COURSE PROJECT

DIGITAL PROCESSING OF SIGNALS

Done by: Mert Mustafov Kamberov, group 90, Student ID:371222025

Topic: Analysis of Solar Activity Collected on 19.09.2022 from the Receiving Station in Noale, Italy

# Introduction

As stated, the topic of this project is the analysis of solar activity collected on the date 19.09.2022 from the receiving station in Noale, Italy. For this purpose, the data from the file Noale\_Italy\_26\_7\_dBm.csv will be used. These data will be processed using the Python programming language. The main libraries to be used are **pandas**, **matplotlib**, **scipy**, and **numpy**. Of course, other libraries can also be used, but since these are the most popular and well-developed for data processing and visualization, I believe they will be ideal for this purpose.

The data from the file will be visualized and compared with data from [SpaceWeatherLive - Solar Activity](https://www.spaceweatherlive.com/en/solar-activity/solar-flares.html" \t "_new), and the differences between the signals will be analyzed. The number of reports, as well as the periods during which they were made, will be determined, and potential gaps in the reports and their causes will be analyzed.

After visualizing the reports using a histogram, a Fourier transformation will be applied, and the result will be visualized. The final task will be to normalize the signal, marking the conclusion of the project.

# Fourier Transform, Discrete Fourier Transform, Fast Fourier Transform

The Fourier Transform (FT) is a mathematical method that decomposes a signal from the time domain into its spectrum in the frequency domain. In simpler terms, the Fourier Transform "breaks down" a given signal into a sum of sinusoids with different frequencies, phases, and amplitudes. The transform is represented by the following equation:

F(ω) represents the signal in the frequency domain, while *f(t)* is the signal itself. Using Euler's number, we take advantage of its formula:

After applying the same transform to F(ω) we get:

If *f(t)* is an even function, then f(ω) = f(t). If *f(x)* is an odd function, then f(ω) = f(-t). When *f(t)* is neither even nor odd, in most cases, it can be divided into an even part and an odd part.

To avoid confusion, it is customary to write the FT (1) and the inverse transform (3) together:

There are also functions for which the Fourier Transform does not exist. However, for most physical functions, the transform does exist.

In the case of the Discrete Fourier Transform (DFT), just as with the continuous transform, there are forward and inverse transforms. With the forward transform, we find the spectrum of the signal, while with the inverse transform, we obtain the discrete signal from the spectrum

To obtain the DFT, we replace integration „∫“with summation „Σ“ and the continuous signal „*f(t)*“ with its discrete counterpart „*f­D(nT)*“.

After applying the Fourier Transform to the discrete signal, we obtain:

For the DFT to be invertible, the sampling frequency Ω must be at least twice the highest frequency in the spectrum of the continuous signal, i.e., Ω ≥ 2ωmax. Since continuous calculations cannot be processed on digital devices, we introduce a sampling frequency Δω, and the DFT takes the following form:

Usually . The DFT and its inverse take the following form:

One of the major problems with the DFT is the speed at which it is performed. To determine the spectrum of a discrete signal with N samples, N2 multiplications and the same number of additions are required. This means the complexity of the DFT algorithm is O(n2). For instance, with a sample size of 105, 1010 operations are needed, which is extremely time- and resource-intensive.

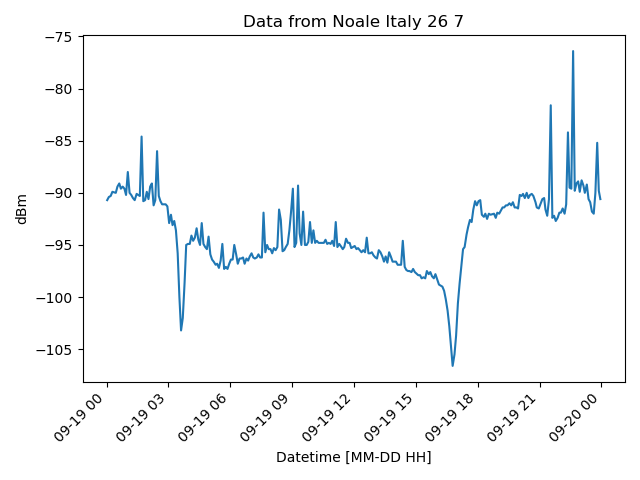
In the Fast Fourier Transform (FFT), the algorithm's complexity is O(n\*log­2(n)). The FFT works best with a signal whose number of samples is a power of 2, but even if it is not, the signal is padded with zeros until the number of samples becomes a power of 2. The algorithm follows these steps:

