**ЗАДАНИЕ**

**ЗА КУРСОВ ПРОЕКТ**

**ПО „ДИГИТАЛНА ОБРАБОТКА НА СИГНАЛИ“**

на студент: Мерт Мустафов Камберов, гр. 90, фак. № 3712220253

**Тема: Анализ на слънчевата активност, събрана за дата 19.09.2022, от приемната станция в Noale\_Italy**

# Увод в курсовия проект

Както е посочено, темата на този проект е анализ на слънчевата активност, събрана за дата 19.09.2022, от приемната станция в Noale\_Italy. За целта ще бъдат използвани данните от **Noale\_Italy\_26\_7\_dBm.csv**. Тези данни ще бъдат обработени с помощта на програмния език **Python**. Библиотеките, които ще бъдат използвани основно са **pandas, matplotlib, scipy** и **numpy.** Разбира се, могат да бъдат използвани и други от посочените, но тъй като тези са най-популярните и съответно най-развити за обработка и визуализация на данни, смятам, че точно те ще бъдат идеални за целта. Данните от файла ще бъдат визуализирани и сравнени с данни от <https://www.spaceweatherlive.com/en/solar-activity/solar-flares.html> и ще бъдат анализирани разликите между сигналите. Ще се изведат броя на отчетите както и през какъв период са направени и ще бъде анализирано евентуалната липса на отчети и причината за това. След като отчетите бъдат визуализирани чрес хистограма, ще бъде приложена трансформация на Фурие и резултатът ще бъде визуализиран. Крайната задача ще бъде сигналът да бъде нормиран и с това ще приключи курсовият проект.

# Трансформация на Фурие, дискретна трансформация на Фурие и бърза трансформация на Фурие

Трансформацията на Фурие(ТФ) е математически похват, който разлага сигнал от времевата област към спектъра на сигнала в честотната област. С по-прости думи, трансформацията на Фурие „раздробява“ даден сигнал на сбор от синусоиди с различна честота, фаза и амплитуда. Трансформацията се представя със следното уравнение:

е представянето на сигнала в честотната област, а е самият сигнал, а чрез числото на Ойлер се възползваме от едноименната му формула:

При прилагане на същата трансформация на получаваме:

Ако *f(t)* е четна функция, тогава *f()=f(t).* Ако *f(x)* е нечетна функция, тогава *f()=f(-t)*. Когато, *f(t)* е нито четна, нито нечетна, в повечето случаи тя може да се раздели на четна част и нечетна част.

За да се избегне объркване, е прието да се записва ТФ(1) и обратната трансформация(3) заедно:

Има и функции, за които не съществува ТФ. Но за повечето физични функции трансформацията съществува.

При дискретната трансформация на Фурие(ДТФ), също както при непрекъснатата, има права и обратна трансформация. При правата намираме спектъра на сигнала, а при обратаната – от спектъра намираме дискретния сигнал.

За да получим ДТФ заместваме интеграцията „∫“ със сумиране „Σ“ и непрекъснатия сигнал „*f(t)*“ с дискретния такъв „*f­D(nT)*“.

След прилагане на ТФ на дискретния сигнал получаваме:

За да може да бъде обратима ДТФ, честотата на вземане на извадки „Ω“ трябва да е поне 2 пъти по-голяма от най-голямата честота на спектъра на непрекъснатия сигнал т.е Ω ≥ 2ωmax. Тъй като непрекъснати изчисления не могат да бъдат обработени на дигитални устройства, въвеждаме честота на извадка “Δω” и ДТФ придобива следния вид:

В най-честия случай . ДТФ и обратниата ДТФ имат следния вид:

Един от големите проблеми на ДТФ е бързината, с която се извършва. За определяне на спектъра на дискретен сигнал с брой на извадките N са ни нужни N2 на брой умножения и също толкова на брой събирания. Това означава, че сложността на алгоритъма за ДТФ е O(n2). Тоест, при извадка от 105 елемента са ни нужни 1010 операции, което е изключително трудо- и капацитетоемко.

