Autocorrelation

Signal Analysis

Vilnius University

Faculty of Mathematics and Informatics

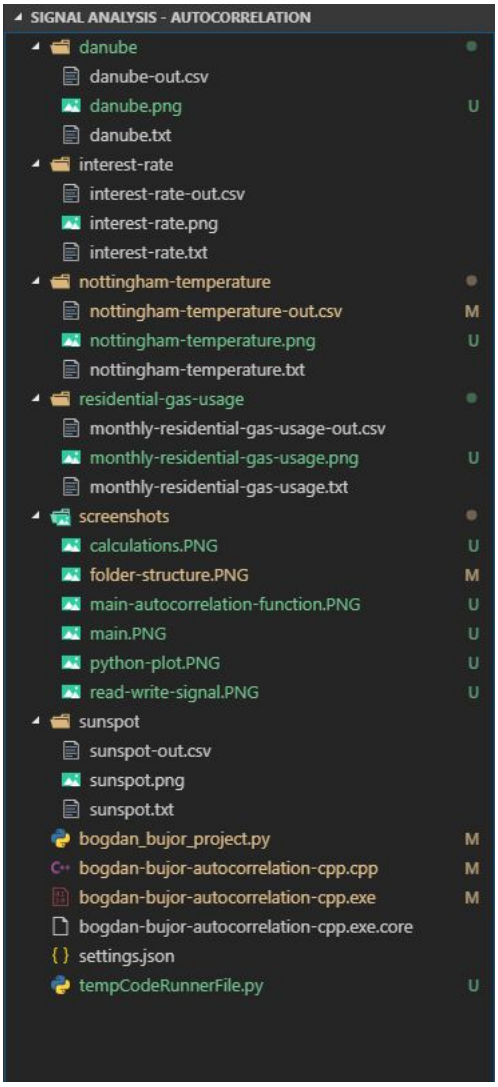
Vasile-Bogdan Bujor

May 8th 2020

Table of contents:

How the system works

1. Folder structure



This is how one will expect the project to be structured. Each subfolder of the main folder has three files inside:

- ➢ Raw data (.txt files)
- ➢ Outputted data from the app (.csv files)
- ➢ Photos with Python plotted data (.png files)

The files outside the subfolders are related to the programming itself (.cpp, .exe, .py, etc).

2. Why did I use both C++ and Python for this task?

C++ is the language I am most used for writing applications. But, like all other applications, it has its flaws. For example, C++ is not notoriously easy to use with data visualization libraries. Python, on the other hand, is perfect for this task. C++ and Python have quite a few advantages over other solutions

- ➢ C++ is way faster than Python, making it perfect for the overall computational part of the application;
- ➢ The ease of using Python (specifically with internal library *matplotlib*) makes it the perfect tool for visualization;
- ➢ Good for scripting and automatization.

Procedural programming style was used.

3. System analysis

3.1. C++

3.1.1. ReadSignal

```
void ReadSignal (double f[], int &n)
{
    ifstream txt;
    double readVar = 0;
    int i = 0;
    txt.open("C:\\Users\\Bogdan\\Documents\\Signal Analysis project\\.vscode\\Signal Analysis - Autocorrelation\\nottingham-temperature\\nottingham-temperature.txt");
    while(txt>>readVar)
    {
        f[i++] = readVar;
    }

    n = i;

    txt.close();
}
```

ReadSignal is the first function used in the .cpp file. Basic reading of files with hardcoded data path for faster project deployment.

3.1.2. WriteSignal

```
22  void WriteSignal (double r[], int N)
23  {
24      ofstream txt;
25      txt.open("C:\\Users\\Bogdan\\Documents\\Signal Analysis project\\.vscode\\Signal Analysis - Autocorrelation\\nottingham-temperature\\nottingham-temperature-out.csv")
26
27      for(int i = 0; i<= N/2; i++)
28          txt<<i+1<<","<<r[i]<<endl;
29
30      txt.close();
31  }
```

WriteSignal now loads another file for writing the results. The *r[]* array has all the necessary data needed for the Python script.

