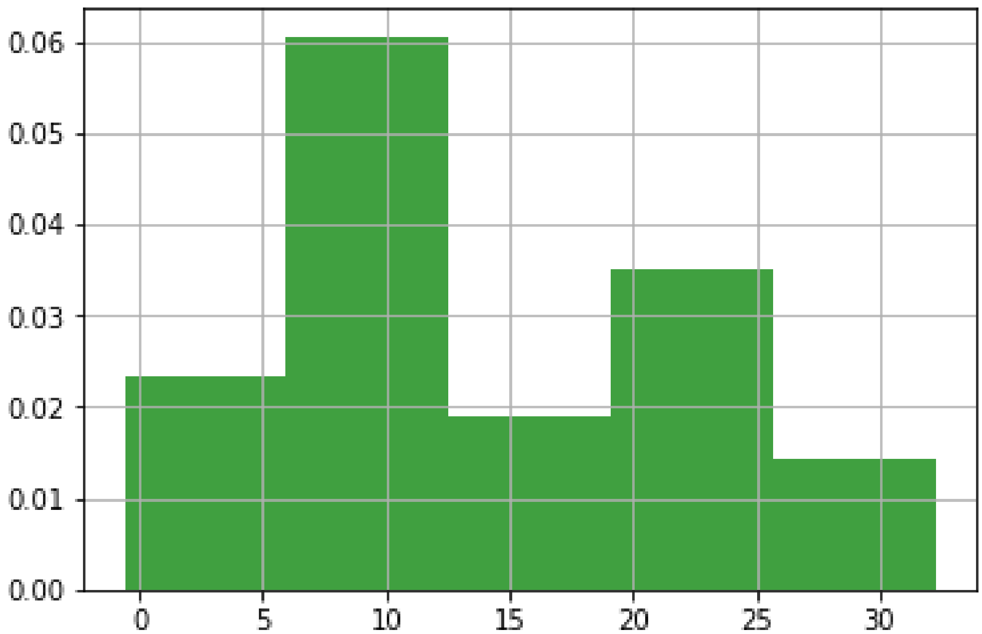
```
# the histogram of the error
n, bins, patches = plt.hist(e, 10, density=True, facecolor='g', alpha=0.75)
plt.grid(True)
plt.show()
```



Here, we can see that the error is normally distributed about zero.

Let's analyze y

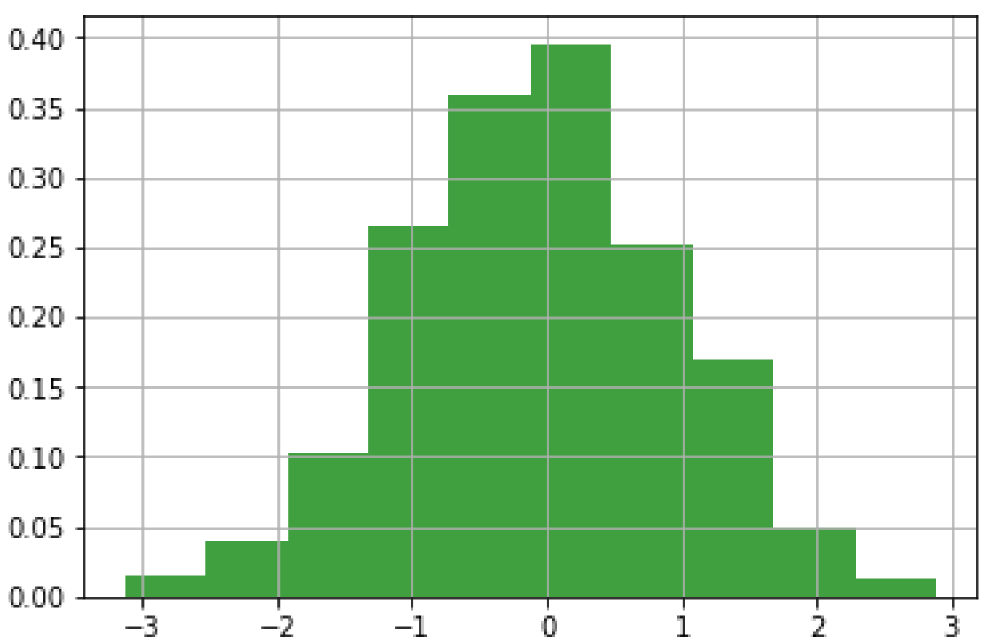
```
# the histogram of the y
n, bins, patches = plt.hist(y, 5, density=True, facecolor='g', alpha=0.75)
plt.grid(True)
plt.show()
```



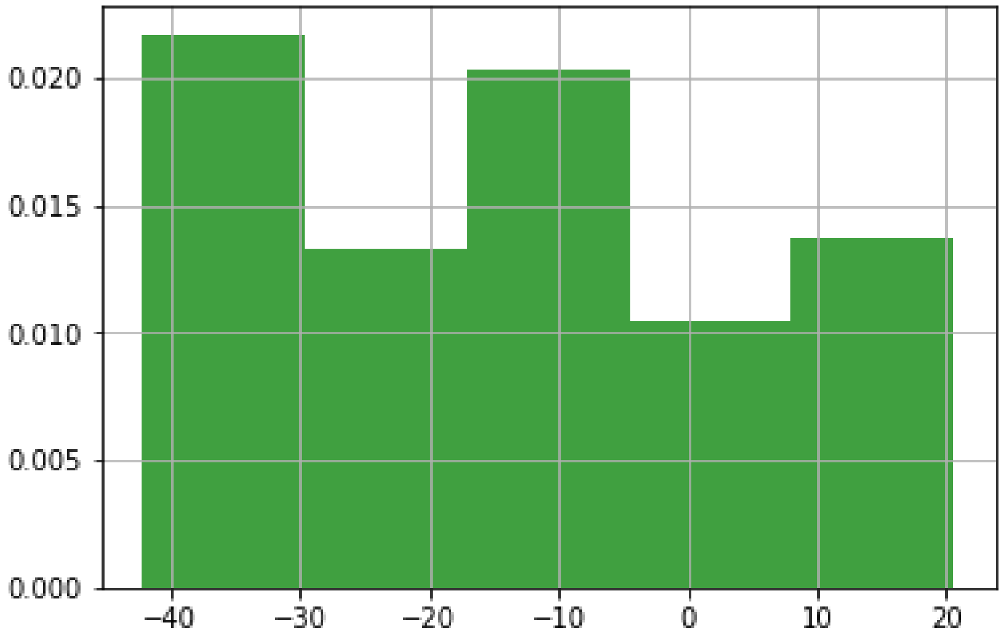
We can see that y is not normally distributed.

Similarly, let's see the histograms of u and x.

```
# the histogram of the u
n, bins, patches = plt.hist(u, 10, density=True, facecolor='g', alpha=0.75)
plt.grid(True)
plt.show()
```



```
# the histogram of the x
n, bins, patches = plt.hist(x[:,1], 5, density=True, facecolor='g', alpha=0.75)
plt.grid(True)
plt.show()
```



Let's build the following model using OLS

$y_t = \beta_0 + \beta_1 x_t + \epsilon_t$

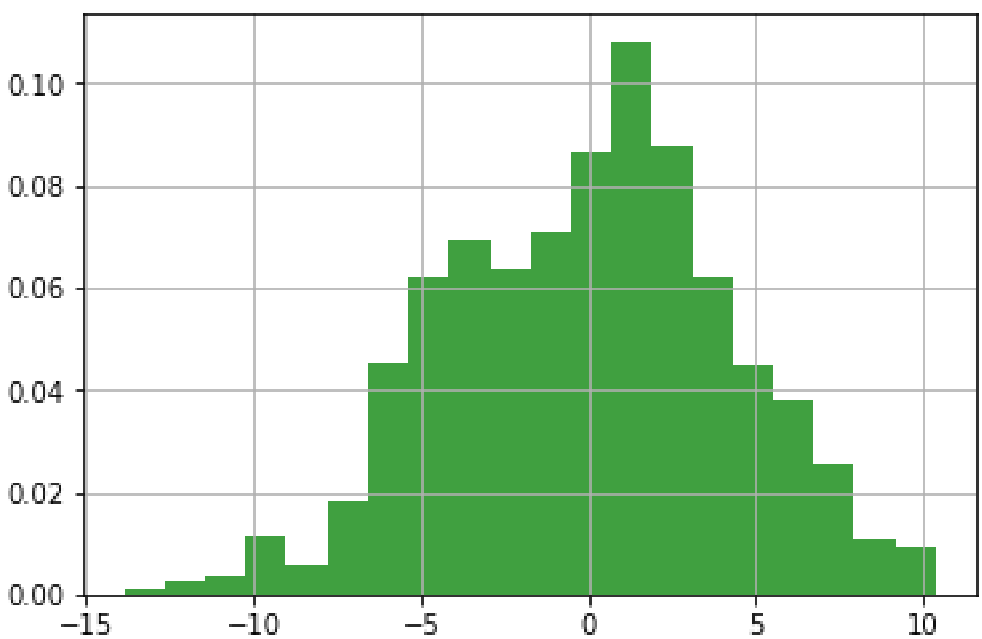
```
# regression of x on y
model1 = sm.OLS(y,x).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.732
Model:                            OLS   Adj. R-squared:                  0.732
Method:                 Least Squares   F-statistic:                     2730.
Date:                Tue, 21 May 2019   Prob (F-statistic):          7.47e-288
Time:                        17:03:52   Log-Likelihood:                -2860.2
No. Observations:                1000   AIC:                             5724.
Df Residuals:                     998   BIC:                             5734.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
```

```
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const           8.7965      0.167     52.775      0.000       8.469       9.124
x1             -0.3762      0.007    -52.249      0.000      -0.390      -0.362
==============================================================================
Omnibus:                        6.544   Durbin-Watson:                   0.061
Prob(Omnibus):                  0.038   Jarque-Bera (JB):                5.690
Skew:                          -0.118   Prob(JB):                       0.0581
Kurtosis:                       2.715   Cond. No.                         28.9
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

We can see that the Durbin-Watson score is 0.061 which suggests that there is positive auto correlation

```python
# the histogram of the data
residuals=np.array(model1.resid).reshape(-1,1)
n, bins, patches = plt.hist(residuals, 20, density=True, facecolor='g', alpha=0.75)
#plt.axis([-1, 1, 0, 20])
plt.grid(True)
plt.show()
```



Here, we can see that the error terms are not normally distributed, there for the assumptions are voilated and the OLS estimators are baised.

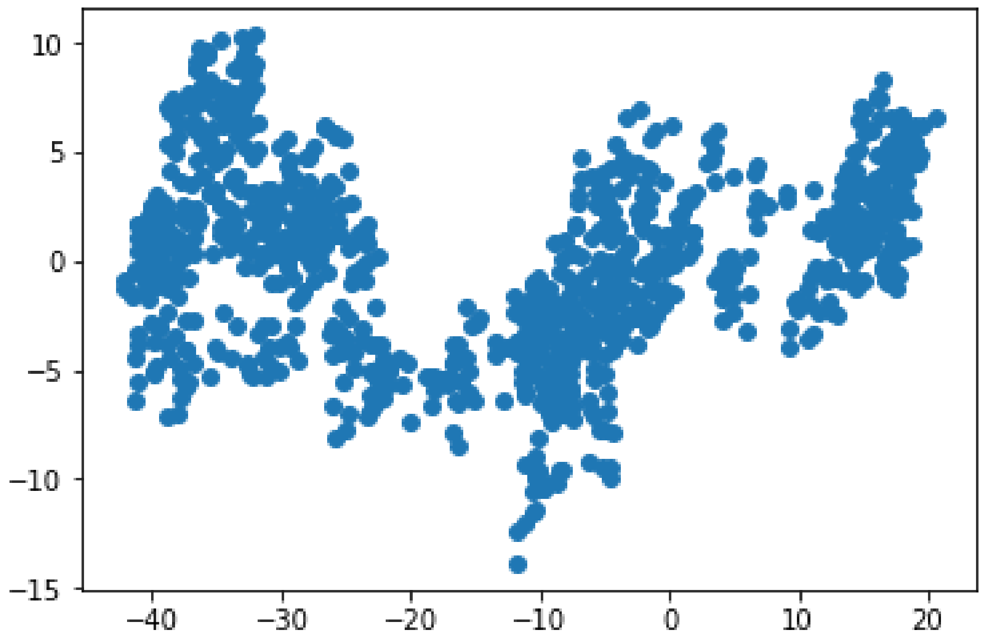
Lets look into the correlation of error terms.

```
model1 = sm.OLS(y,x).fit(cov_type='HC0')
        print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.732
Model:                            OLS   Adj. R-squared:                  0.732
Method:                 Least Squares   F-statistic:                     3141.
Date:                Tue, 21 May 2019   Prob (F-statistic):         1.58e-310
Time:                        17:03:56   Log-Likelihood:                -2860.2
No. Observations:                1000   AIC:                             5724.
Df Residuals:                     998   BIC:                             5734.
Df Model:                           1
Covariance Type:                  HC0
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          8.7965      0.153     57.330      0.000       8.496       9.097
x1            -0.3762      0.007    -56.045      0.000      -0.389      -0.363
==============================================================================
Omnibus:                        6.544   Durbin-Watson:                   0.061
Prob(Omnibus):                  0.038   Jarque-Bera (JB):                5.690
Skew:                          -0.118   Prob(JB):                       0.0581
Kurtosis:                       2.715   Cond. No.                         28.9
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
```

Let's plot residual plots to get a better understanding of the error distribution.

```
plt.scatter(x[:,1],residuals.flatten())
```



We can see that there is heterogeneity in error terms. This leads to bias in our estimators and therefore, the OLS estimators are no longer BLUE (best linear unbiased estimators).

```
sc.stats.levene(y,x[:,1])
```

```
LeveneResult - statistic = 663.1214037868205, pvalue=1.6141243531100978e-126
```

```
sc.stats.ttest_ind(y,x[:,1])
```

```
Ttest_indResult - statistic = 43.294390393344614, pvalue=2.0469050973291668e-289
```

Based on the Levene Test and Welch's t-test, the null hypothesis that the random sampling distribution has equal variance can be rejected.

Let's analyze the case for Multivariate Linear Regressions

```
t=1
e=[]
y=[]
u=[]
x=[]
#initial y gen
n = 2
y.append(random.normalvariate(4,1))
#initial x gen
x.append(np.random.normal(10,1,n))
#generate errors
```

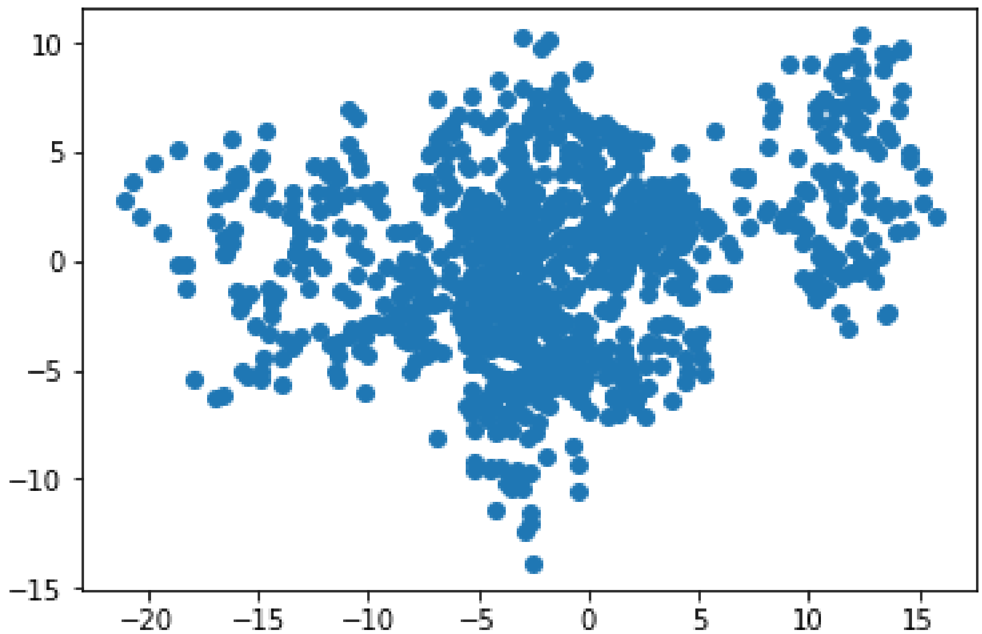
```
        simn = 1000
        random.seed(2)
        for isim in range(0,simn):
            #error e gen
            e.append(random.normalvariate(0,1))
            #error u gen
            u.append(np.random.normal(0,1,n))
        for isim in range(1,simn):
            #generate y
            y.append(y[isim-t]+e[isim])
            #generate x
            x.append(x[isim-t]+u[isim])
        x=np.array(x).reshape(-1,n)
        y=np.array(y)
        ones = np.ones((len(x),1))
        x = np.hstack((ones,x))

# regression of x on y
        model1 = sm.OLS(y,x).fit()
        print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.612
Model:                            OLS   Adj. R-squared:                  0.612
Method:                 Least Squares   F-statistic:                     787.7
Date:                Tue, 21 May 2019   Prob (F-statistic):          6.15e-206
Time:                        20:45:46   Log-Likelihood:                -3221.0
No. Observations:                1000   AIC:                             6448.
Df Residuals:                     997   BIC:                             6463.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.0352      0.426     28.281      0.000      11.200      12.870
x1            -0.2656      0.031     -8.548      0.000      -0.327      -0.205
x2            -0.7065      0.022    -31.722      0.000      -0.750      -0.663
==============================================================================
Omnibus:                       14.657   Durbin-Watson:                   0.040
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               13.561
Skew:                           0.239   Prob(JB):                      0.00114
Kurtosis:                       2.689   Cond. No.                         41.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

