Signal noise extraction using Discrete Fourier Transform in Python

Authors: Jakub Sochacki, Tomasz Kuczyński

Gdańsk University of Technology

04 February 2022

Table of contents: Page

Abstract

The leading purpose of this report is concerned with the depiction of the investigation and manipulation process of given signal data together with the use of numerical methods in order to obtain a satisfactory format of the signal containing no background noise.

Introduction

Given the data representing the sum of some signal and background noise, the first step is concerned with the basic preparation for analysis and determination of the period of the function using correlation analysis. Once that is accomplished, the next step involves applying the actual Fast Fourier Transform (FFT) algorithm on the selected interval in order to decompose a function that is dependent on time into a function that is dependent on frequency. Then closer and more precise inspection of the selected interval provides the necessary insights regarding which frequencies should be cleaned out. After selecting a specific cut-off point, certain frequencies below that threshold are zeroed and inverse Fast Fourier Transform is applied providing the original signal but without the unwanted background noise.

Technologies involved

1) The programming language of our choice was Python together with the following modules used:
    - Pandas : for data manipulation
    - Numpy : for numerical computations and algorithms
    - Matplotlib : for data visualisation
2) Jupiter notebooks for the collaboration purposes
3) Algorithms used:
    - Fast Fourier Transform

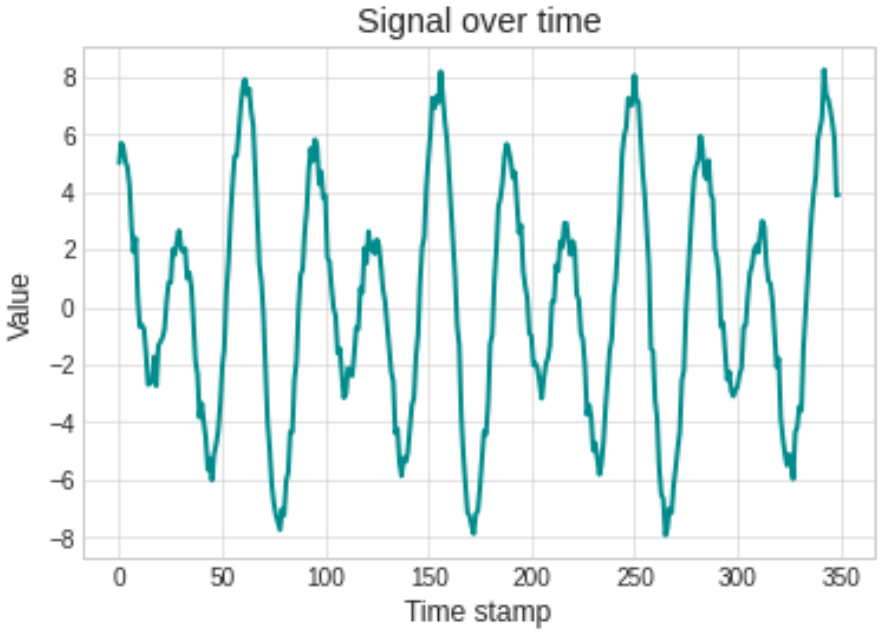
    https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html#numpy.fft.fft
    - Inverse Fast Fourier Transform

    https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft.html#numpy.fft.ifft
    - Pearson Correlation Coefficient

    https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html

Preliminary analysis

Given measurements are equally-spaced without detailed specification of the time unit, so observaions are labeled from 1 to 350.



Based on a visual inspection of the plot, it is observed that the signal follows a recurring pattern with a certain period. The goal of the further analysis is to find that period.

Determination of the period

To find the period of the signal, the Pearson's Correlation Coefficient is used. The method is concerned with comparing the first $[0: n]$ elements of the series to the next $[i: n + i]$ elements where i is incremented with each iteration. From the data that the study covers, n turns out to be 50 and i takes values from 1 to 300.

```
x1 = signal['values'][0:50]
correlations = []
for i in range(1, 300):
  x2 = signal['values'][i:i+50]
  correlations.append(np.corrcoef(x1,x2)[0, 1])
```

The above code results in the list of correlation coefficients between the initial subset of 50 observations and the subsequent subsets of the signal data. The next step is to find y of the maximum correlation coefficient and the corresponding x time stamp.