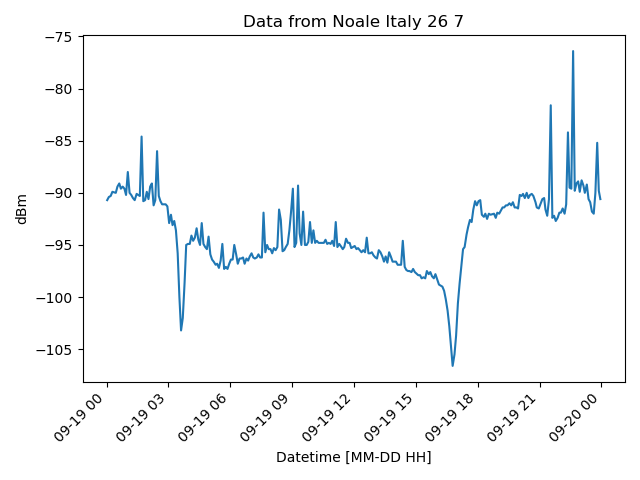
При бързата трансформация на Фурие (БТФ) сложността на алгоритъма е O(n\*log­2(n)). БТФ работи най-добре със сигнал, чиийто брой извадки е степен на 2, но дори да не е – сигналът се допълва с нули докато броя на извадките не е степен на 2. Алгоритъмът има следните стъпки:

* Сигналът се разделя на 2 подсигнала, като елеметите от първия са елементите на четна позиция от първоначалния сигнал, а елементите на втория – на нечетна. Това се извършва на подсигналите, докато нямаме подсигнали с по 2 елемента.
* Извършва се ДФТ за всяка от тези подсигнали с по 2 елемента.
* Възползвайки се, от симетричните свойства на ДФТ и внимавайки за равенството и четността на позициите на елементите в подсигналите, сумираме трансформираните подсигнали 2 по 2. Това действие се повтаря докато не остане само един трансформиран сигнал, който се явява спектъра на първоначалния сигнал.

# Реализация на курсовия проект

a)

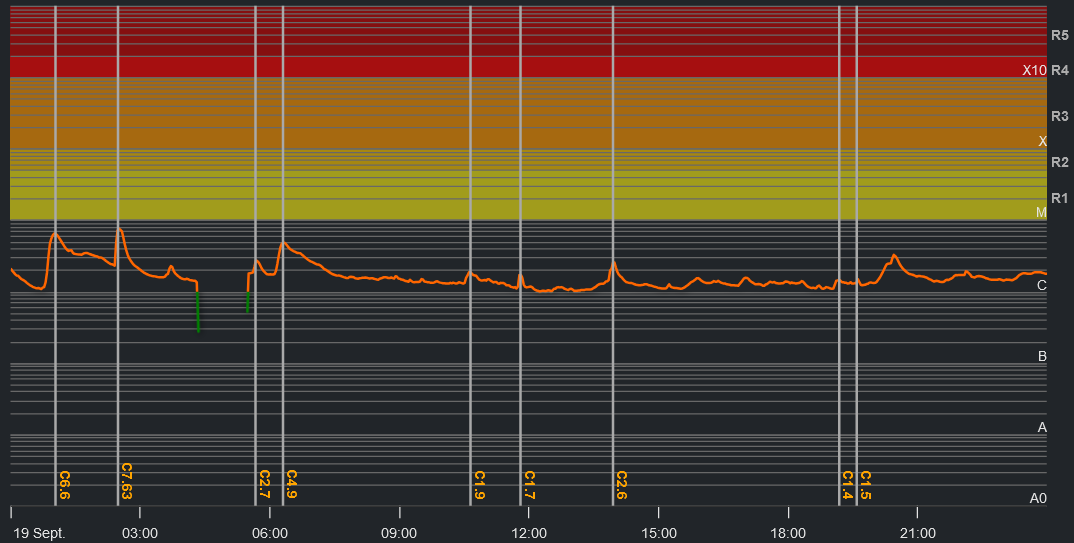
За записванет съдържанието на файла използвам библиотеката **pandas**, а за визуализацията им ползвам **matplotlib**. След изпълнение на задачата получава следната графика:



**Фиг.1** Слънчева активност от **Noale\_Italy**

б)

Данните за слънчевата активност за същия ден от space weather се намират в следния линк: <https://www.spaceweatherlive.com/en/archive/2022/09/19/xray.html>. Графиката от сайта изглежда по следния начин:



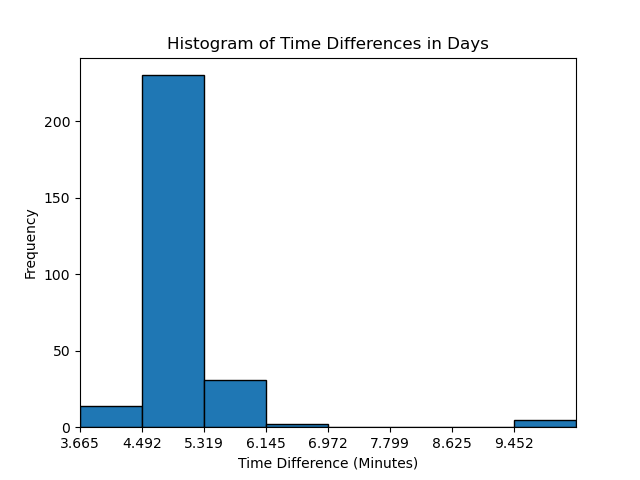
**Фиг. 2** Слънчева активност от GOES X-ray satellite на 19.09.2022

Основната разлика между двете графики е, че във втората ги няма тази ясно изразени „вдлъбнатини“, които са резултат от изгрева и залеза на слънцето. Друга ключова разлика е, че втората графика е доста по-полегата от първата.