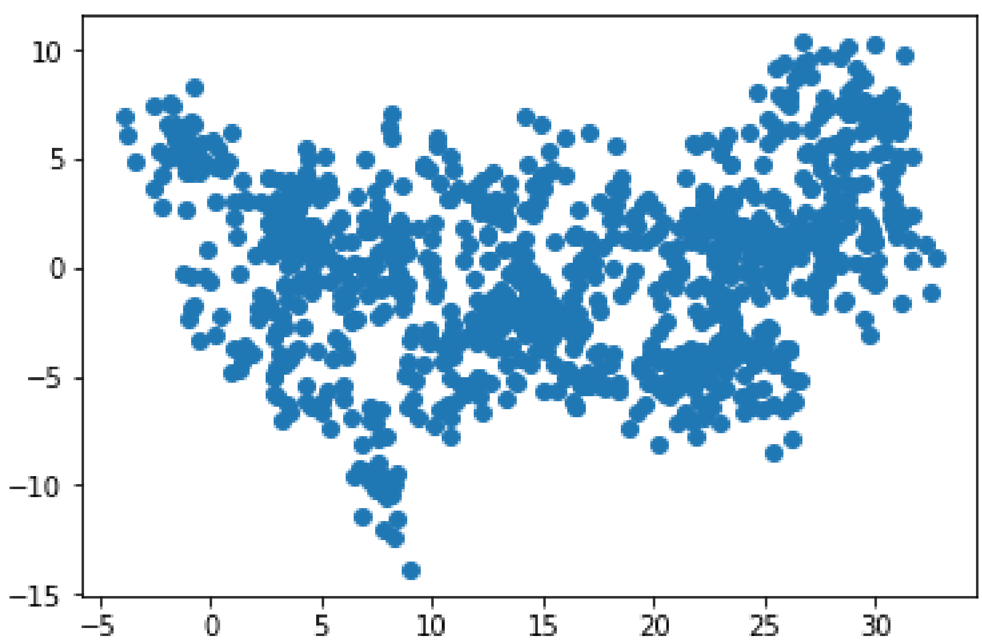
```
model1 = sm.OLS(y,x).fit(cov_type='HC0')
        print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.612
Model:                            OLS   Adj. R-squared:                  0.612
Method:                 Least Squares   F-statistic:                     828.4
Date:                Tue, 21 May 2019   Prob (F-statistic):          1.11e-212
Time:                        20:48:45   Log-Likelihood:                -3221.0
No. Observations:                1000   AIC:                             6448.
Df Residuals:                     997   BIC:                             6463.
Df Model:                           2
Covariance Type:                  HC0
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.0352      0.329     36.594      0.000      11.391      12.680
x1            -0.2656      0.033     -7.939      0.000      -0.331      -0.200
x2            -0.7065      0.019    -36.516      0.000      -0.744      -0.669
==============================================================================
Omnibus:                       14.657   Durbin-Watson:                   0.040
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               13.561
Skew:                           0.239   Prob(JB):                      0.00114
Kurtosis:                       2.689   Cond. No.                         41.4
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
```

```
plt.scatter(x[:,1],residuals.flatten())
```




```
plt.scatter(x[:,2],residuals.flatten())
```




```
sc.stats.levene(y,x[:,1],x[:,2])
```

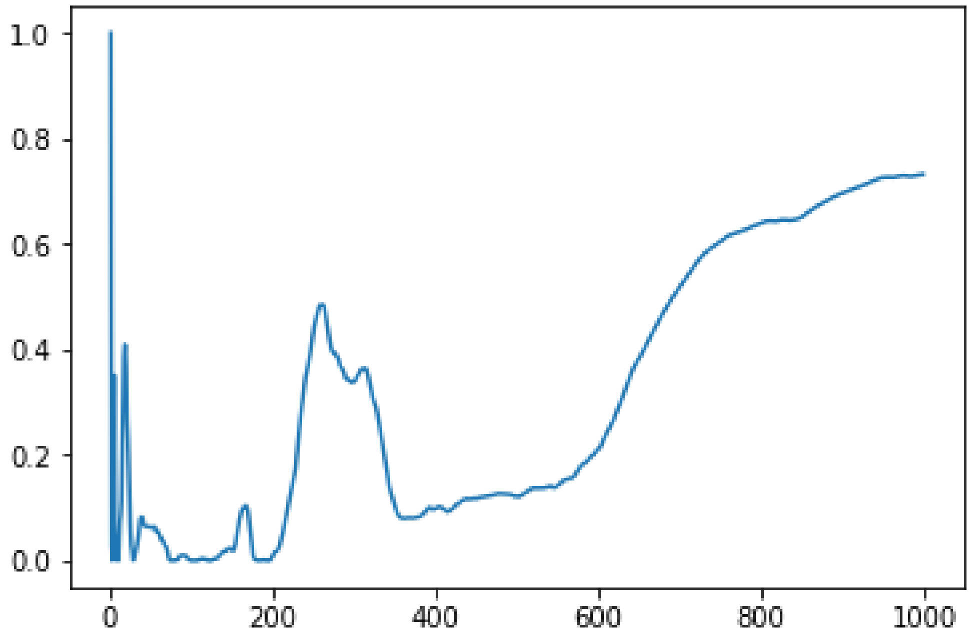
```
LeveneResult - statistic = 160.18648446129512, pvalue=8.031994802470273e-67
```

```
[sc.stats.ttest_ind(y,x[:,1]),sc.stats.ttest_ind(y,x[:,2])]
```

```
Ttest_indResult - statistic = 6.957613621196703, pvalue=4.673644578948617e-12,
          Ttest_indResult - statistic = -34.82903392988044, pvalue=4.1258493384478805e-208
```

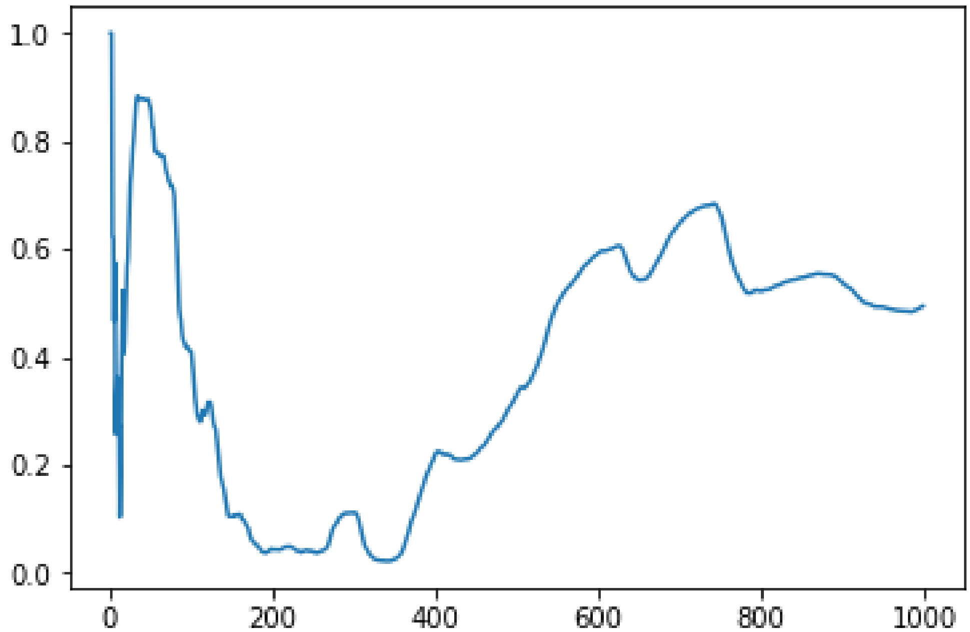
Now let's analyze how the number of samples effect the overall R-square

```
t=1
e=[]
y=[]
u=[]
x=[]
#initial y gen
y.append(random.normalvariate(4,1))
#initial x gen
x.append(random.normalvariate(10,1))
#generate errors
simn = 1000
random.seed(2)
R_sq = [0]
for isim in range(0,simn):
    #error e gen
    e.append(random.normalvariate(0,1))
    #error u gen
    u.append(random.normalvariate(0,1))
for isim in range(1,simn):
    #generate y
    y.append(y[isim-t]+e[isim])
    #generate x
    x.append(x[isim-t]+u[isim])
    x_ar = np.array(x).reshape(-1,1)
    ones = np.ones((len(x),1))
    x_ar = np.hstack((ones,x_ar))
    y_ar=np.array(y)
    model1 = sm.OLS(y_ar,x_ar).fit()
    R_sq.append(model1.rsquared)
plt.plot(range(1,simn),R_sq[1:])
```




```
t=1
e=[]
y=[]
u=[]
x=[]
#initial y gen
n = 2
y.append(random.normalvariate(4,1))
#initial x gen
x.append(np.random.normal(10,1,n))
#generate errors
simn = 1000
random.seed(2)
R_sq =[0]
for isim in range(0,simn):
    #error e gen
    e.append(random.normalvariate(0,1))
    #error u gen
    u.append(np.random.normal(0,1,n))
for isim in range(1,simn):
    #generate y
    y.append(y[isim-t]+e[isim])
    #generate x
    x.append(x[isim-t]+u[isim])
    x_ar = np.array(x).reshape(-1,n)
    ones = np.ones((len(x),1))
```

```
    x_ar = np.hstack((ones,x_ar))
    y_ar=np.array(y)
    model1 = sm.OLS(y_ar,x_ar).fit()
    R_sq.append(model1.rsquared)
plt.plot(range(1,simn),R_sq[1:])
```

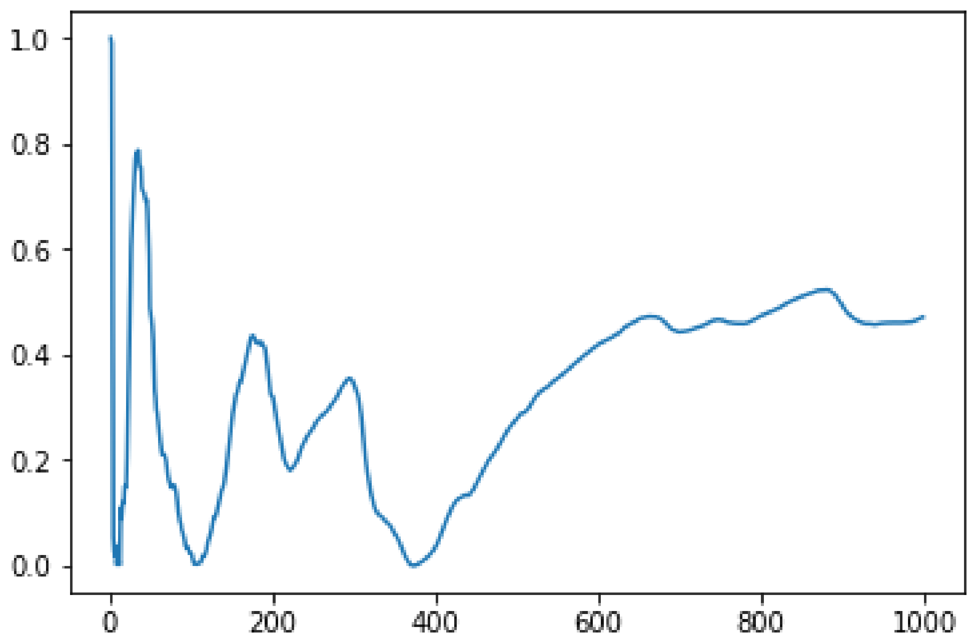


Lets start analyzing for different time lags starting from $t = 1, 2, 3.....10$

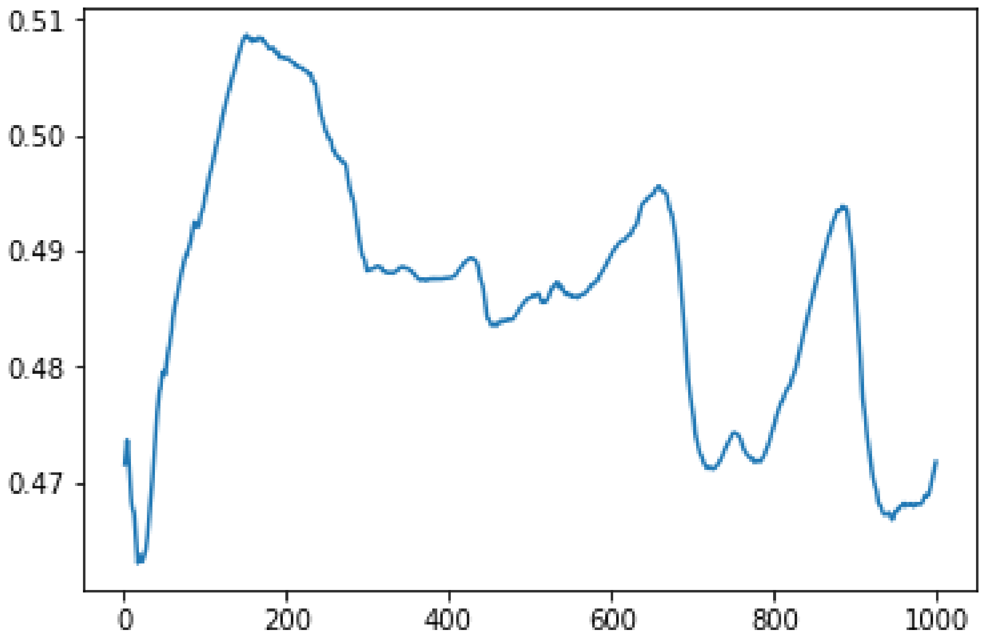
```
e=[]
y=[]
u=[]
x=[]
#initial y gen
n = 1
y.append(random.normalvariate(4,1))
#initial x gen
x.append(np.random.normal(10,1,n))
#generate errors
simn = 1000
outp = []
time = []
R_sq = []
random.seed(2)
T = [1,5,10,15,20,25,30,35,40,45,50]
for t in T:
    for isim in range(0,simn):
        #error e gen
```

```
        e.append(random.normalvariate(0,1))
        #error u gen
        u.append(np.random.normal(0,1,n))
    for isim in range(1,simn):
        #generate y
        y.append(y[isim-t]+e[isim])
        #generate x
        x.append(x[isim-t]+u[isim])
        x_ar = np.array(x).reshape(-1,n)
        ones = np.ones((len(x),1))
        x_ar = np.hstack((ones,x_ar))
        y_ar=np.array(y)
        model1 = sm.OLS(y_ar,x_ar).fit()
        outp.append(model1.rsquared)
        #time.append(t)
    print("time lags - ",t)
    R_sq.append(model1.rsquared)
    plt.plot(range(1,simn),outp)
    plt.show()
    outp=[]
```