* The signal is divided into two sub-signals, where the elements of the first sub-signal are the even-position elements of the original signal, and the elements of the second sub-signal are the odd-position elements. This is done recursively for the sub-signals until there are sub-signals with 2 elements.
* A DFT is performed on each of these sub-signals with 2 elements.
* Using the symmetric properties of the DFT and carefully handling the equality and parity of the element positions in the sub-signals, we sum the transformed sub-signals in pairs (2 by 2). This process is repeated until only one transformed signal remains, which is the spectrum of the original signal.

# Course project realization

a)

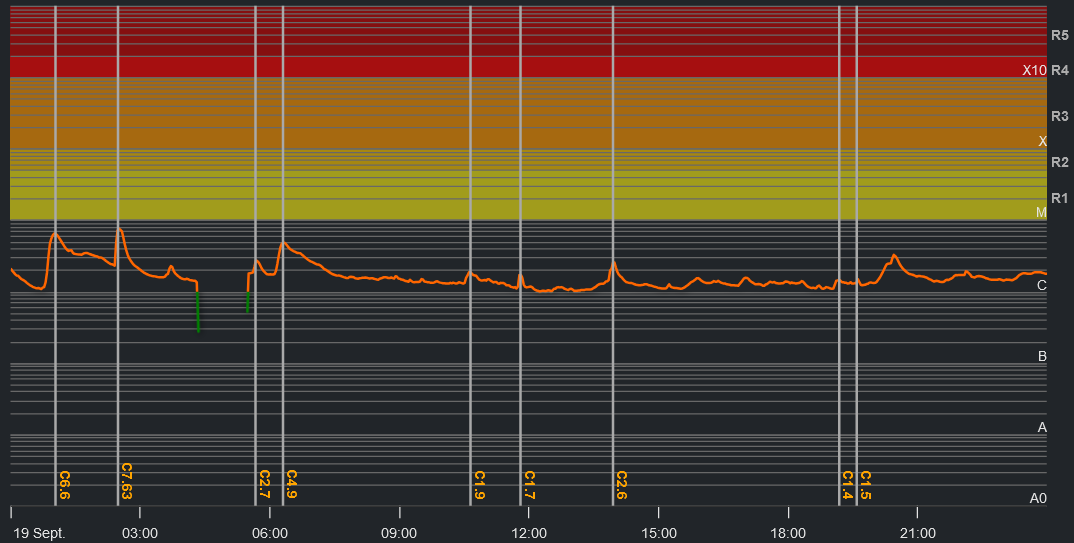
For saving the content of the file, I use the pandas library, and for visualization, I use matplotlib. After completing the task, the following graph is obtained:



**Figure 1**: Solar Activity from Noale\_Italy

b)

The data for solar activity for the same day from Space Weather can be found at the following link: <https://www.spaceweatherlive.com/en/archive/2022/09/19/xray.html>. The graph from the website looks as follows:



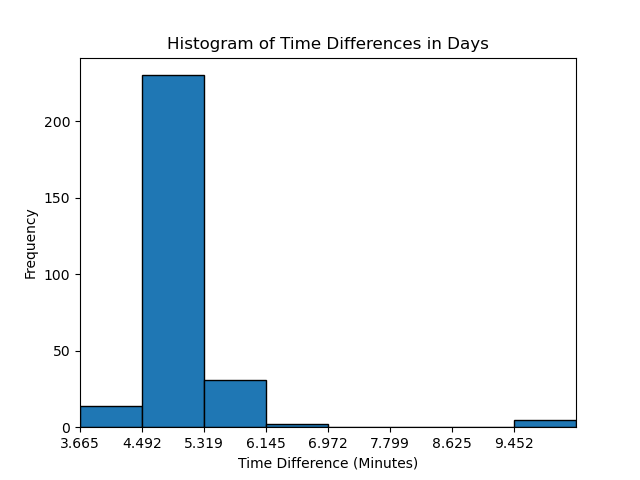
**Figure 2**: Solar Activity from the GOES X-ray Satellite on 19.09.2022

The main difference between the two graphs is that the second one does not have the distinct "dips," which are the result of the sunrise and sunset. Another key difference is that the second graph is much smoother compared to the first.

c)

The measurements were taken approximately every 5 minutes, and the total number of reports is 282. In my opinion, there are 5 missing reports due to the fact that a day has 1440 minutes, and dividing 1440 by 5 equals 288. Since the last measurement of the day is at 24:00, which is technically 00:00 of the next day, logically, there should be 287 reports in a day. In our case, there are 282, and the 5 missing reports are most likely due to errors in the recording system.

d)

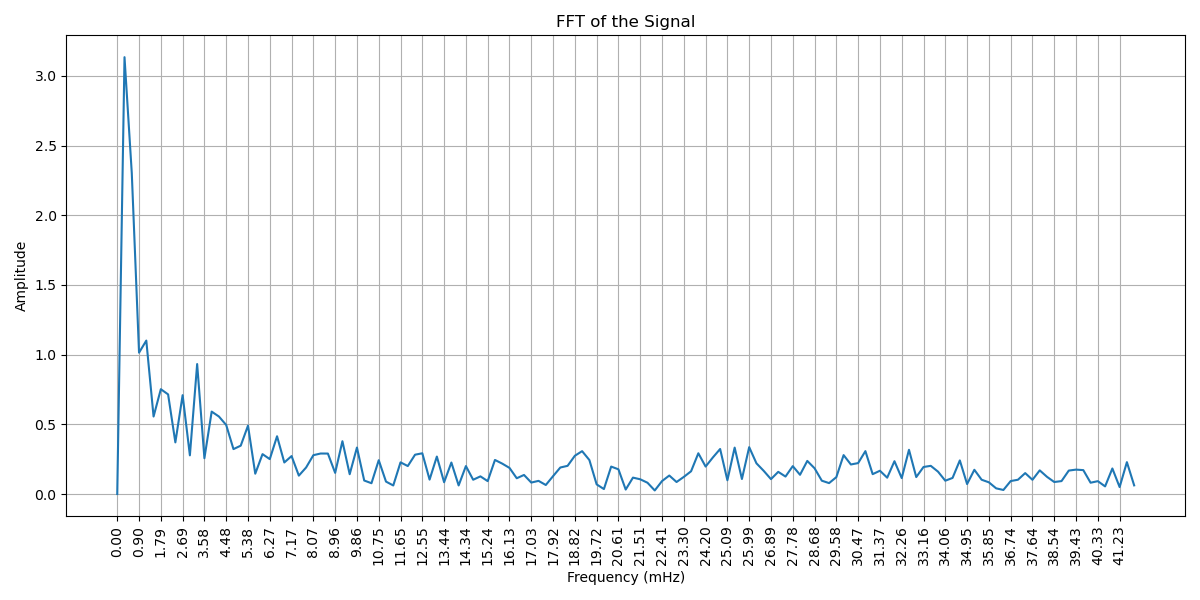


**Figure 3**: Histogram of Reporting Times from Noale\_Italy

The histogram provides strong support for the claims made in subsection B). It is clearly visible that most measurement intervals fall between 4.5 and 5.3 minutes, while the neighboring intervals are drastically fewer. Another notable observation is that there are a few intervals exceeding 9.5 minutes—exactly 5 of them. As stated earlier, these correspond to measurements with a missed reading preceding them, effectively doubling their interval length.

e)

After applying the Fast Fourier Transform to the measurements, the frequency spectrum of the data is obtained, as shown in Figure 4. The frequency is given in millihertz. The most prominent frequency is approximately 0.38 mHz.