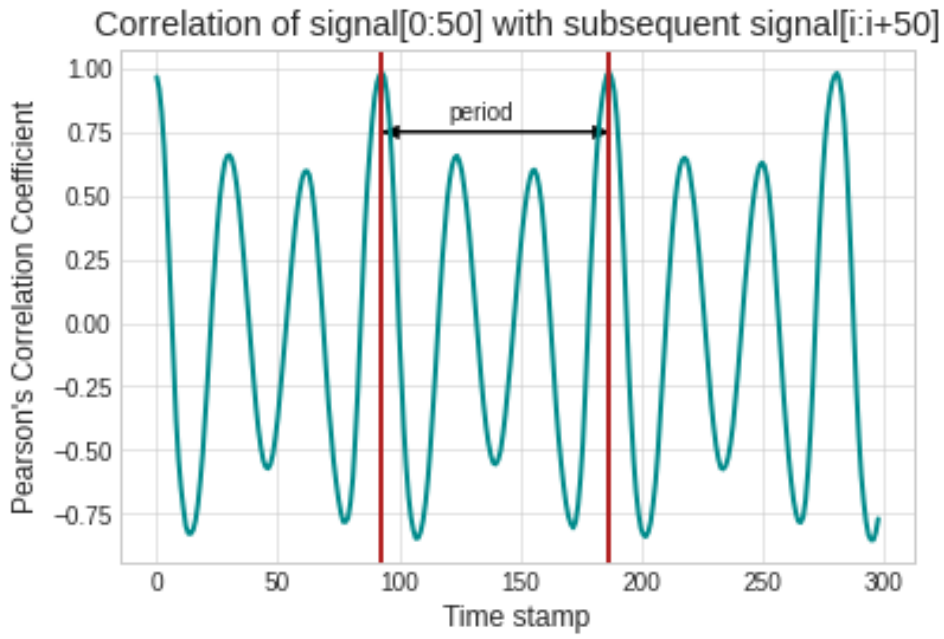
```
y_max_cor = max(correlations)
x_cor1 = correlations.index(y_max_cor)
```

Then, based on the visual inspection of the correlations plot, the next closest and greatest coefficient and its corresponding x timestamp are found in a specified range.

```
x_cor2 = correlations.index(max(correlations[150:200]))
```

The difference between the two calculated timestamps is the period of the signal.

```
period = x_cor2 - x_cor1
```



The obtained period is found to be 94 units of time.

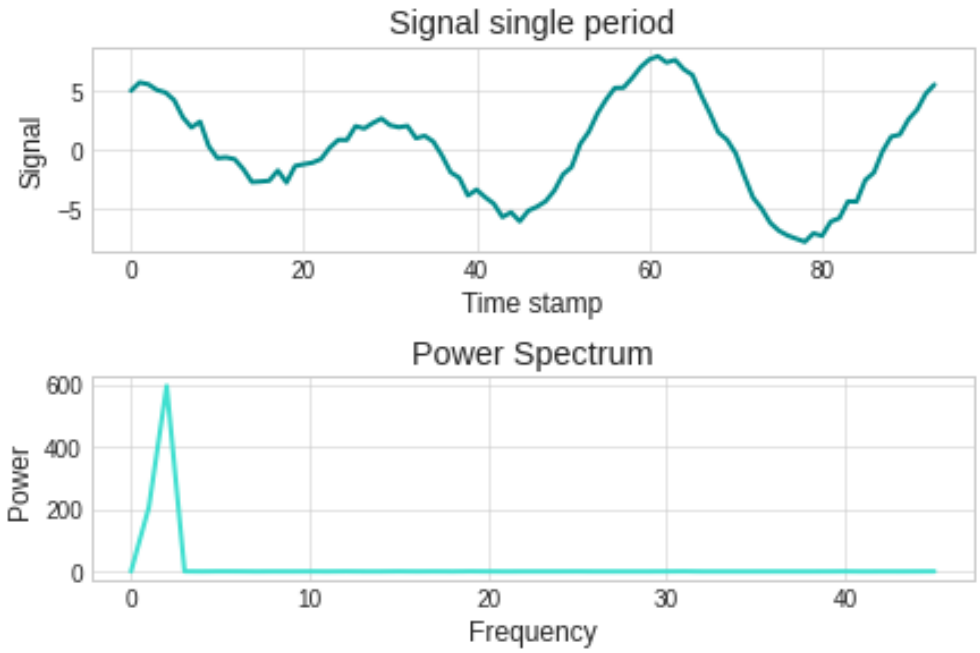
FFT Algorithm application

At this point, the Fast Fourier Transformation algorithm is applied to the values of the one particular period vector.

```
fourier_applied = np.fft.fft(period_vals, period)
dt = 1
```