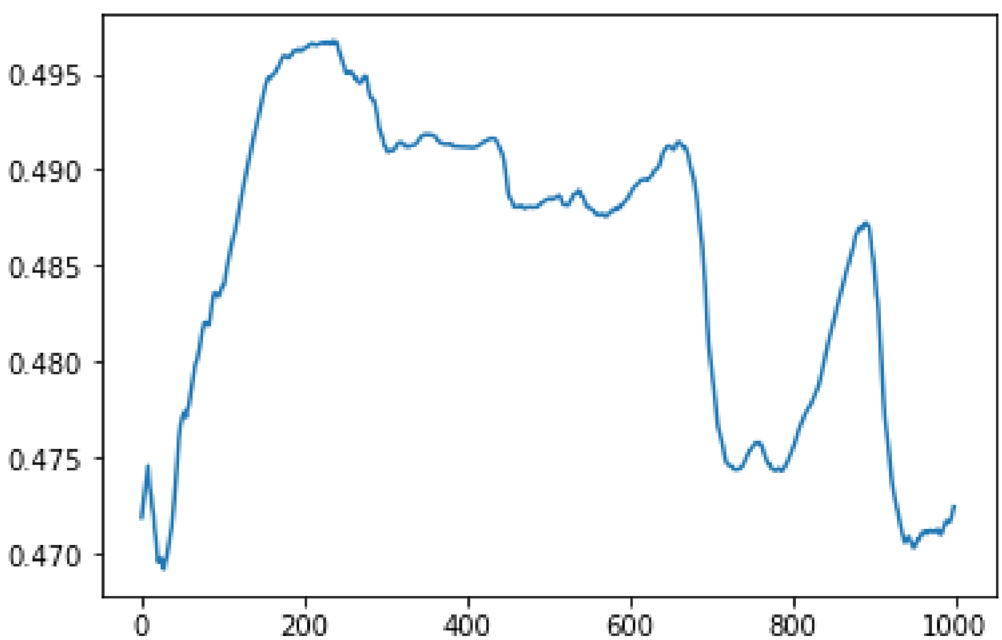
```
time lags -  1
```



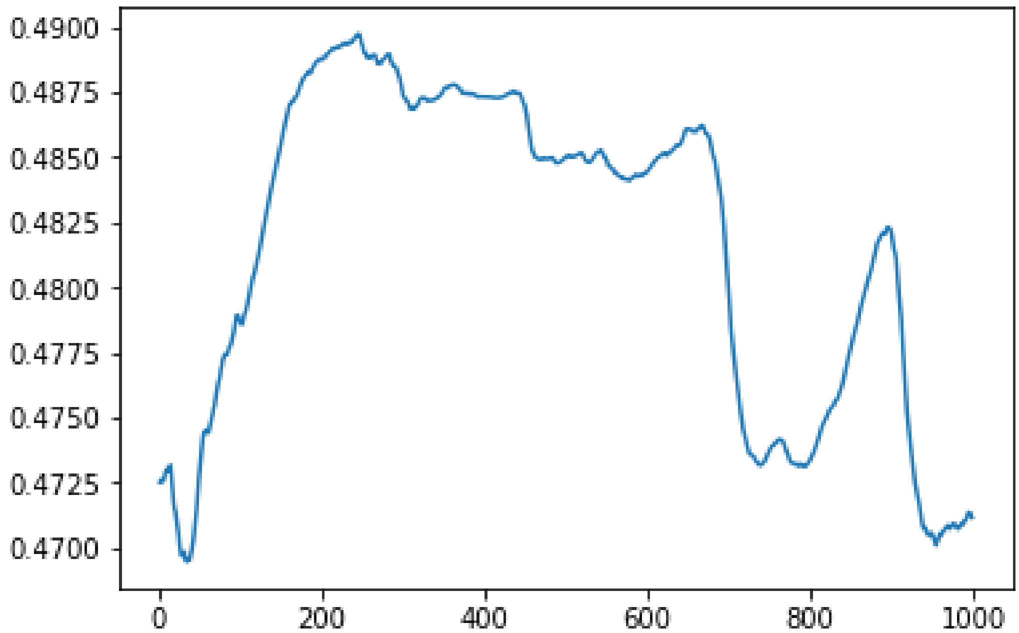

```
time lags -  5
```


0.51
0.50
0.49
0.48
0.47
0
200
400
600
800
1000

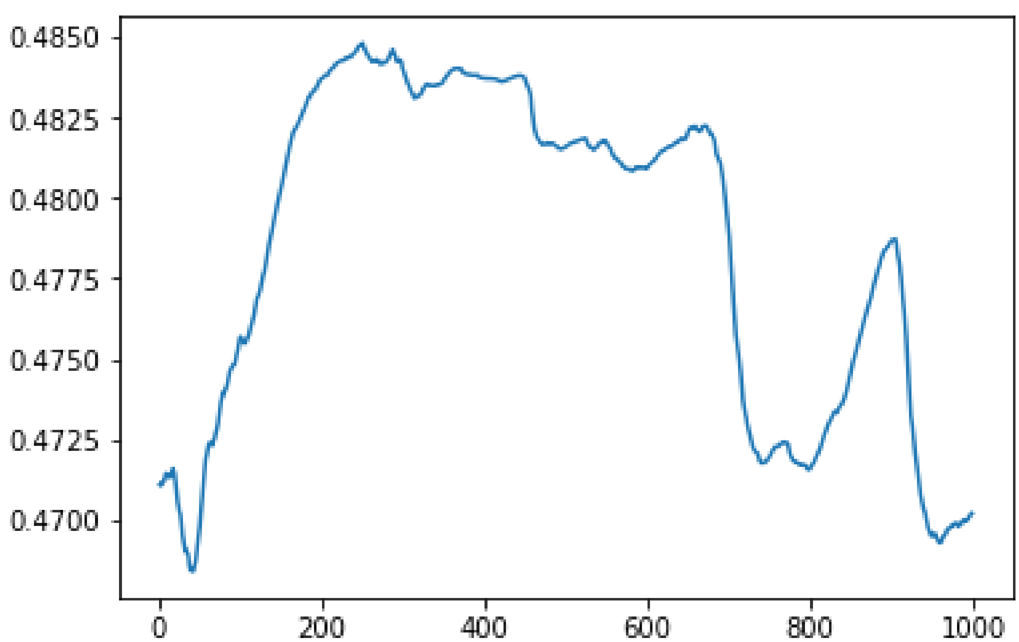
```
time lags -  10
```


0.495
0.490
0.485
0.480
0.475
0.470
0
200
400
600
800
1000

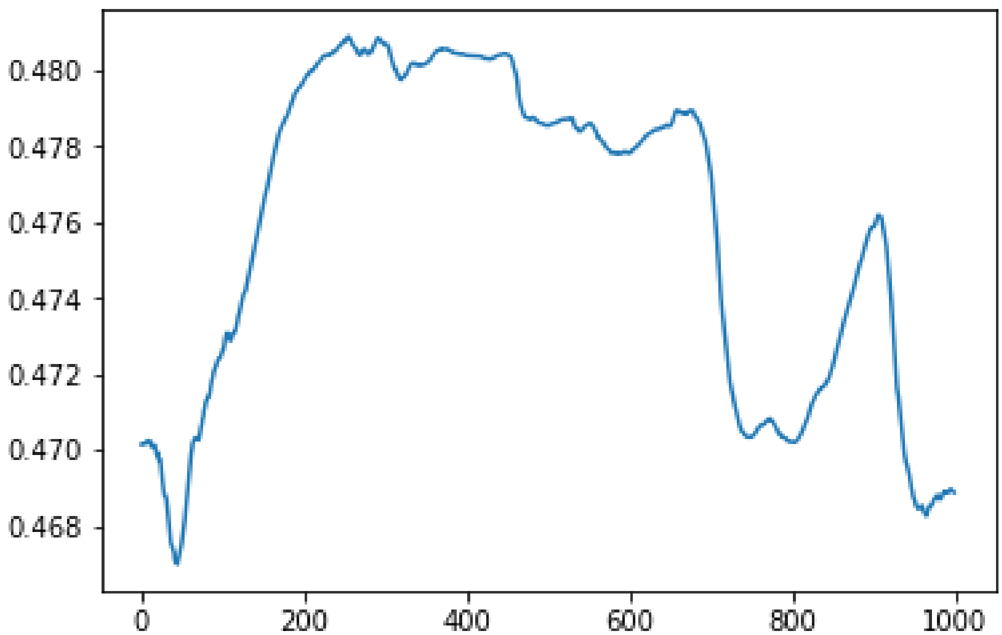
```
time lags -  15
```



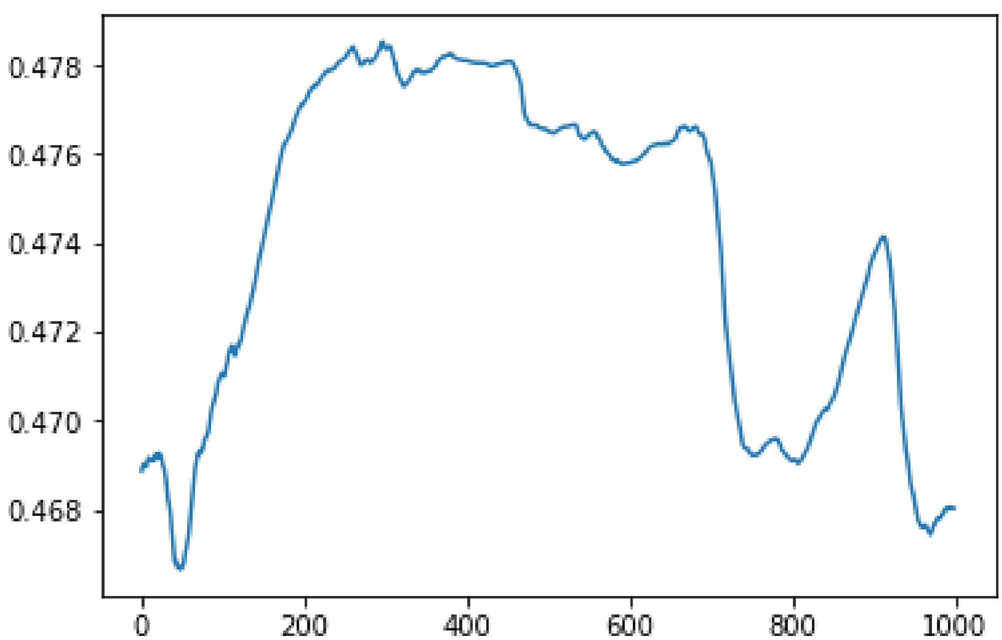

```
time lags -  20
```



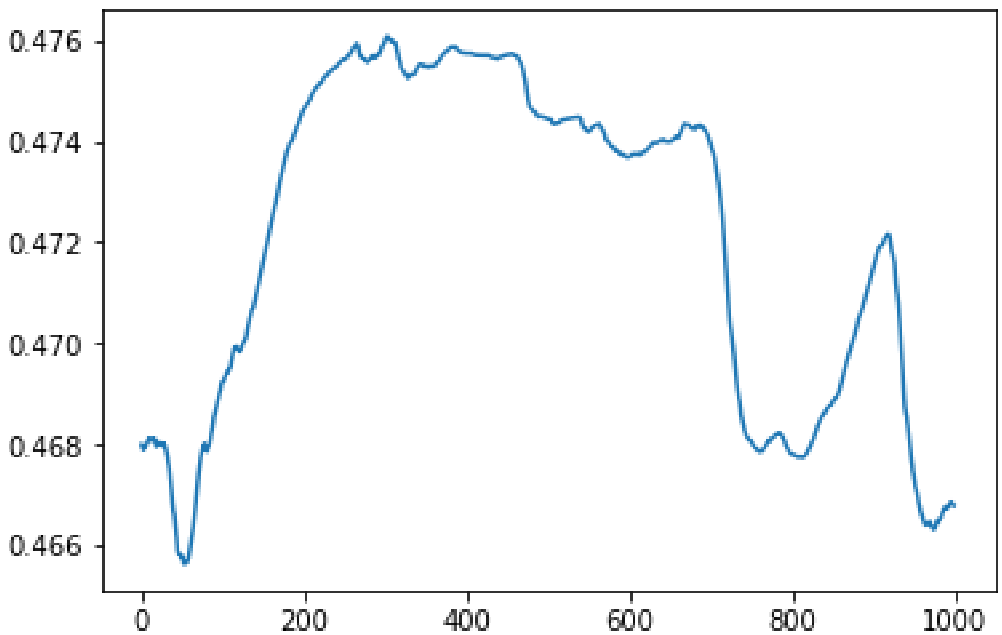
time lags - 25



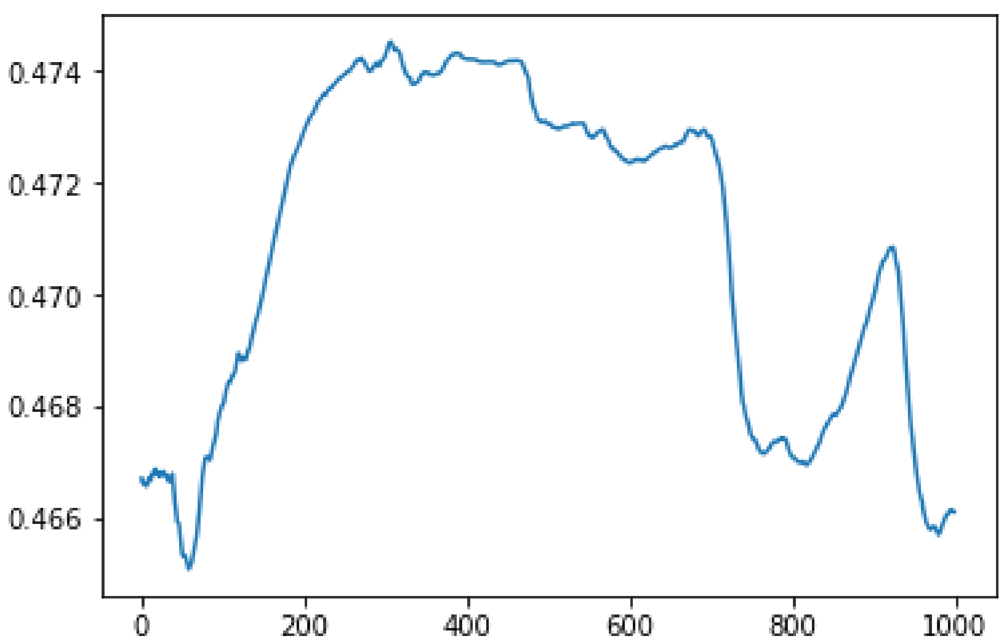
time lags - 30



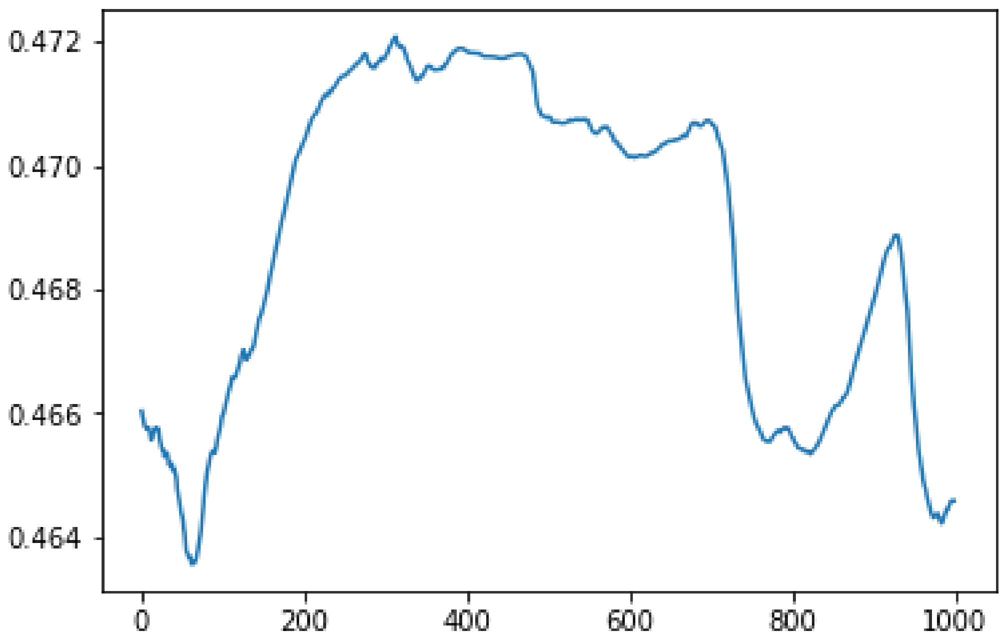
time lags - 35



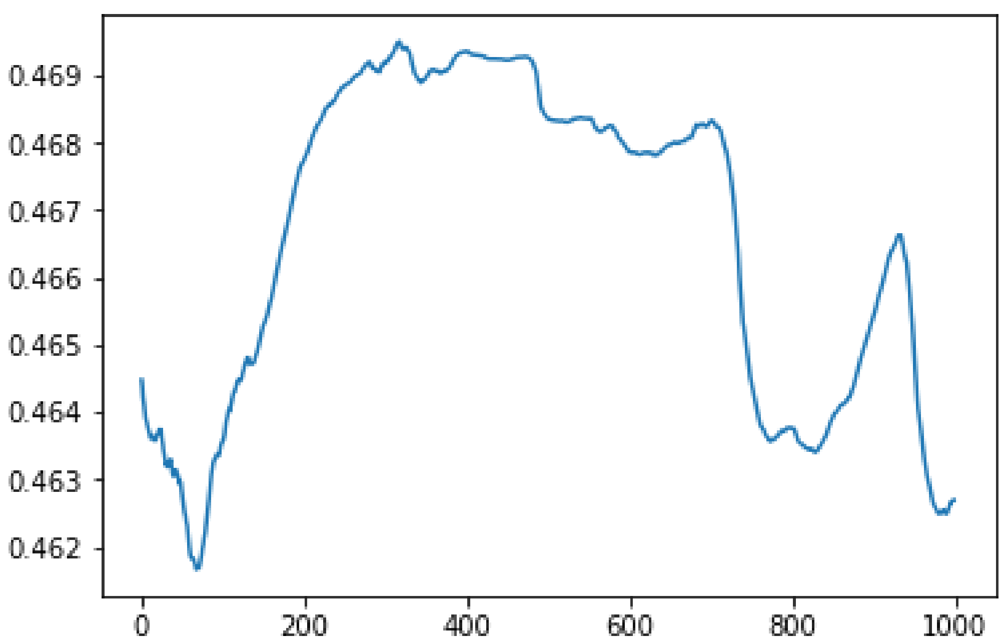
time lags - 40



```
time lags -  45
```



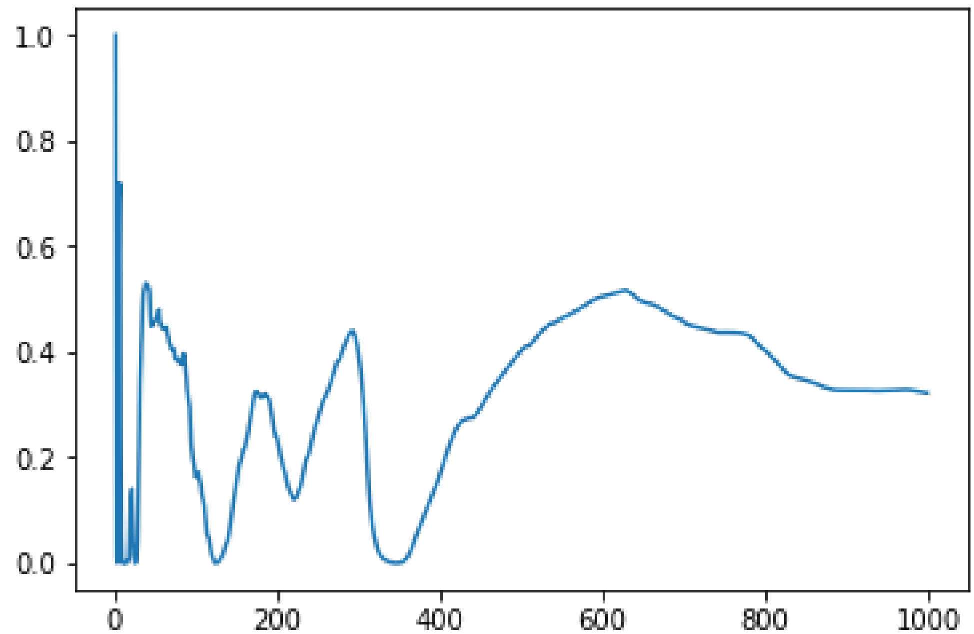

```
time lags -  50
```



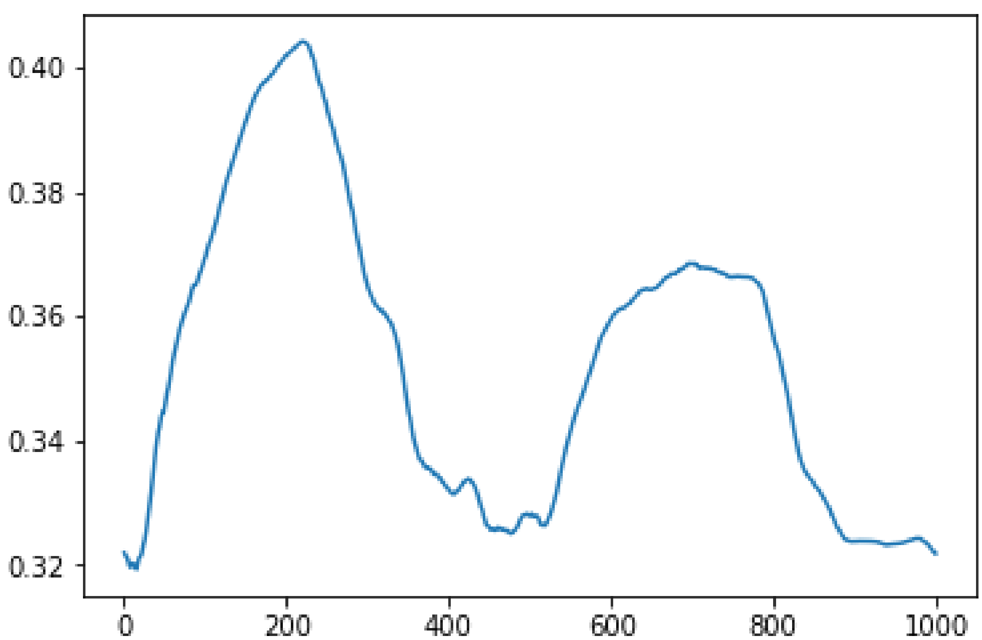
Now consider the case where $n = 1, ...10$ and t ϵ $[1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$

```
simn = 1000
outp = []
time = []
R_sq = []
random.seed(2)
T = [1,5,10,15,20,25,30,35,40,45,50]
for n in range(1,10):
    #generate errors
    e=[]
    y=[]
    u=[]
    x=[]
    #initial y gen
    y.append(random.normalvariate(4,1))
    #initial x gen
    x.append(np.random.normal(10,1,n))
    for t in T:
        for isim in range(0,simn):
            #error e gen
            e.append(random.normalvariate(0,1))
            #error u gen
            u.append(np.random.normal(0,1,n))
        for isim in range(1,simn):
            #generate y
            y.append(y[isim-t]+e[isim])
            #generate x
            x.append(x[isim-t]+u[isim])
            x_ar = np.array(x).reshape(-1,n)
            ones = np.ones((len(x),1))
            x_ar = np.hstack((ones,x_ar))
            y_ar=np.array(y)
            model1 = sm.OLS(y_ar,x_ar).fit()
            outp.append(model1.rsquared)
            #time.append(t)
        print("number of parameters - ",n)
        print("time lags - ",t)
        R_sq.append(model1.rsquared)
        plt.plot(range(1,simn),outp)
        plt.show()
        outp=[]
```