3.1.3. CalculateSmallFBarred and CalculateBigFBarred

```
33  double CalculateSmallFBarred(double f[], int N, int d)
34  {
35      double sum = 0;
36      for(int i = 0; i <= N-d; i++)
37      {
38          sum = sum + f[i];
39      }
40
41      return 1.0/(N - d + 1) * sum;
42  }
43
44  double CalculateBigFBarred(double f[], int N, int d)
45  {
46      double sum = 0;
47      for(int i = 0; i <= N-d; i++)
48      {
49          sum = sum + f[d + i];
50      }
51
52      return 1.0/(N - d + 1) * sum;
53  }
54
```

For calculating $\bar{f}$ and $\bar{F}$ I decided to write two functions, as that was the fastest approach and the one with the least confusion. A better approach would be to write one function and write an *if/else* decision tree at the $sum = sum + x$.

3.1.4. CalculateAutocorrelation

```
54
55  void CalculateAutocorrelation (double r[], double f[], int N)
56  {
57      int j = 0, d = 0;
58      double upperFraction = 0, lowerFraction = 0, finalFraction = 0; //write as upperFraction = Lower =.. =0
59      double leftLowerSum = 0, rightLowerSum = 0;
60      double smallFBarred = 0, bigFBarred = 0;
61      for(d = 0; d <= N/2; d++) //this is the big for, includes the whole fraction, d increments
62      {
63          finalFraction = 0; //resets every time d increases. d = 0, 1, ... [N/2]
64          upperFraction = 0;
65          lowerFraction = 0;
66          leftLowerSum = 0;
67          rightLowerSum = 0;
68
69          smallFBarred = CalculateSmallFBarred(f, N, d);
70          bigFBarred = CalculateBigFBarred(f, N, d);
71          //calculate upper fraction start
72          for(j = 0; j <= N-d; j++)
73          {
74              upperFraction = upperFraction + (f[j] - smallFBarred * (f[d+j] - bigFBarred));
75          }
76          //calculate upper fraction end
77          //calculate lower  fraction start
78          for(j = 0; j <= N-d; j++)
79          {
80              leftLowerSum = leftLowerSum + (f[j] * f[j] - 2 * (f[j] * smallFBarred) + smallFBarred * smallFBarred);
81              rightLowerSum = rightLowerSum + (f[d+j] * f[d+j] - 2 * (f[d+j] * bigFBarred) + bigFBarred * bigFBarred);
82              //(A-B)^2 = A^2 - 2*A*B + B^2
83              lowerFraction = sqrt(leftLowerSum * rightLowerSum);
84          }
85          //calculate lower fraction end
86
87          finalFraction = upperFraction / lowerFraction;
88
89          r[d] = finalFraction;
90          cout<<"d = "<<d<<", r[d] = "<<r[d]<<endl;
91      }
92  }
93
```

- Variable initialization;
- Resetting main variables every time *d* has been incremented;
- Calculating $\bar{f}$ and $\bar{F}$;
- Splitting the big fraction into smaller parts, *upperFraction* and *lowerFraction*
    - Further splitting *lowerFraction* into *leftLowerFraction* and *rightLowerFraction.* That way, the calculation was further simplified and made clear;
- Calculation of the whole fraction and setting the value of *r[d]* to it;
- Writing everything to console for easier tracking.

3.1.5. Main

```
95   int main(void)
96   {
97       double f[310] = {0};
98       int n = 0;
99       ReadSignal(f, n);
100      int const N = 309;
101      double r[N] = {0};
102      CalculateAutocorrelation(r, f, N);
103      WriteSignal(r, N);
104  }
```

The use of the main function in the .cpp file is basically only for app initialization. I preferred to use arrays over vectors because of application efficiency when ingesting known datasets. This comes with the obvious drawback that is, even though it's more memory efficient and faster to parse by the system, it has a fixed element length. Reading, calculating and writing the data needed.

That's everything for the .cpp file. How do we visualize the data? Welcome Python scripting.

3.2. Python

3.2.1. Data visualization

```
1   import matplotlib.pyplot as plt
2   import csv
3
4   x = []
5   y = []
6
7   with open('C:\\Users\\Bogdan\\Documents\\Signal Analysis project\\.vscode\\' +
8       'Signal Analysis - Autocorrelation\\interest-rate\\interest-rate-out.csv','r') as csvfile:
9       plots = csv.reader(csvfile, delimiter=',')
10      for row in plots:
11          x.append(int(row[0]))
12          y.append(float(row[1]))
13
14  fo = open('C:\\Users\\Bogdan\\Documents\\Signal Analysis project\\.vscode\\' +
15      'Signal Analysis - Autocorrelation\\interest-rate\\interest-rate-out.csv', 'r')
16  content = fo.readlines()
17  print content
18
19  plt.plot(x,y)
20  plt.xlabel('Lag')
21  plt.ylabel('Autocorrelation')
22  plt.title('Autocorrelation of monthly Interest rates\nReserve Bank of Australia\nJan 1969 - Sep 1994')
23  plt.legend()
24  plt.show()
```