В)

Измерванията са направени приблизително през 5 минути, а общият брой на отчетите е 282. Липсващите отчети, според мен, са 5 на брой, поради факта, че в един ден има 1440 минути, а 1440 разделено на 5 е равно на 288. Тъй като последното измерване за деня е в 24:00, а това се явява 00:00 следващият ден, то по тази логига следва в един ден да има 287 отчети. В нашия случай има 282, а 5те липсващи отчети, най-вероятно, се дължат на грешки в отчитащата система.

Г)

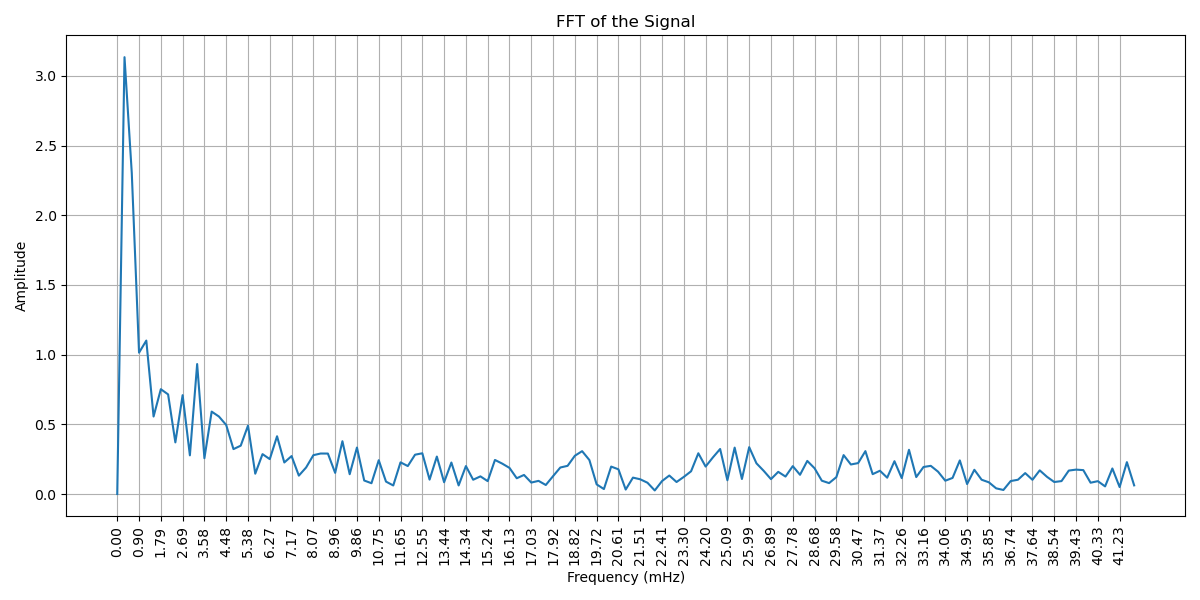


**Фиг. 3** Хистограма на времето на отчетите от Noale\_Italy

Хистограмата дава положителен аргумент към твърденията от подточка В). Ясно се вижда, че най-много интервали на измервания са между 4.5 и 5.3 минути. Съседните на тях са драстично по-малко. Друго, което прави впечатление е, че има няколко измервания с интервал над 9.5 минути, точно 5 такива. Както е написано по-горе, това са измервания, които имат пропуснато измерване преди тях и съответно тяхната дължина се удвоява.

Д)

След прилагане на бързата трансформация на Фурие върху измерванията се получава честотния спектър на измерванията, която е изобразена на фиг. 4. Честотата на спектъра е в милихерцове. Най-открояващата се честота е приблизително 0.38 mHz.



**Фиг. 4** Честотен спектър на данните от Noale\_Italy

Е)

Нормирам сигнала в диапазона от 0 до 255, използвайки метода:

За програмната реализация на това използвам матричните действия на масивите от библиотеката **numpy**, с които не е нужно да правим цикъл за поставената задача.

# Анекс

Програмната реализация е извършена с помощта на Python. Използвал съм както скрипт, така и Jupyter Notebook, в който, според мен, нещата стават много по-прилежни. Реализацията на кода е качена в GitHub за по-удобен достъп тук. Реализацията:

|  |
| --- |
| 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() |