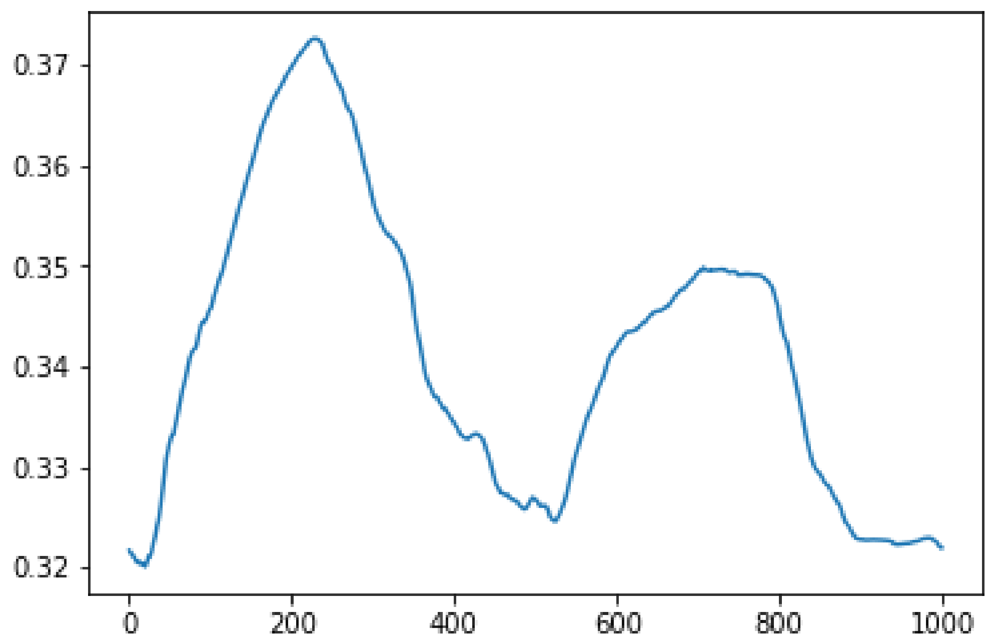
```
number of parameters -  1
time lags -  1
```



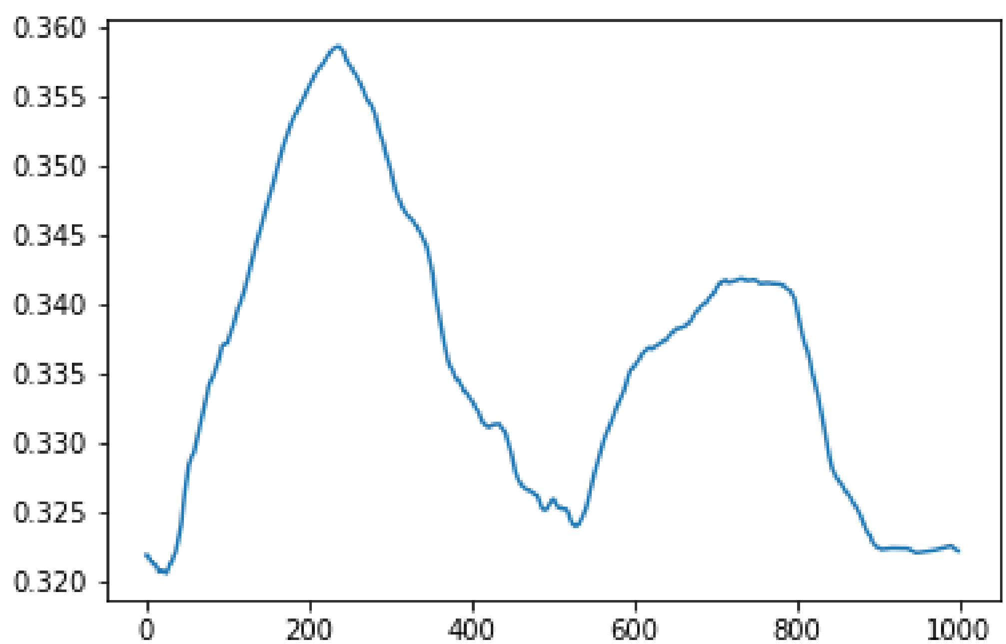

```
number of parameters -  1
time lags -  5
```



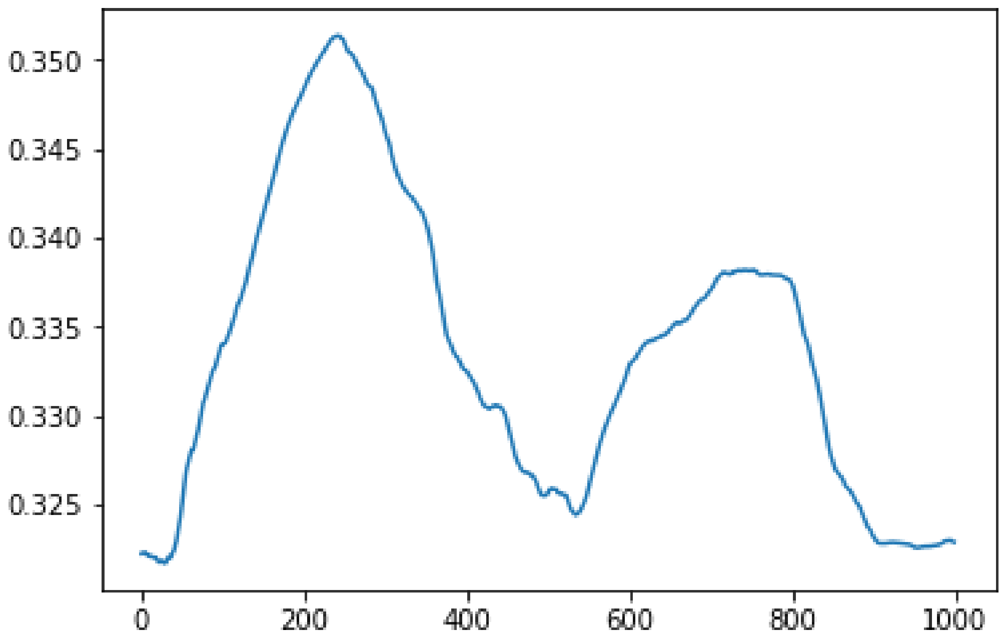
```
number of parameters -  1
time lags -  10
```



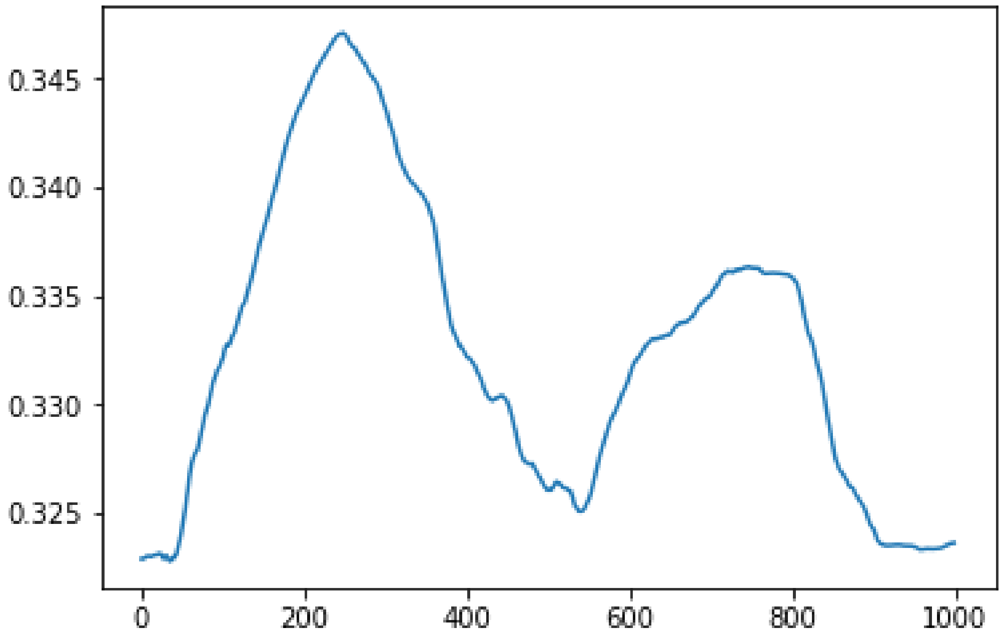

```
number of parameters -  1
time lags -  15
```



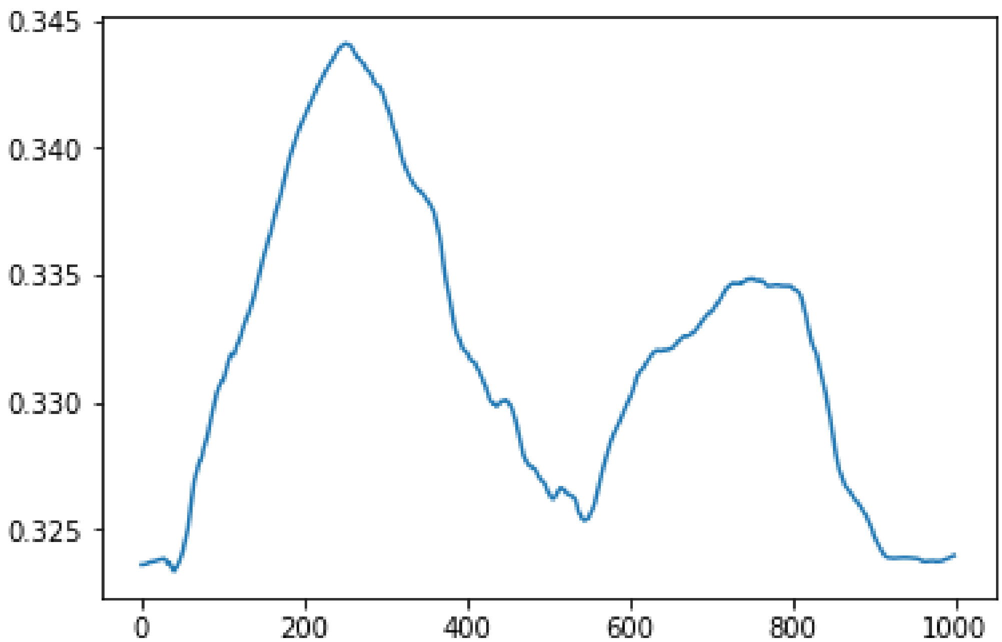
```
number of parameters -  1
time lags -  20
```



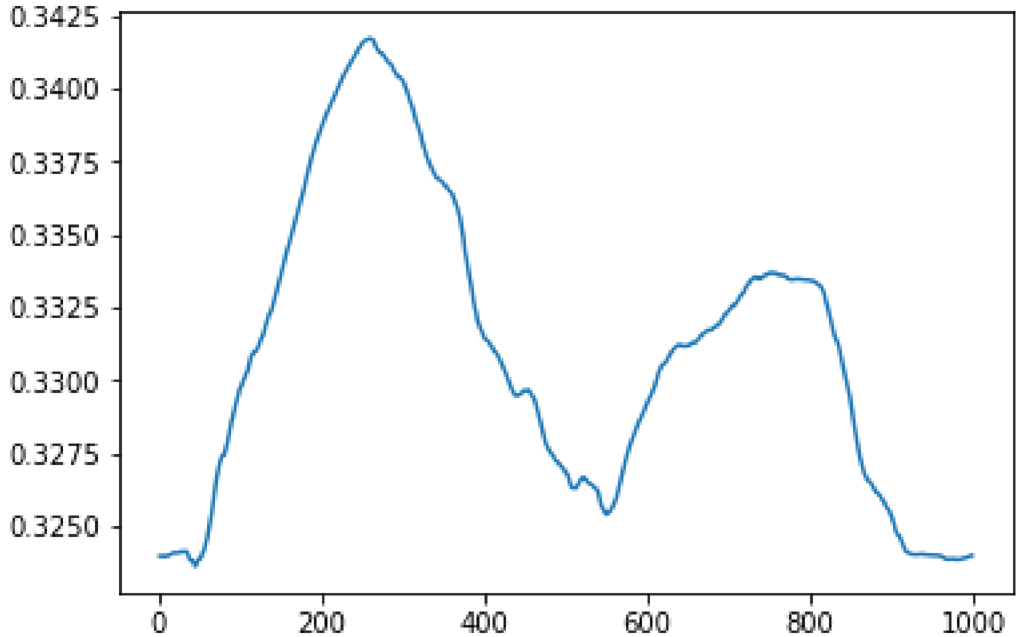

```
number of parameters -  1
time lags -  25
```



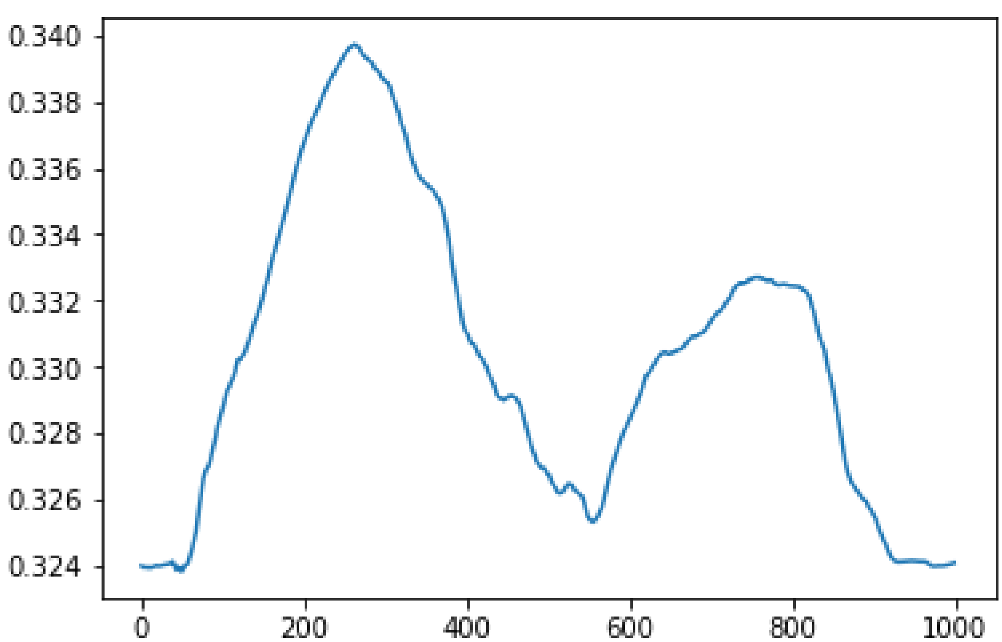

```
number of parameters -  1
time lags -  30
```



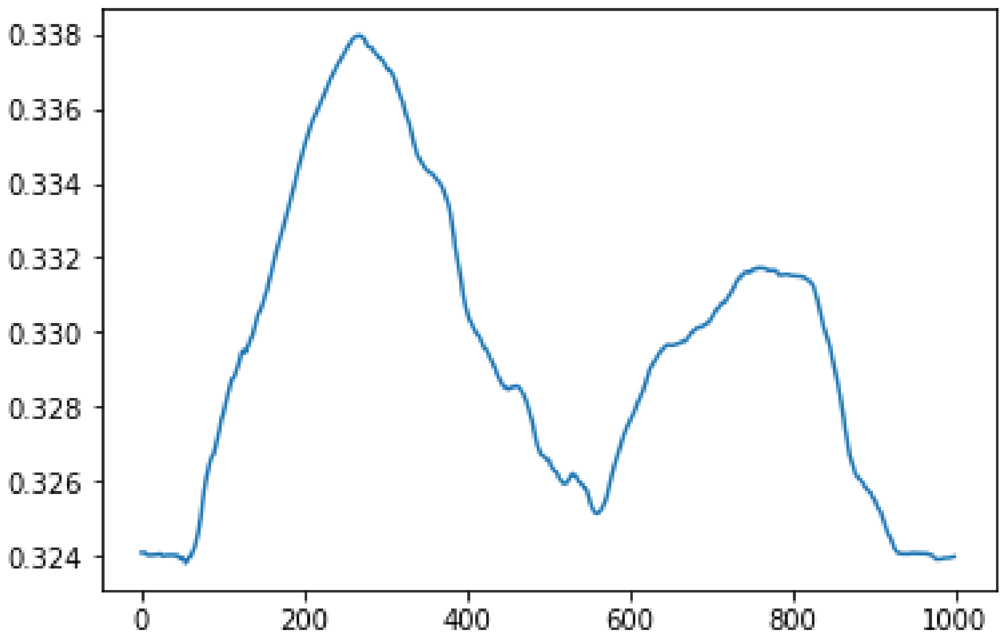
```
number of parameters -  1
time lags -  35
```