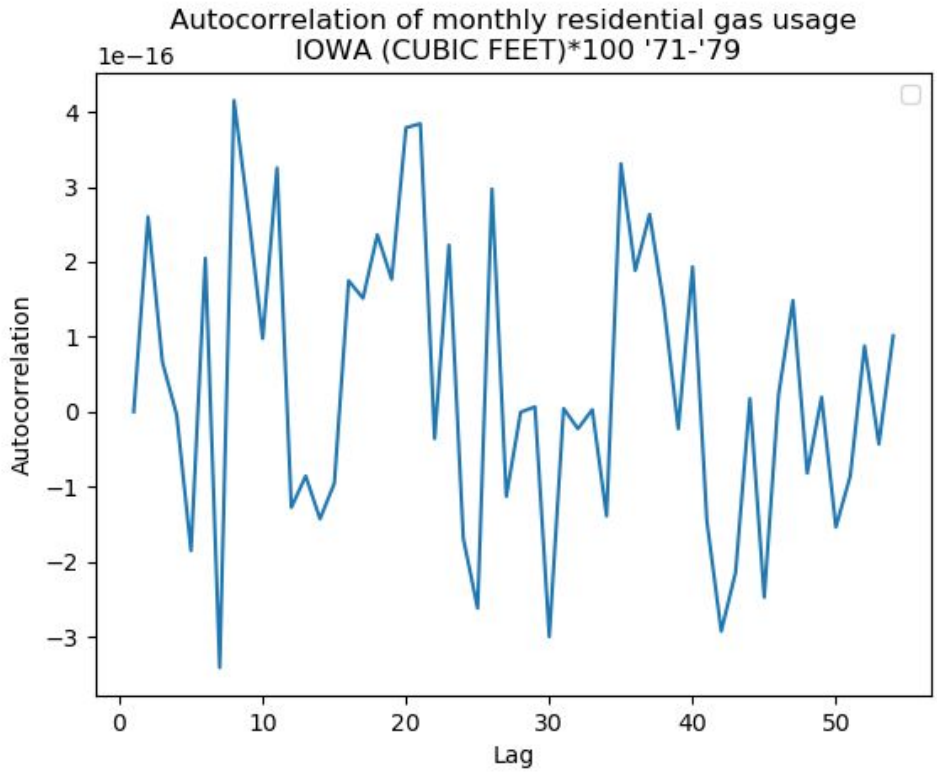
- ➢ Importing necessary libraries (only *matplotlib* and *csv*);
- ➢ Declaring arrays in order to use them as axis for our plots;
- ➢ Read data outputted earlier from C++;
- ➢ Writing data to console for tracking purposes;
- ➢ Writing labels and plotting the data into a graph;

4. Scripting and reusability

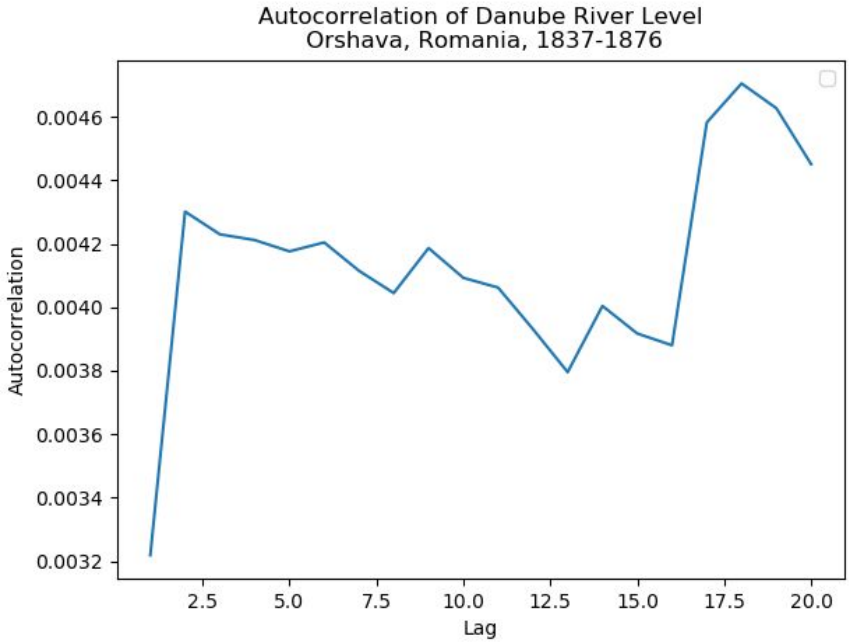
As stated earlier, the C++ and Python combo is great for scripting and automatization purposes. In its final state, the application would be able to ingest datasets of various sizes with instant computation and plotting. Making use of Python's nature a data analyst would be able to dynamically adjust different parameters like the path of the saved plots. A cron job could run a bash script once a day, verifying specific locations for new datasets, or another Python web scraper could search for new datasets online continuously. C++ speed and Python flexibility.