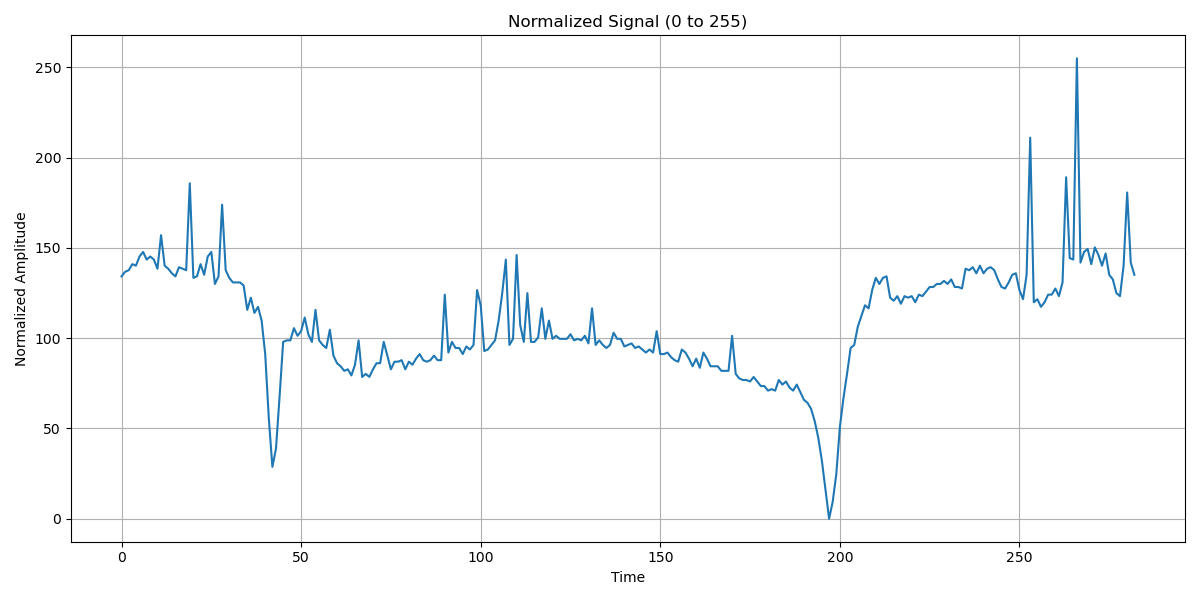


**Figure 4:** Frequency Spectrum of the Data from Noale\_Italy

f)

I normalize the signal to a range from 0 to 255 using the following method:

For the implementation of this, I use the array operations from the **numpy** library, which eliminate the need for a loop to accomplish the task.



**Figure 5**: Normalized signal [0-255] of the data from Noale\_Italy

# Annex

The implementation was carried out using Python. I used both a script and Jupyter Notebook, as I believe the latter provides a much neater way to organize the work. The code implementation has been uploaded to [GitHub](https://github.com/meroo893/Digital-processing-of-signals/tree/main/Fourier_course_wrok) for more convenient access. The implementation:

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.image as mpimg  import numpy as np  from scipy.fft import fft, fftfreq  from scipy.signal import detrend  def load\_data():  data = pd.read\_csv('../data/Noale\_Italy\_26\_7\_dBm.csv')  data['Date'] = pd.to\_datetime(data['Date'])  data['Value'] = pd.to\_numeric(data['Value'])  return data  def plot\_data(data):  plt.plot(data['Date'], data['Value'])  plt.title('Data from Noale Italy 26 7')  plt.xticks(rotation=45, ha='right')  plt.xlabel('Datetime [MM-DD HH]')  plt.ylabel('dBm')  plt.tight\_layout()  plt.savefig(f'../artifacts/date\_to\_value\_plot\_FOURIER.png', format='png')  plt.show()  def plot\_histogram(data):  timediff = data['Date'].diff()[1:]  timediff\_minutes = timediff.dt.total\_seconds() / 60  bin\_count = 8  # Plot histogram  plt.xlabel('Time Difference (Minutes)')  plt.xlim(min(timediff\_minutes), max(timediff\_minutes))  # Setting better alignment for x ticks  plt.xticks(np.arange(min(timediff\_minutes), max(timediff\_minutes),  (max(timediff\_minutes) - min(timediff\_minutes)) / bin\_count))  hist\_info = plt.hist(timediff\_minutes, bins=bin\_count, edgecolor='black') # You can adjust the number of bins  plt.ylabel('Frequency')  plt.title('Histogram of Time Differences in Days')  plt.savefig(f'../artifacts/timediff\_histo\_FOURIER.png', format='png')  plt.show()  return hist\_info  def plot\_fft(data):  data = np.asarray(data)  data = detrend(data)  N = len(data)  # Perform FFT  yf = fft(data)  # Generate frequency bins (with appropriate sampling rate if applicable)  sampling\_rate = avg\_sample\_time / 60 # 5 minutes per sample  xf = fftfreq(int(N), 1 / sampling\_rate) # Frequency bins  # Keep only the positive half of the spectrum  xf = xf[:int(N // 2)]  yf = yf[:int(N // 2)]  # Plot FFT  plt.figure(figsize=(12, 6))  plt.plot(xf \* 1000, 2.0 / N \* np.abs(yf)) # Convert xf to mHz  plt.title("FFT of the Signal")  plt.xlabel("Frequency (mHz)")  plt.ylabel("Amplitude")  # plt.yscale('log')  plt.xticks([xf[i] \* 1000 for i in range(len(xf)) if i % 3 == 0], rotation=90) # Adjust tick step  plt.grid()  plt.tight\_layout()  plt.savefig(f'../artifacts/fft\_plot.png', format='png')  plt.show()  dominant\_freqs = xf[np.argsort(np.abs(xf) \*\* 2)[-15:]] # Top 5 frequencies  print("Dominant Frequencies:", dominant\_freqs)  def normalize(data):  x\_max = np.max(data)  x\_min = np.min(data)  normalized = ((data - x\_min) / (x\_max - x\_min)) \* 255  return normalized  if \_\_name\_\_ == '\_\_main\_\_':  signal = load\_data()  plot\_data(signal)  plot\_data(signal)  plt.imshow(mpimg.imread('../data/solar-activity.png'))  plt.axis('off') # Turn off axis labels  plt.title("Solar activity data from www.spaceweatherlive.com")  plt.show()  hist\_info = plot\_histogram(signal)  frequencies = hist\_info[0]  bins = hist\_info[1]  total\_samples = sum(frequencies)  avg\_sample\_time = sum(  [frequencies[i] \* ((bins[i] + bins[i + 1]) \* 0.5) for i in range(len(frequencies))]) / total\_samples  missing\_samples = 24 \* 60 / 5 - total\_samples - 1  print({"minutes in a day": 24 \* 60, "total": total\_samples, "missing": missing\_samples,  "average sample time [min]": avg\_sample\_time})  plot\_fft(signal['Value'])  normalized\_signal = normalize(signal['Value'])  # Plot the normalized signal  plt.figure(figsize=(12, 6))  plt.plot(normalized\_signal)  plt.title("Normalized Signal (0 to 255)")  plt.xlabel("Time")  plt.ylabel("Normalized Amplitude")  plt.grid()  plt.tight\_layout()  plt.savefig("../artifacts/normalized\_signal.png")  plt.show() |