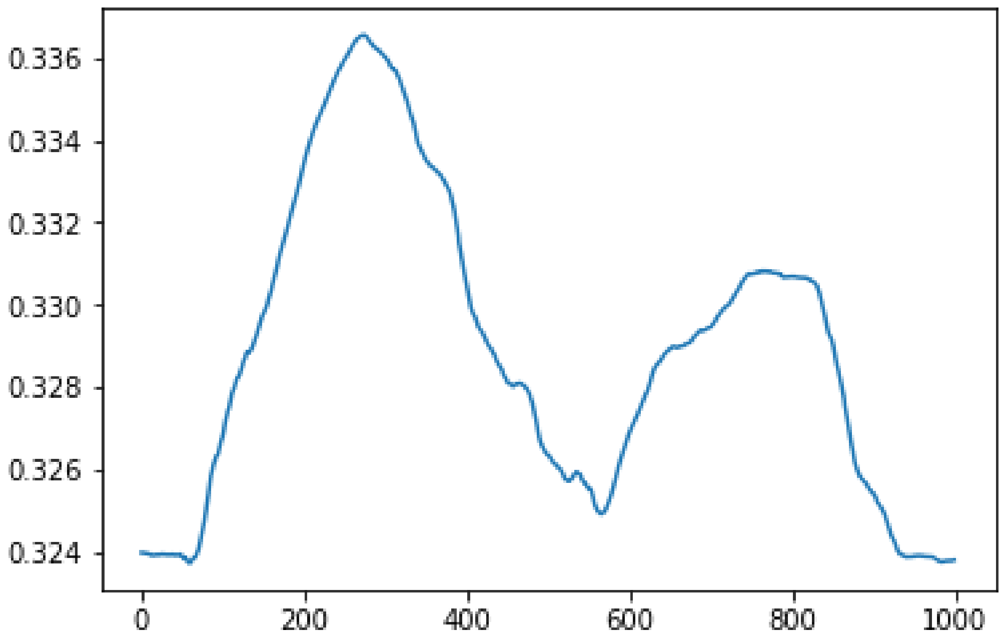

```
number of parameters -  1
time lags -  40
```



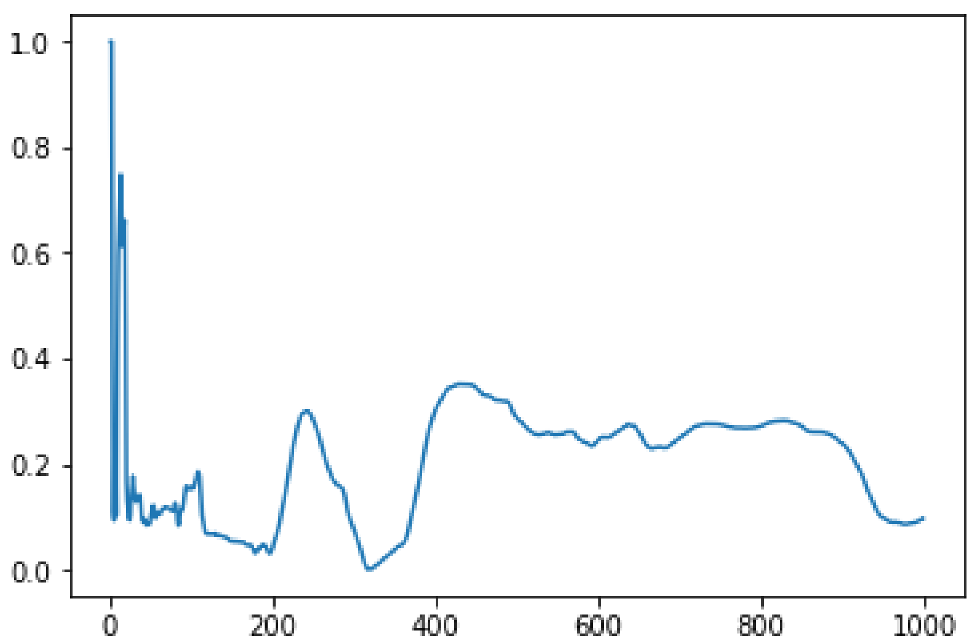
```
number of parameters -  1
time lags -  45
```



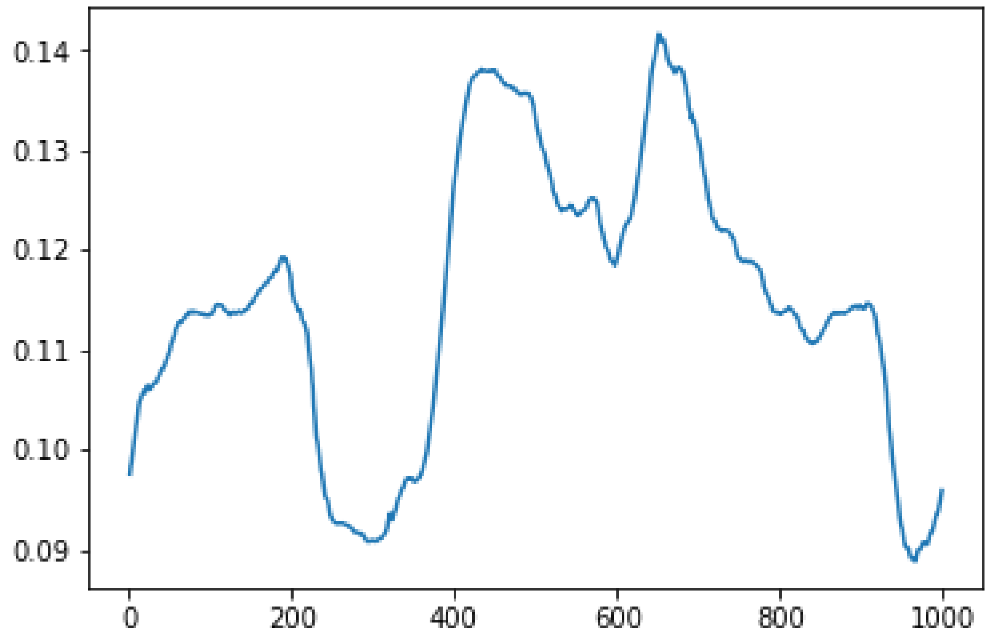

```
number of parameters -  1
time lags -  50
```



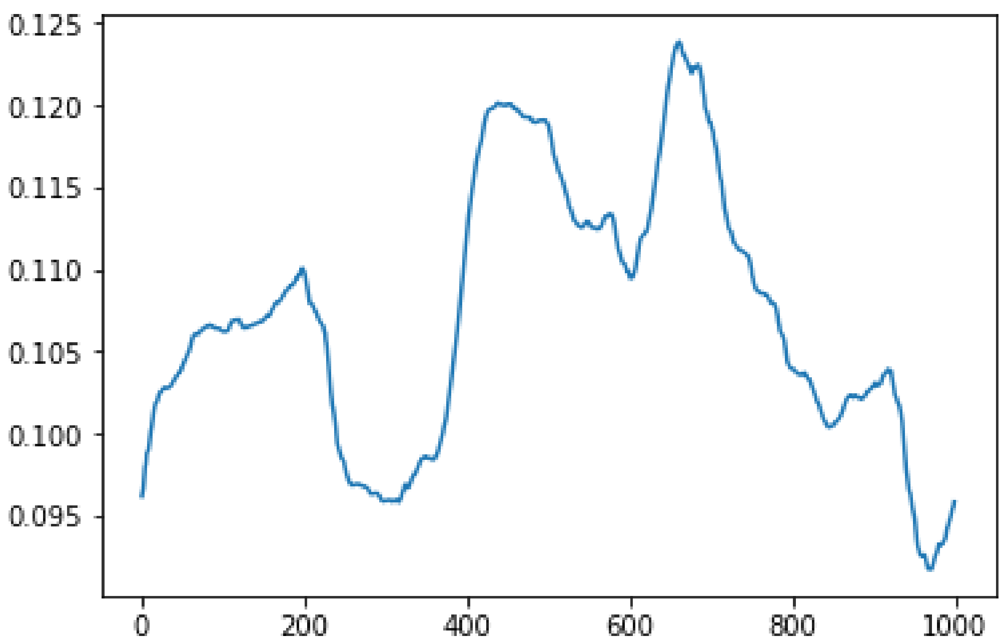

```
number of parameters -  2
time lags -  1
```



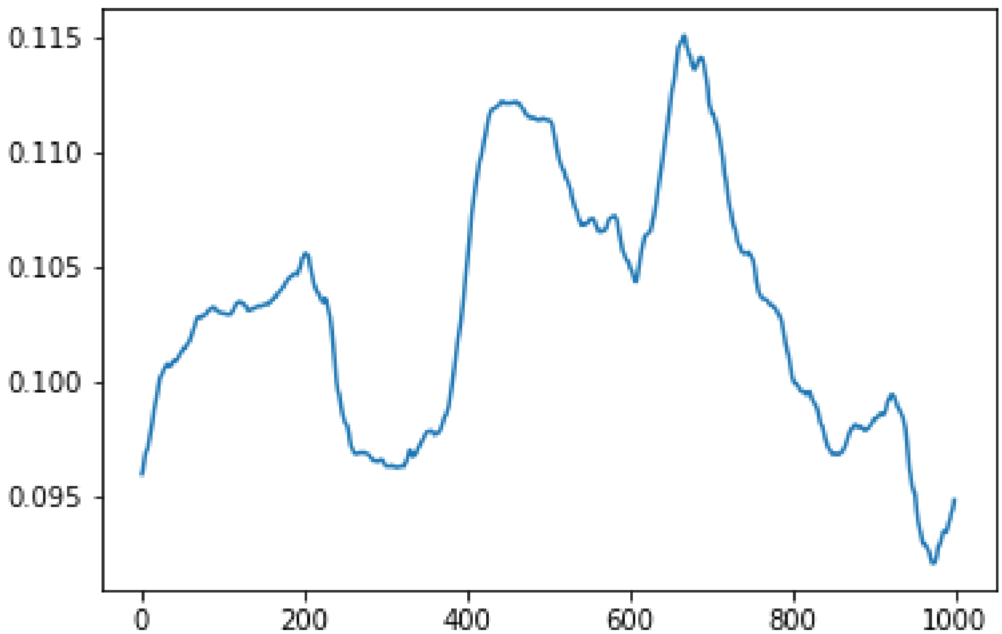
```
number of parameters -  2
time lags -  5
```



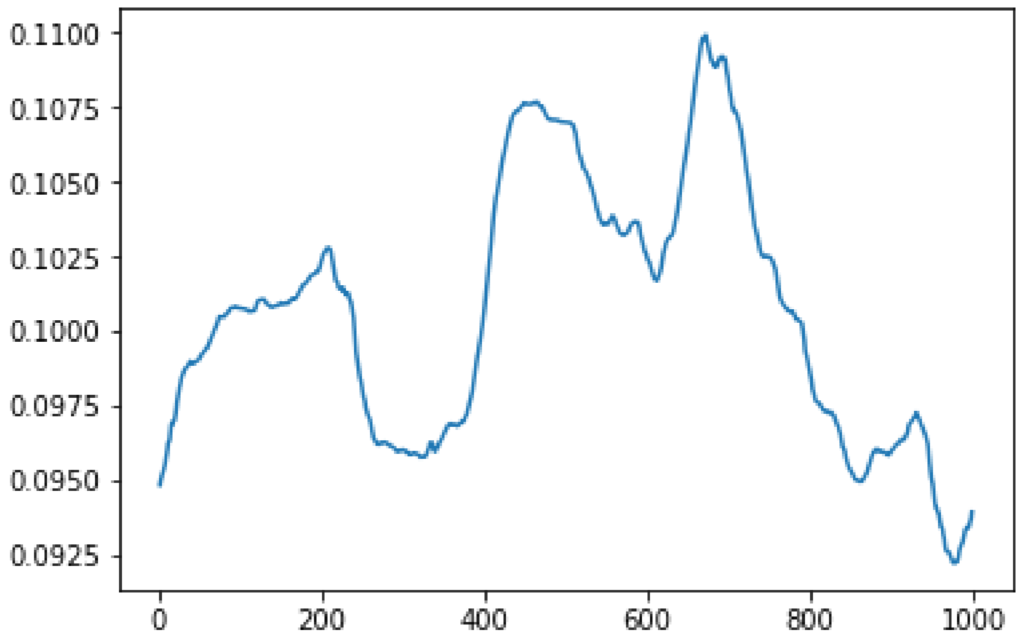

```
number of parameters -  2
time lags -  10
```



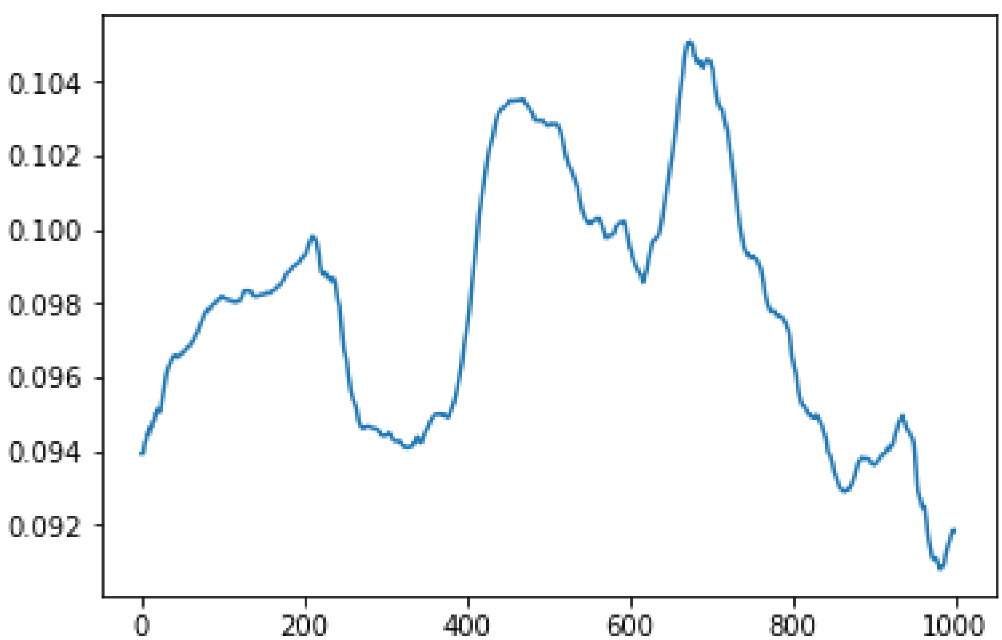
```
number of parameters -  2
time lags -  15
```



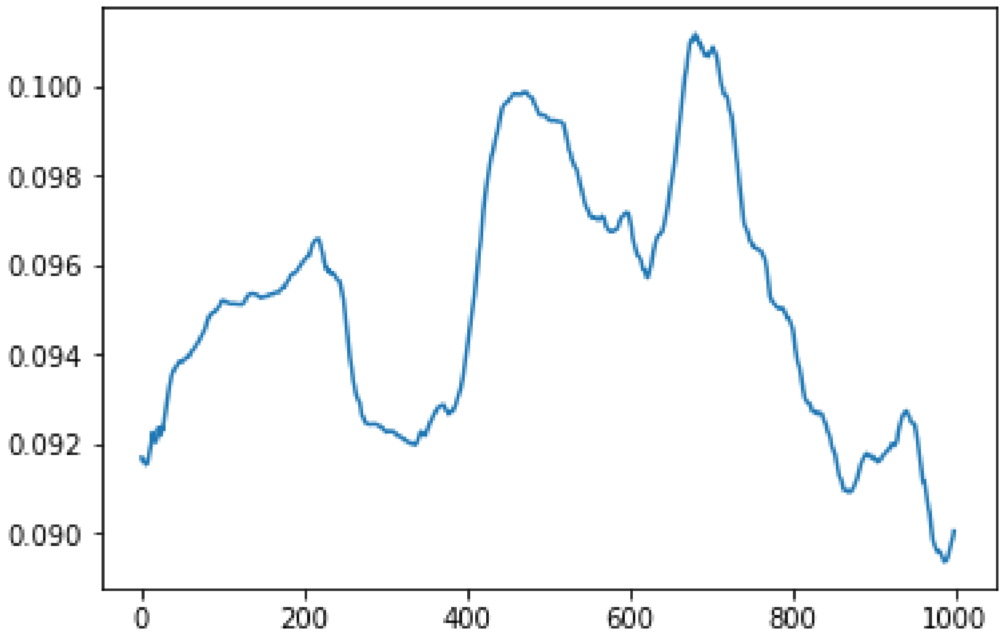

```
number of parameters -  2
time lags -  20
```