Results

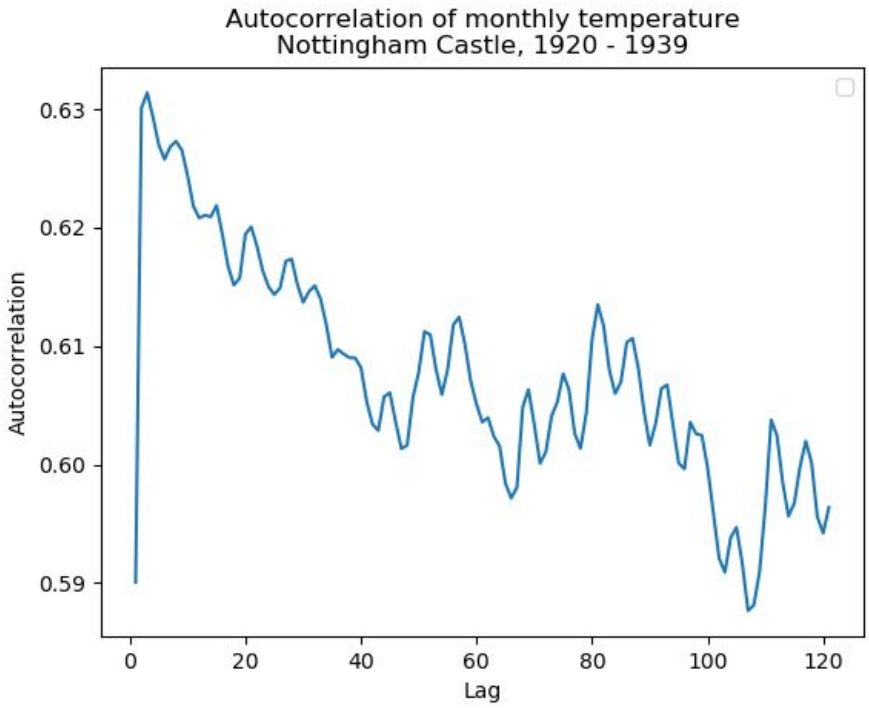
The result of autocorrelation is always 1 when lag = 0, so it was omitted.



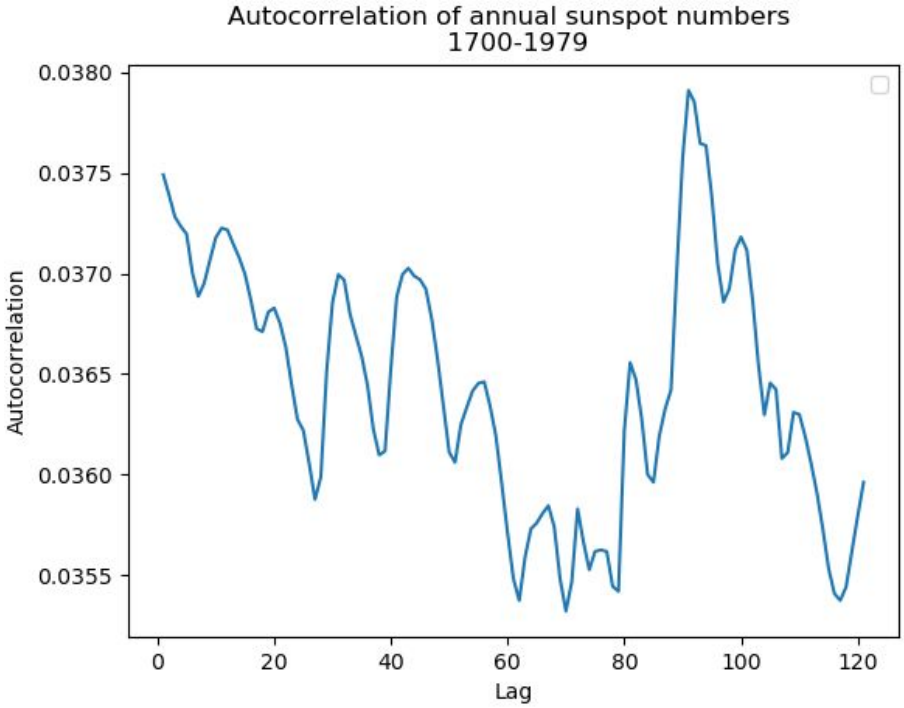
A negative autocorrelation usually changes the direction of the influence. It implies that If a particular value is below average, the next value is likely to be above average. Viceversa is also true