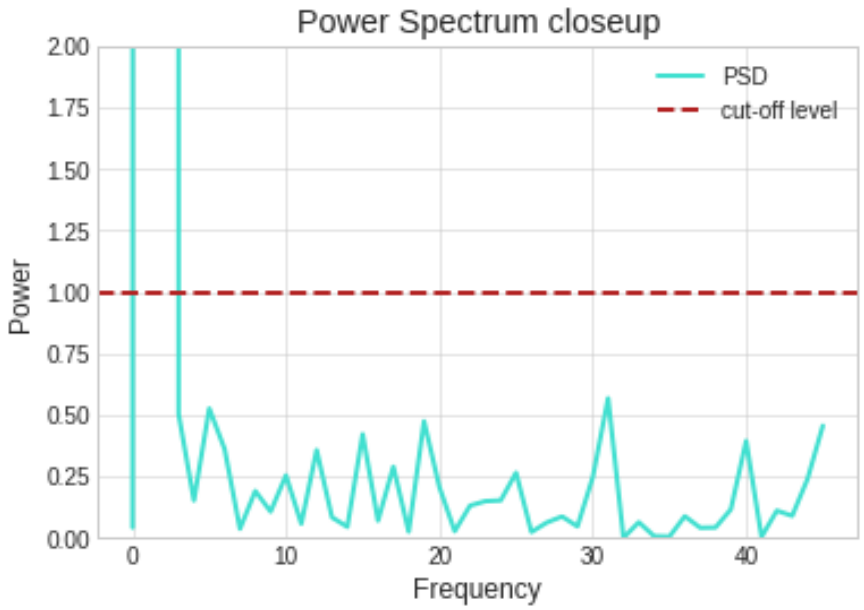
The results are also in a vector form but this time consisting of complex-valued Fourier coefficients. This newly created vector is essential to gather insights into the signal data as the obtained magnitudes are going to describe how strong particular frequencies are. Before that happens, however, the power spectrum has to be computed. That is done by multiplying the complex values from the Fourier coefficient vector by their conjugates. The resulting values are real and represent the magnitudes of the Fourier Coefficients squared.

```
# frequency power - power spectrum
PSD = fourier_applied*np.conj(fourier_applied) / period
```



Noise extraction

Based on the inspection of the power spectrum, there is only one significant frequency peak visible. Spectrum values can be filtered in such a way that only those with significant frequencies are left, and the remaining Fourier coefficients are set to 0. It is necessary to specify a certain cut-off level, below which the values will be considered as having negligible power.

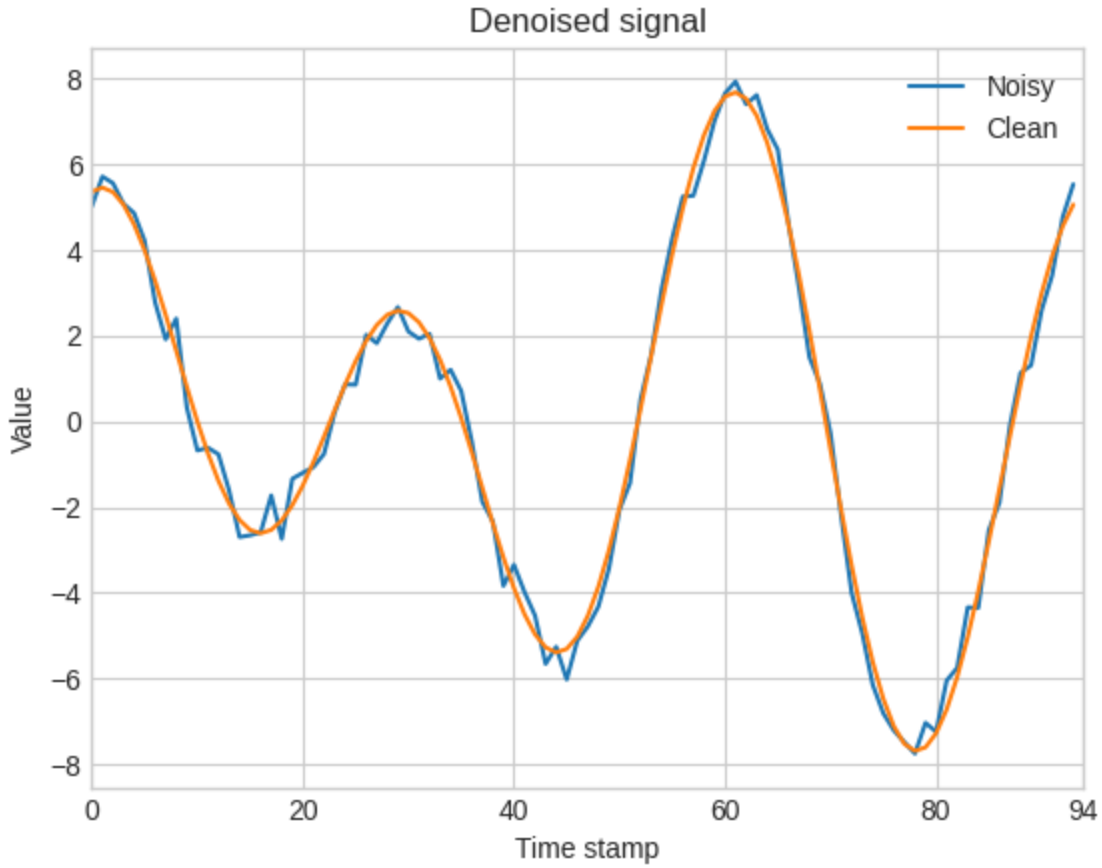


A cut-off level at 1.0 has been selected. Then, the Inverse Fast Fourier Transform is performed, providing a vector of values with the background noise removed.

```
ind = PSD > 1
PSD_clean = PSD * ind
fourier_applied = ind * fourier_applied

fourier_inv = np.fft.ifft(fourier_applied)
```

Results



The plot depicts the final result of the signal form with the background noise removed.