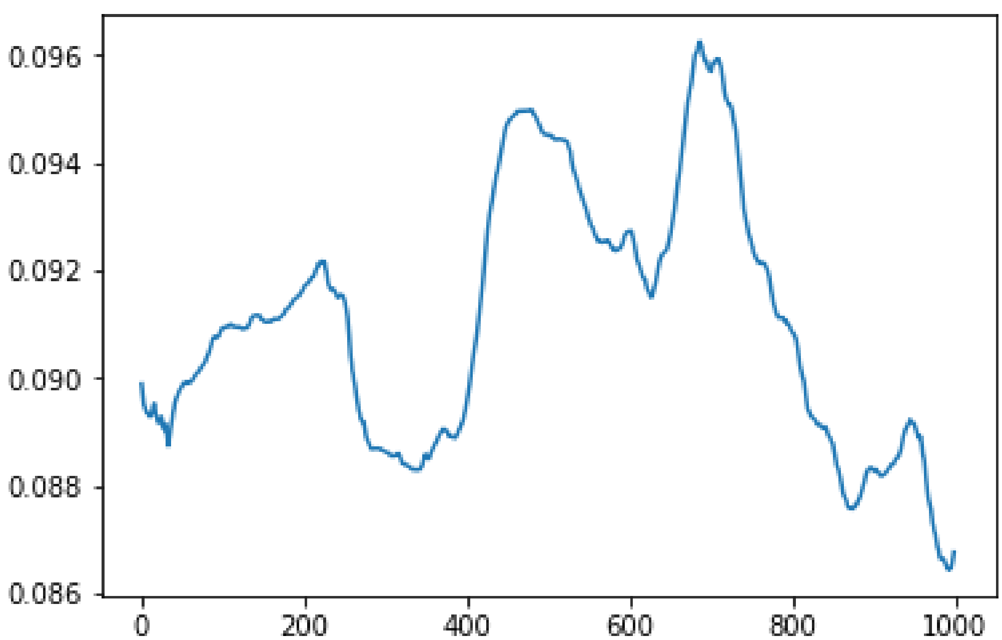

```
number of parameters -  2
time lags -  25
```



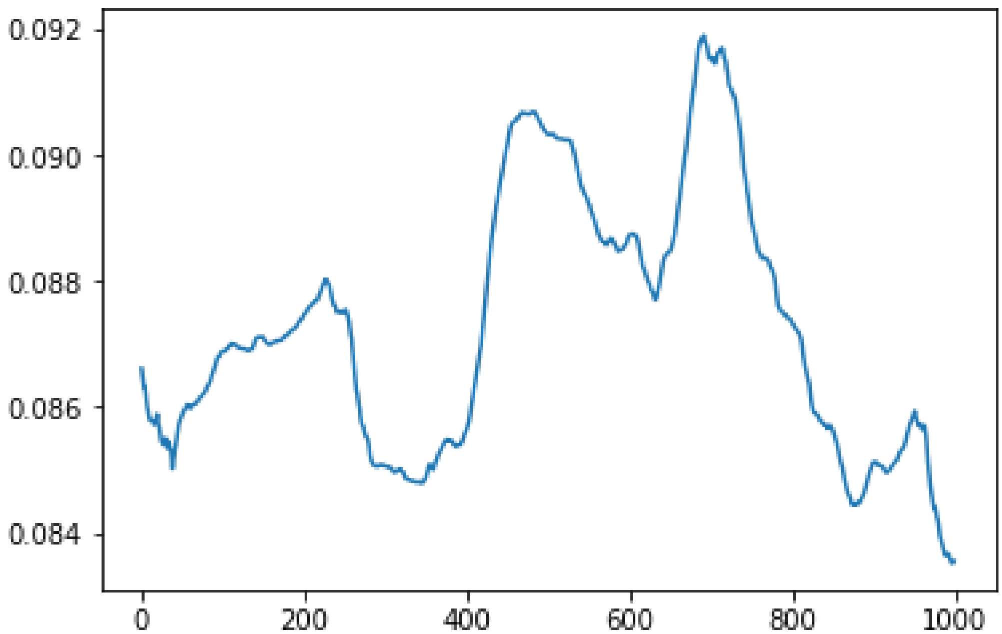
```
number of parameters -  2
time lags -  30
```



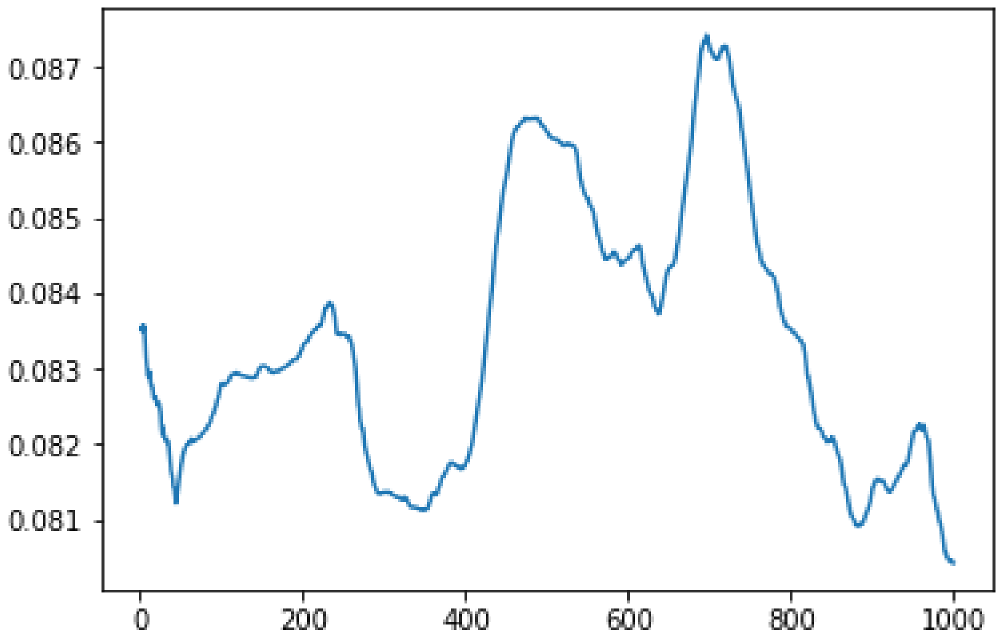

```
number of parameters -  2
time lags -  35
```



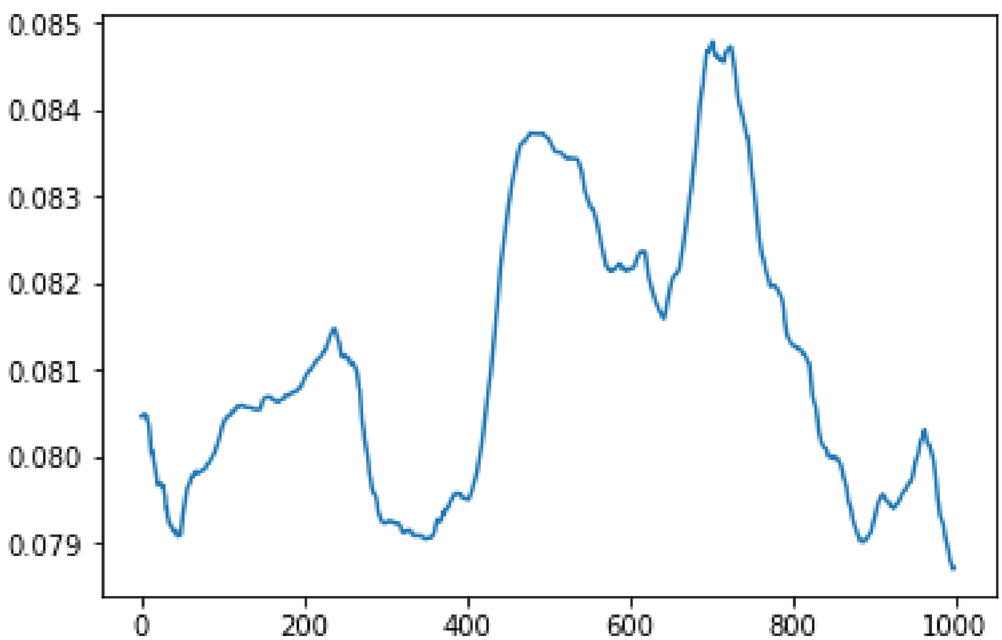
```
number of parameters -  2
time lags -  40
```



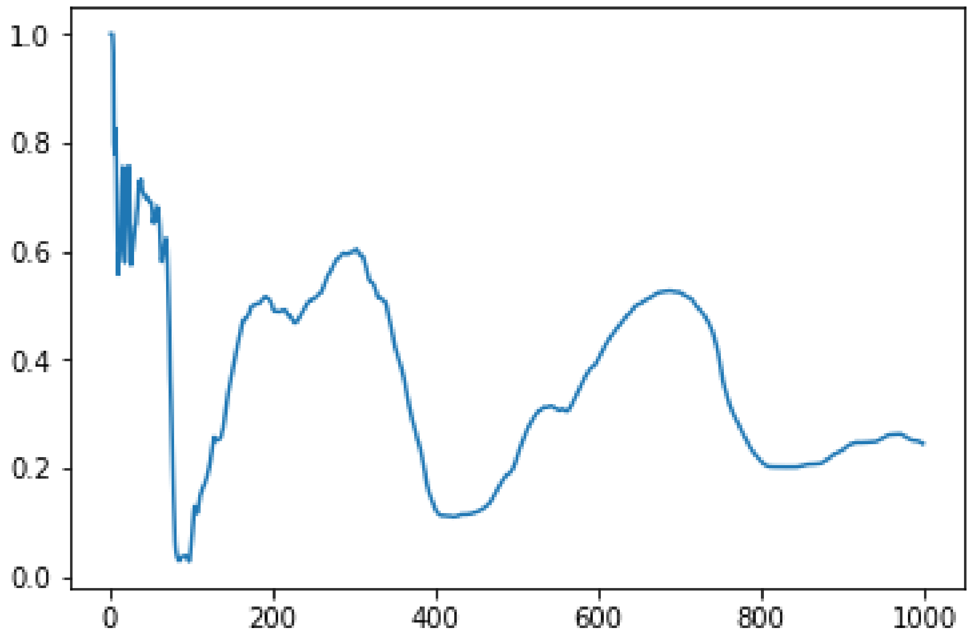

```
number of parameters -  2
time lags -  45
```



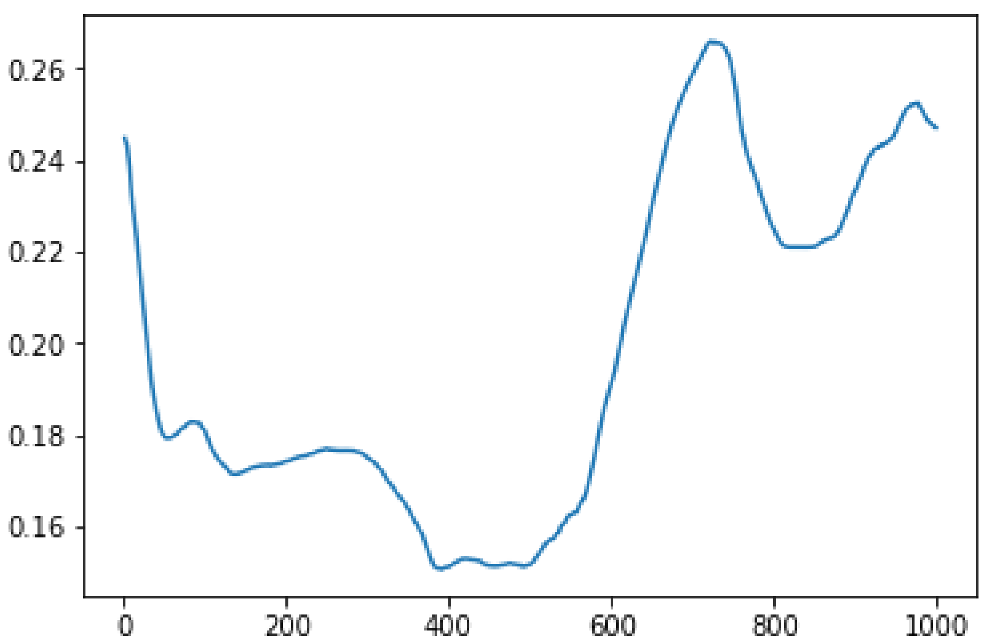

```
number of parameters -  2
time lags -  50
```



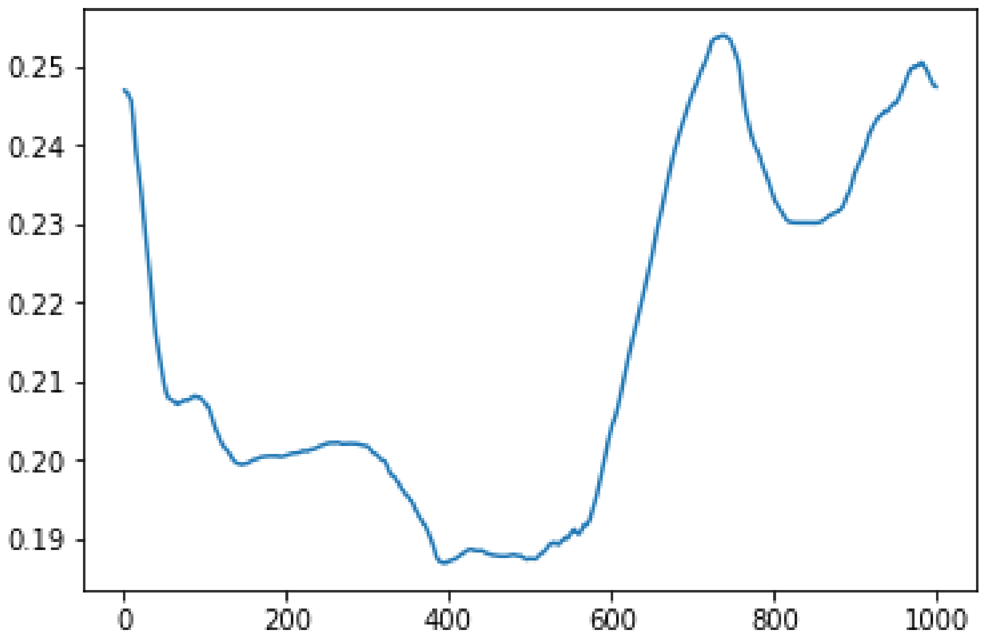
```
number of parameters -  3
time lags -  1
```



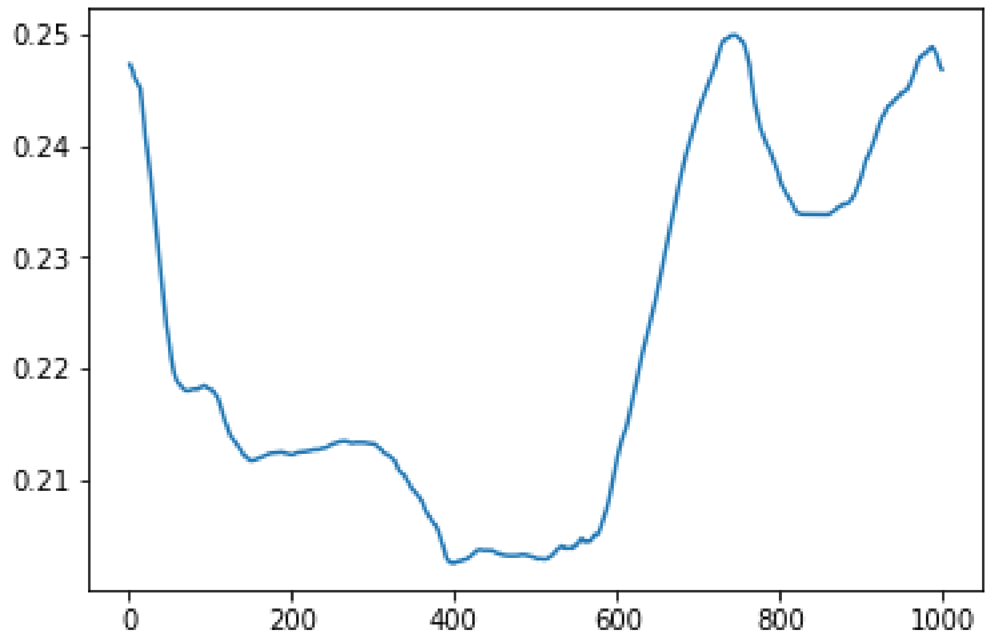

```
number of parameters -  3
time lags -  5
```



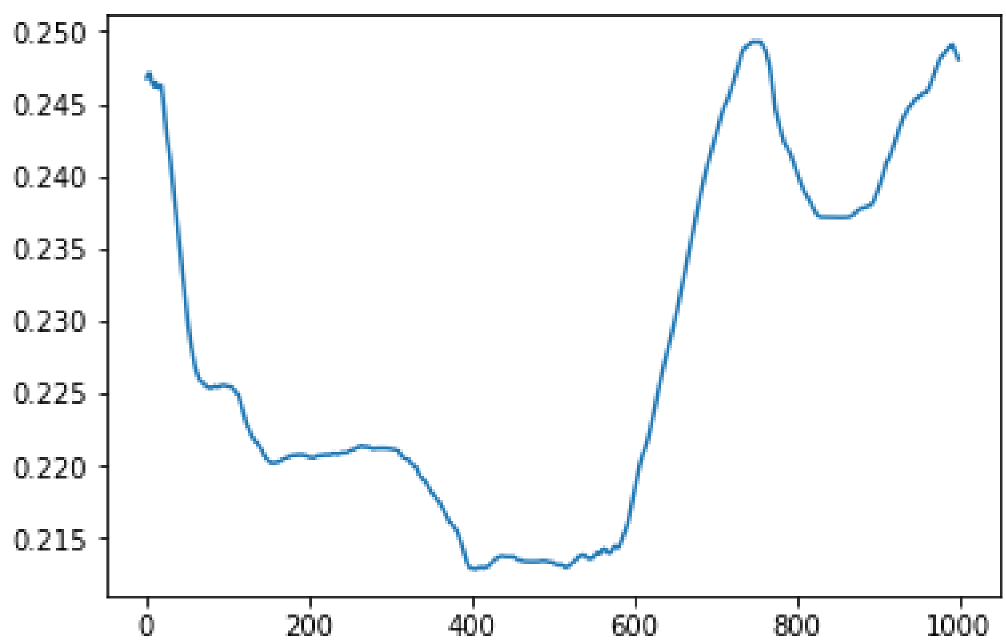
```
number of parameters -  3
time lags -  10
```