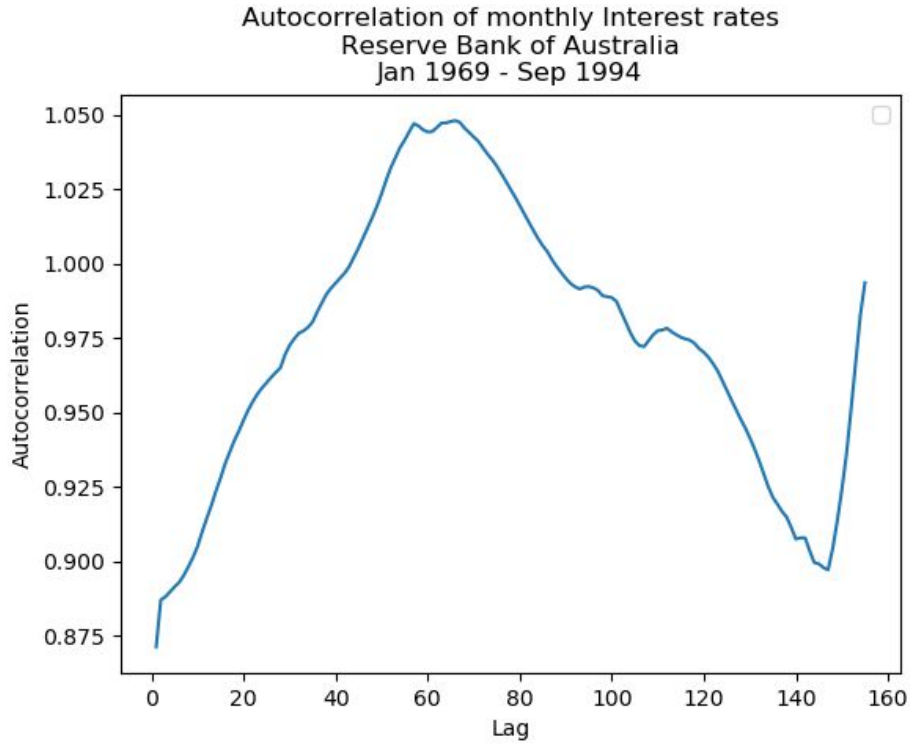


I did not find the autocorrelation of the Danube River Level too convincing, as the acf values were very low and the dataset was quite small.



The autocorrelation function is a descriptive statistic. If there is a "trend" in the data then the acf will suggest non-stationarity. However a non-stationary autocorrelation function does not necessarily suggest a "trend". If the series is impacted by one or more level shifts then the autocorrelation function will suggest non-stationarity.





Maybe the most interesting autocorrelation in my opinion was the one of the monthly interest rates from Australia. For lag between 44 and 87, the acf is over 1! There are a few possibilities: it's either the autocorrelation function is not working properly, the dataset is faulty, overlapping autocorrelation structure because of redundant values

Works Cited

Signalų Analizė Ir Apdorojimas, klevas.mif.vu.lt/~meska/SAA/.

"Autocorrelation and Trends." *Cross Validated*, stats.stackexchange.com/questions/71911/autocorrelation-and-trends.

FinYang. "FinYang/Tsdl." *GitHub*, 24 Nov. 2018, github.com/FinYang/tsdl.