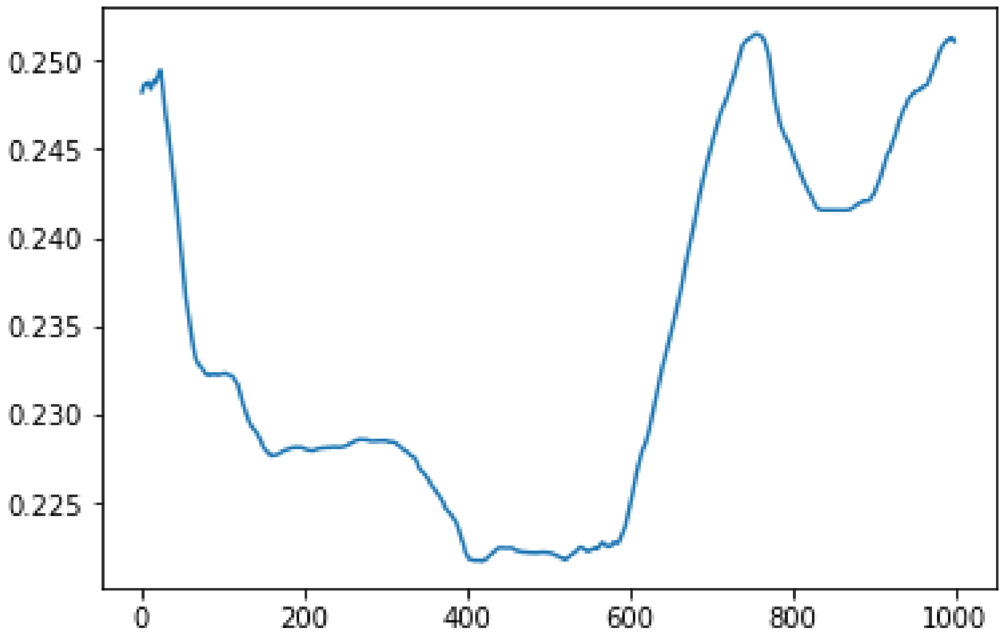

```
number of parameters -  3
time lags -  15
```



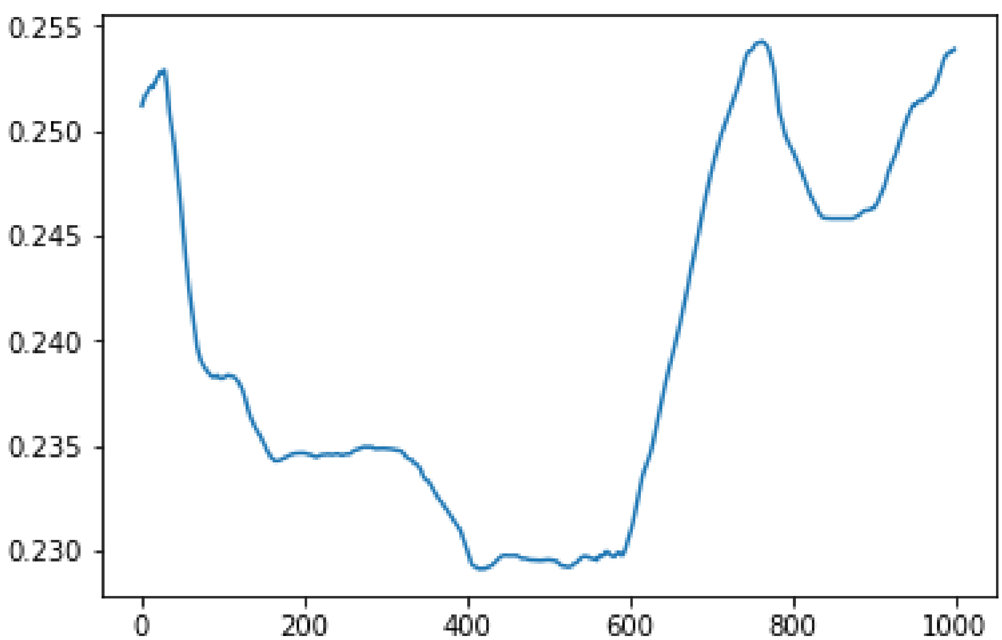

```
number of parameters -  3
time lags -  20
```



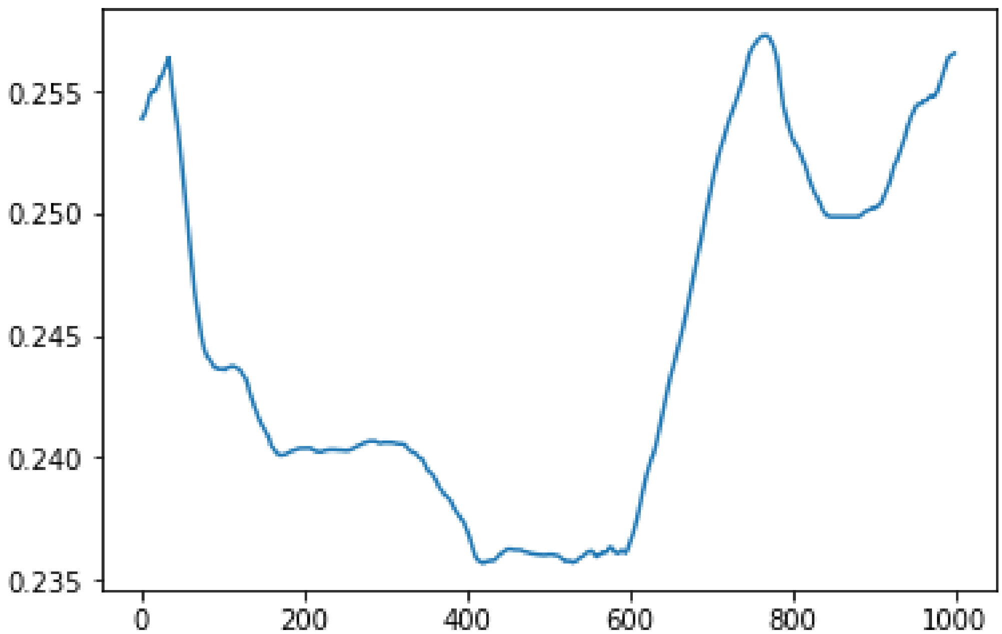
```
number of parameters -  3
time lags -  25
```



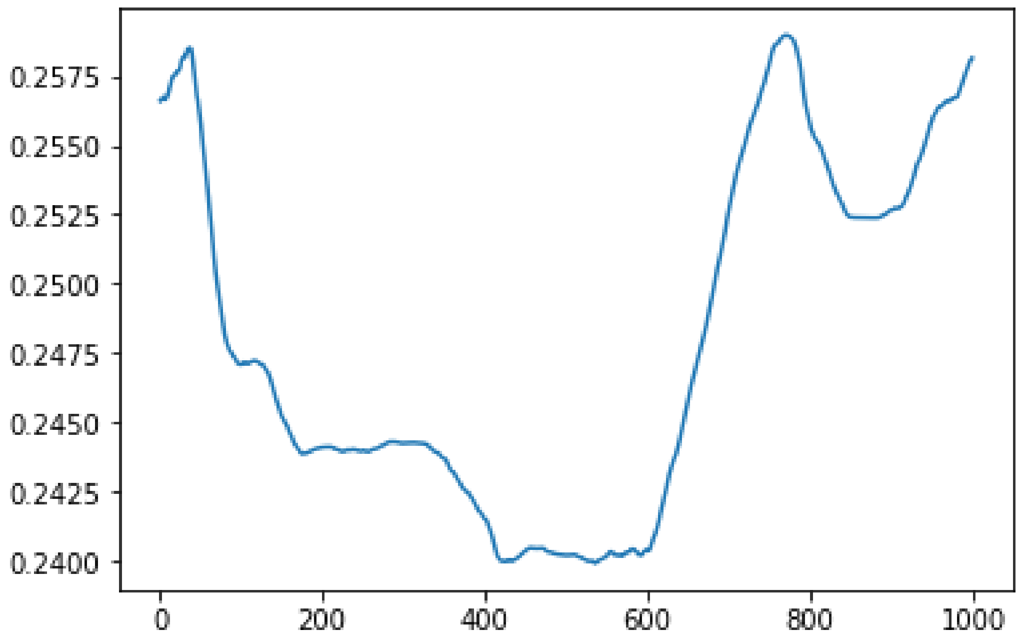

```
number of parameters -  3
time lags -  30
```



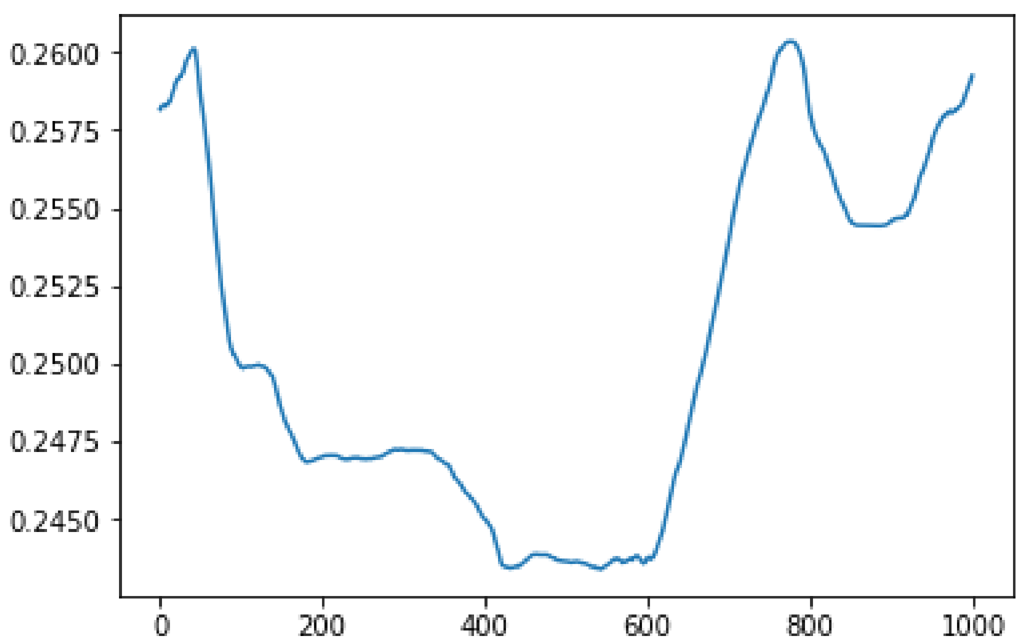
```
number of parameters -  3
time lags -  35
```



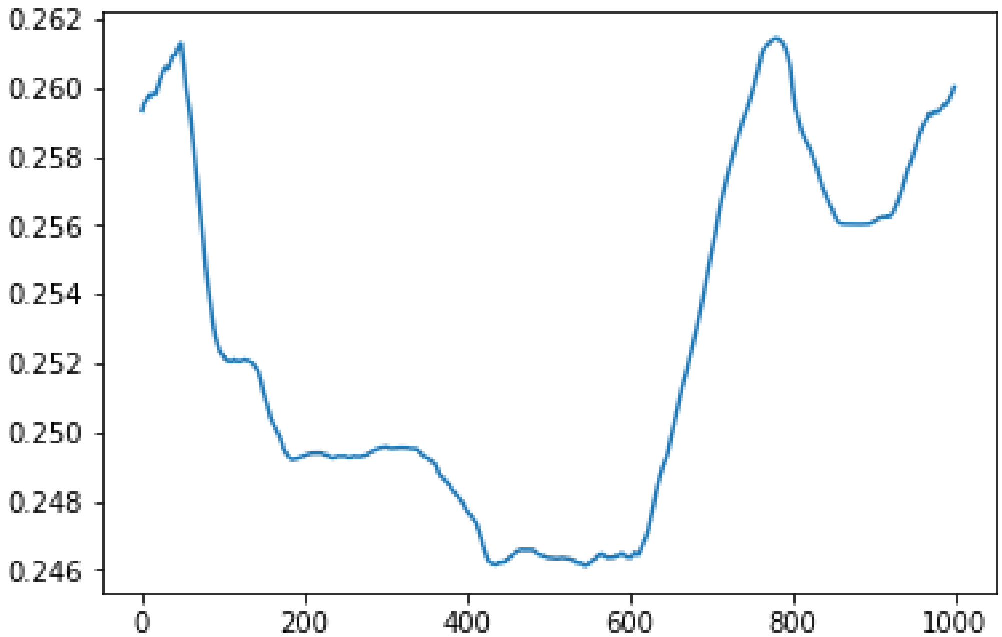

```
number of parameters -  3
time lags -  40
```



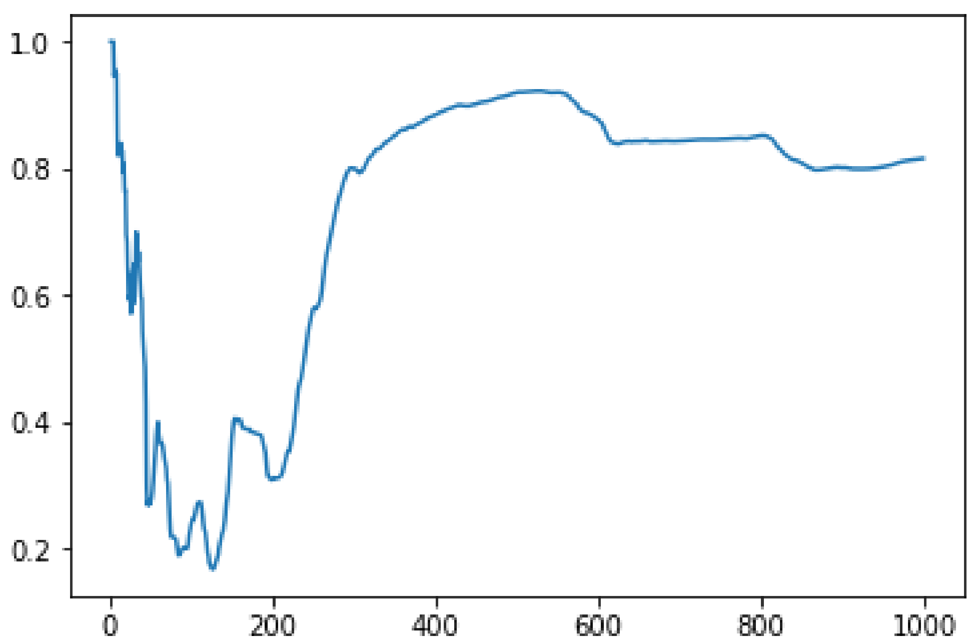

```
number of parameters -  3
time lags -  45
```



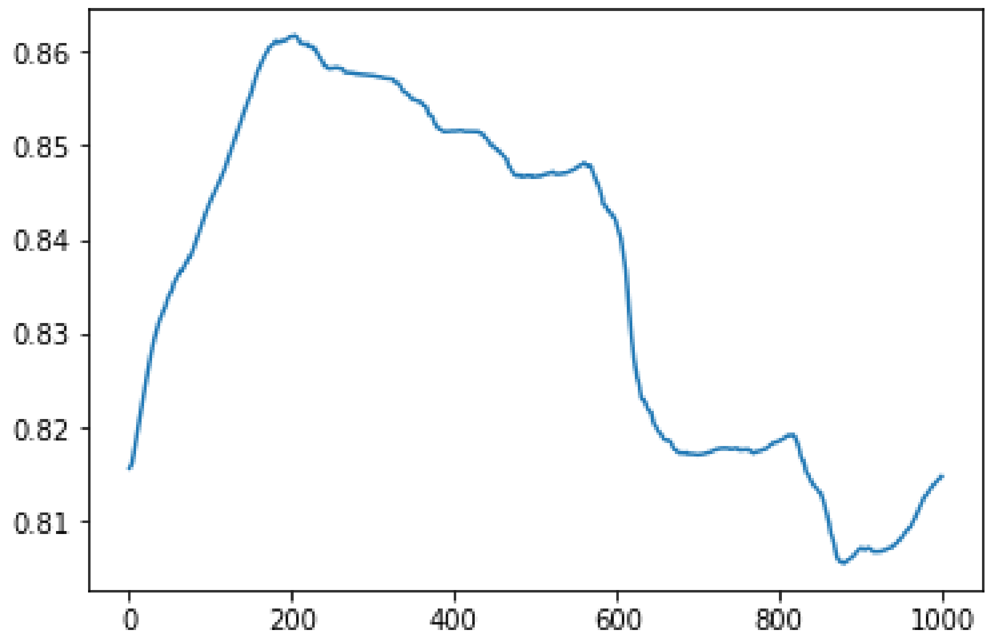
```
number of parameters -  3
time lags -  50
```



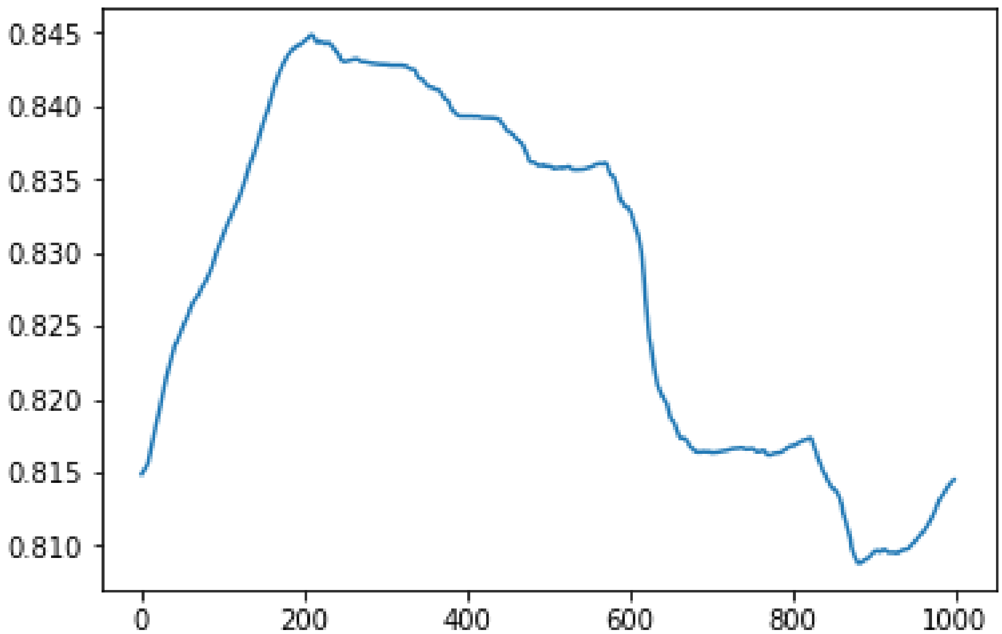

```
number of parameters -  4
time lags -  1
```



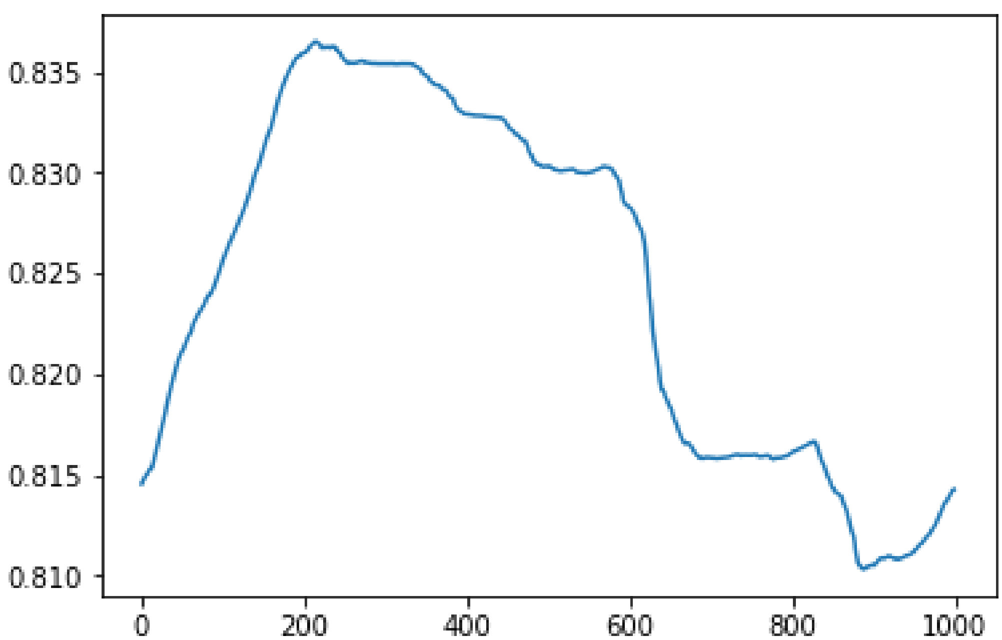
```
number of parameters -  4
time lags -  5
```



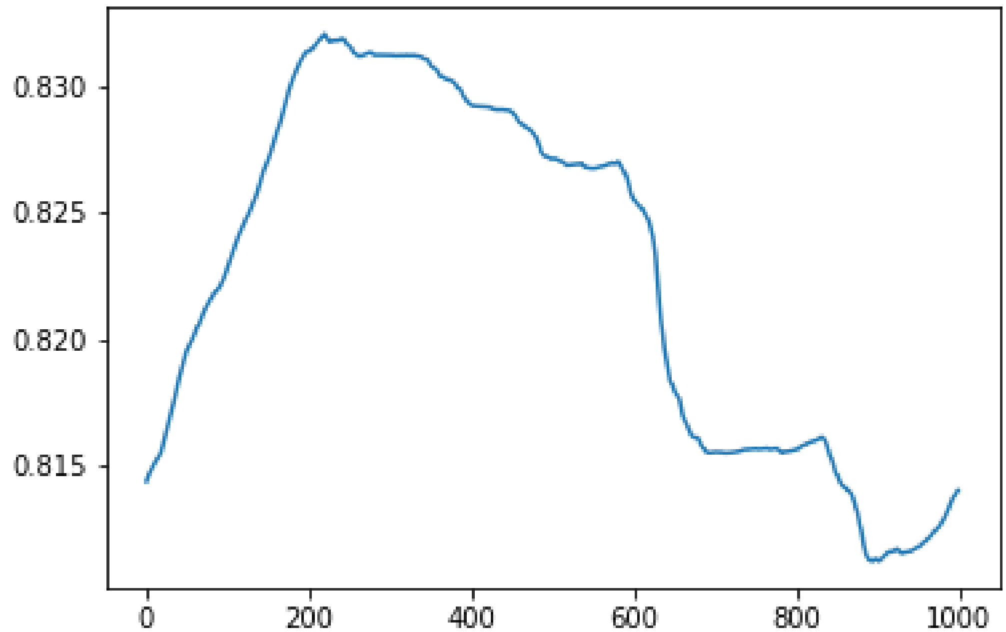

```
number of parameters -  4
time lags -  10
```



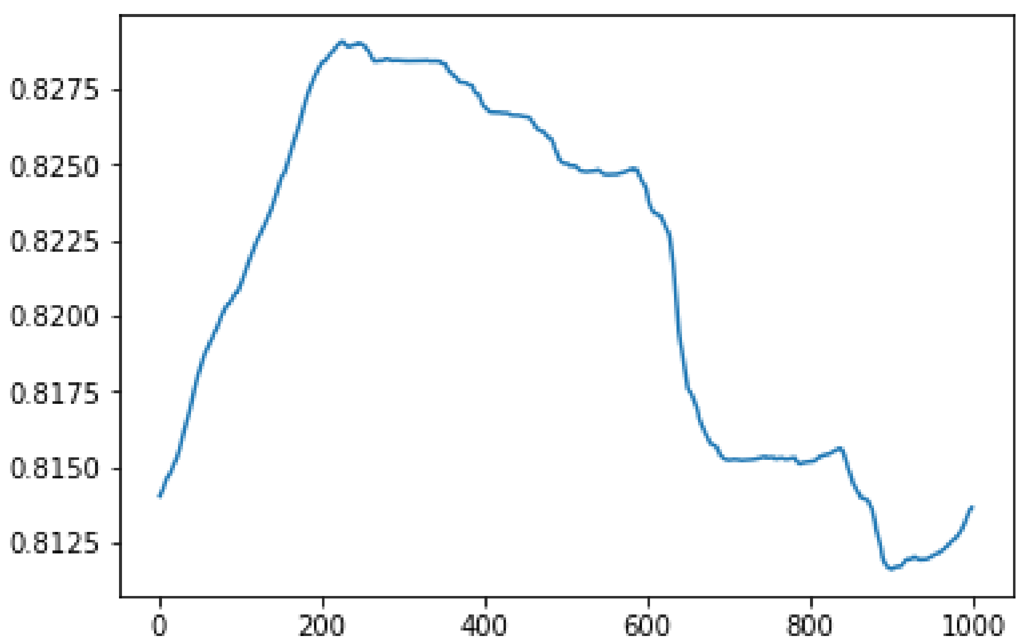

```
number of parameters -  4
time lags -  15
```



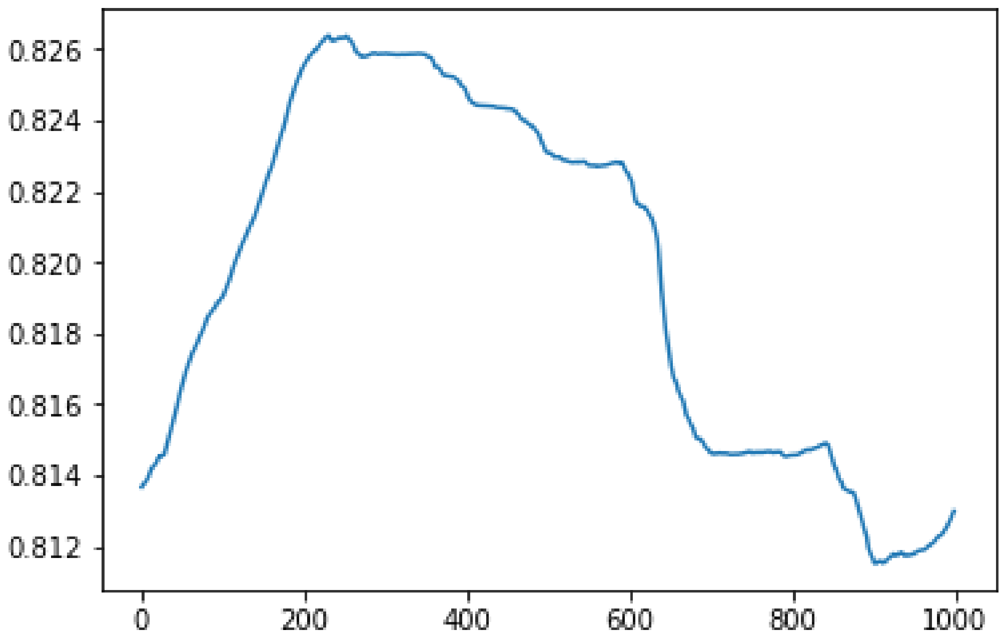
```
number of parameters -  4
time lags -  20
```



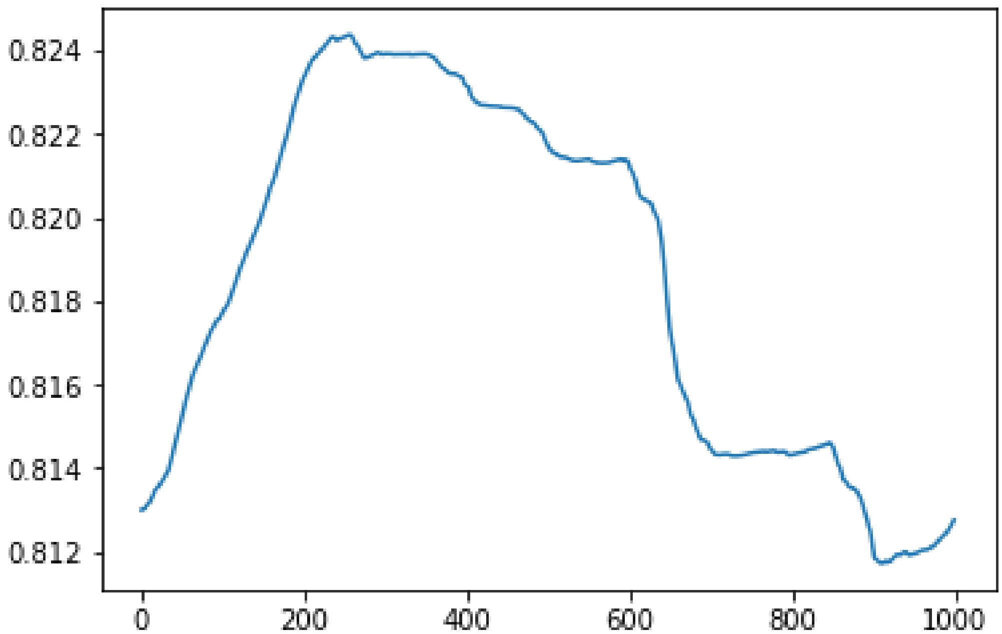

```
number of parameters -  4
time lags -  25
```



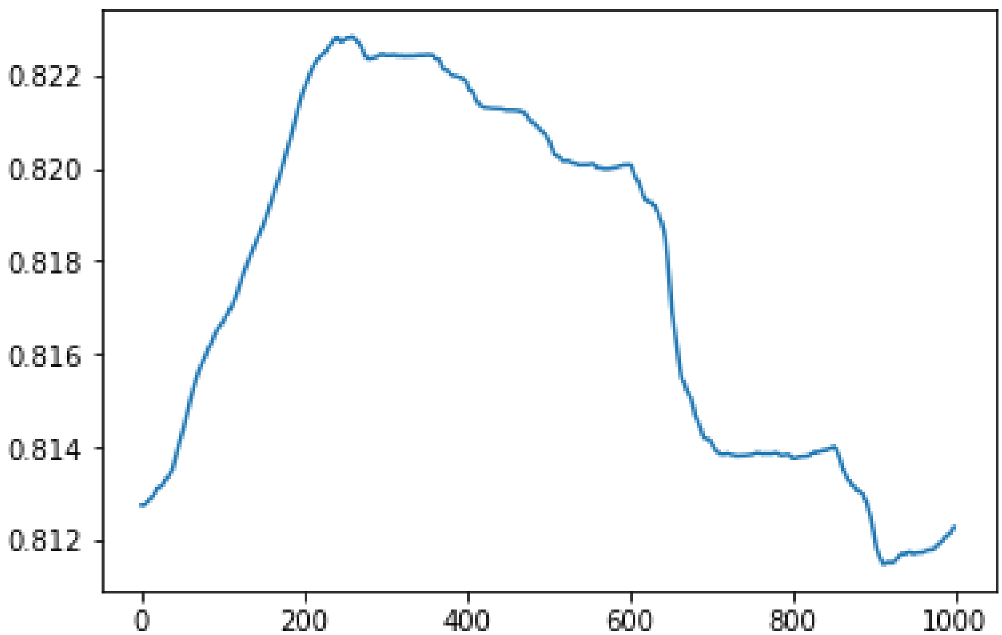
```
number of parameters -  4
time lags -  30
```



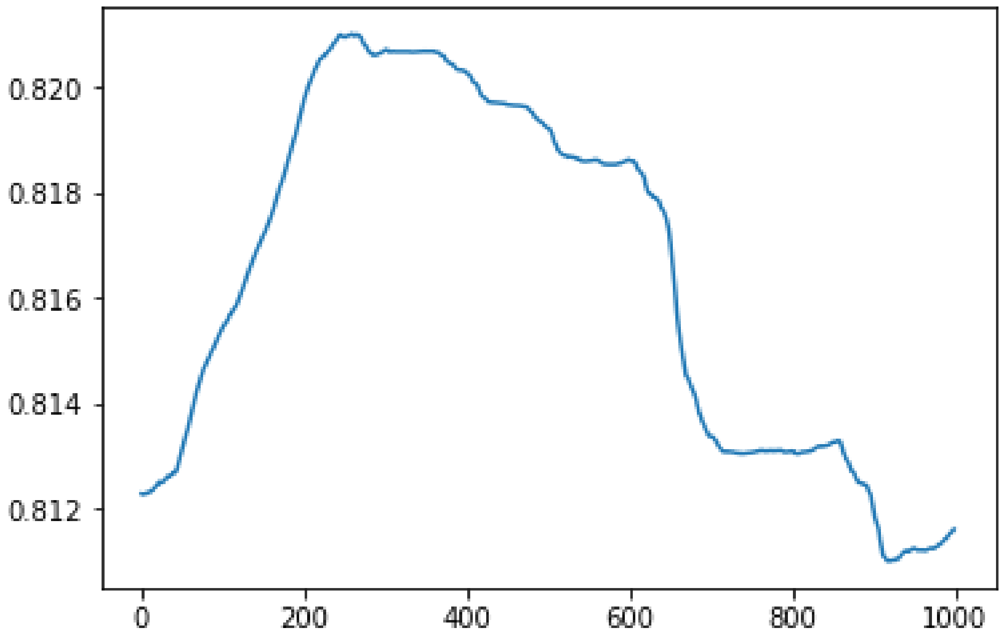

```
number of parameters -  4
time lags -  35
```



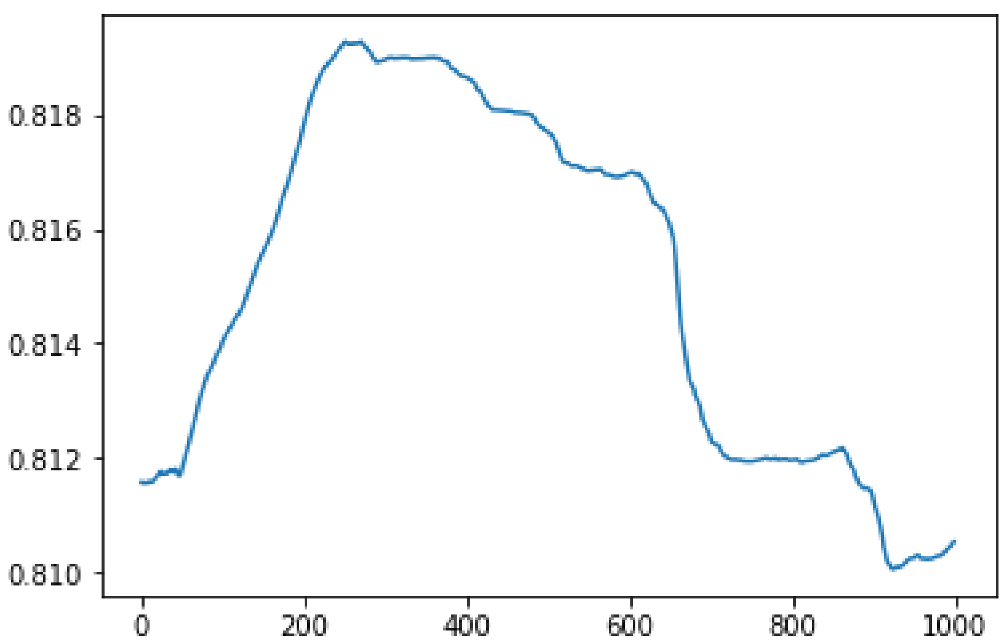

```
number of parameters -  4
time lags -  40
```



```
number of parameters -  4
time lags -  45
```




```
number of parameters -  4
time lags -  50
```



Similar plots are observed for higher order of t and N; where t = time - lags, N = number of parameters.

Conclusion We saw that, the OLS output of serially correlated data gives spurious results, the OLS estimators are no longer best estimators because the random sampling assumption is violated.

In coming posts, we will discuss on how to fix the serial correlation issue in OLS.