MOWNIT - Zestaw 4 Szybka transformata Fouriera

Opracował: Mateusz Woś

Wszystkie zadania zostały zaimplementowane w języku Python.
Korzystałem z bibliotek matplotlib do rysowania wykresów i biblioteki numpy, która zawiera wszystkie potrzebne funkcje z biblioteki gsl do wykonania sprawozdania.

1. *Proszę wygenerować 256-elementową tablicę wartości funkcji będącej sumą czterech cosinusów o różnych okresach i amplitudach, korzystając np. z wzoru: data[i]=cos(4*Pi*i/N)+cos(16*Pi*i/N)/5+cos(32*Pi*i/N)/8+cos(128*Pi*i/N)/16 gdzie N = 256. Narysować wykres tej funkcji korzystając z Gnuplota. Tę funkcję nazywamy sygnałem.*

Import bibliotek:

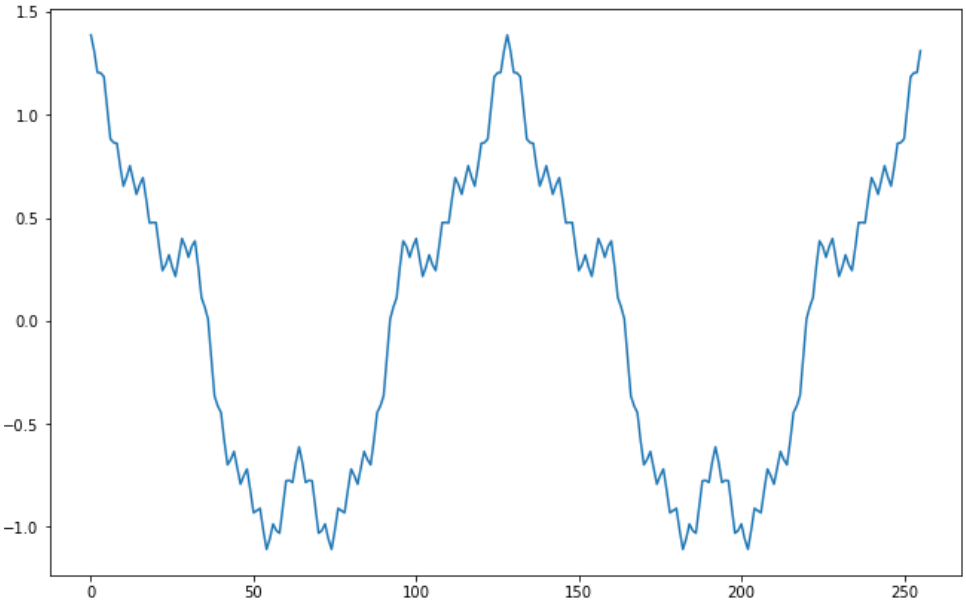
```
import numpy as np
from matplotlib import pyplot as plt
```

Wypełnienie tablicy danymi z zadania:

```
data = []
for i in range(256):
    data.append(np.cos(4*np.pi*i/256)
                +(np.cos(16*np.pi*i/256)/5)
                +(np.cos(32*np.pi*i/256)/8)
                +(np.cos(128*np.pi*i/256)/16))
```

Wygenerowanie wykresu funkcji sygnału:

```
plt.figure(figsize=(11,7))
plt.plot(data)
plt.show()
```



2. *Proszę użyć funkcji gsl_fft_real_radix2_transform do otrzymania transformaty Fouriera sygnału. Narysować wykres słupkowy tej transformaty. Czy na tym wykresie widać związek z częstotliwościami i amplitudami cosinusów wchodzących w skład funkcji? Ten wykres nazywamy widmem częstotliwości.*

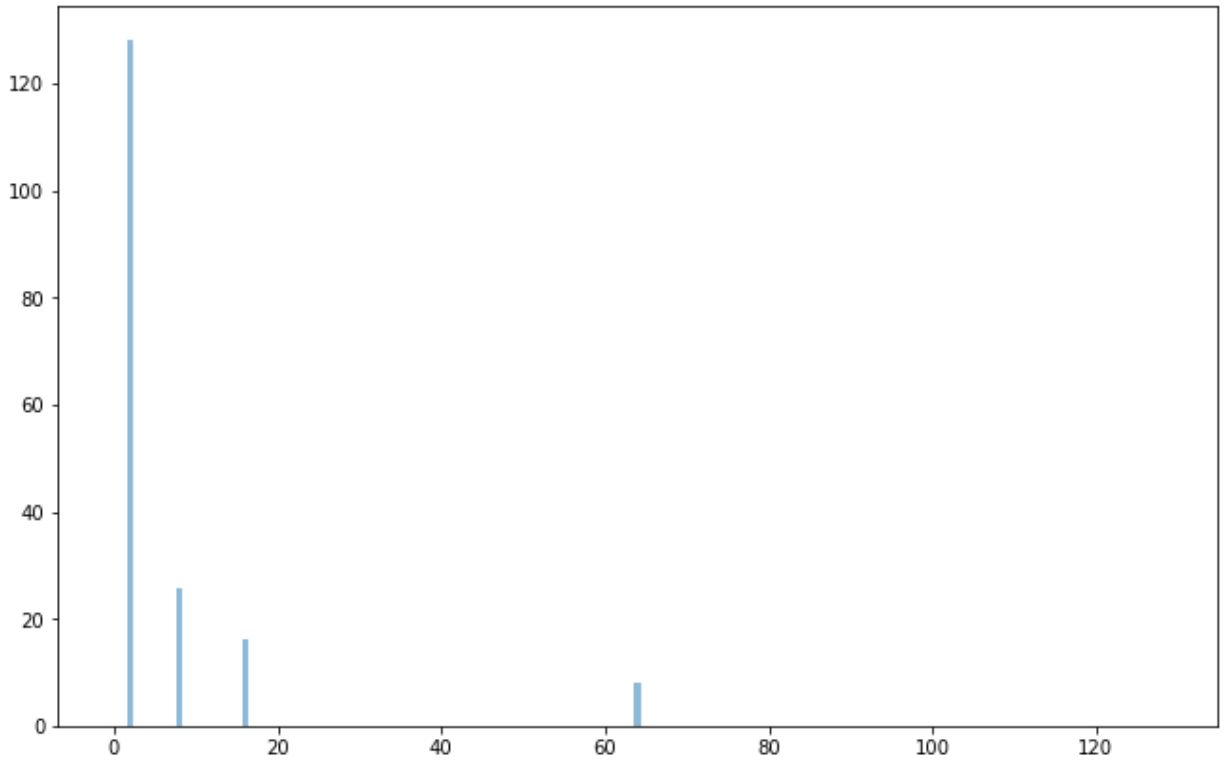
Wykonanie transformaty Fouriera:

```
transfor = np.fft.rfft(data)
```

Stworzenie wykresu słupkowego:

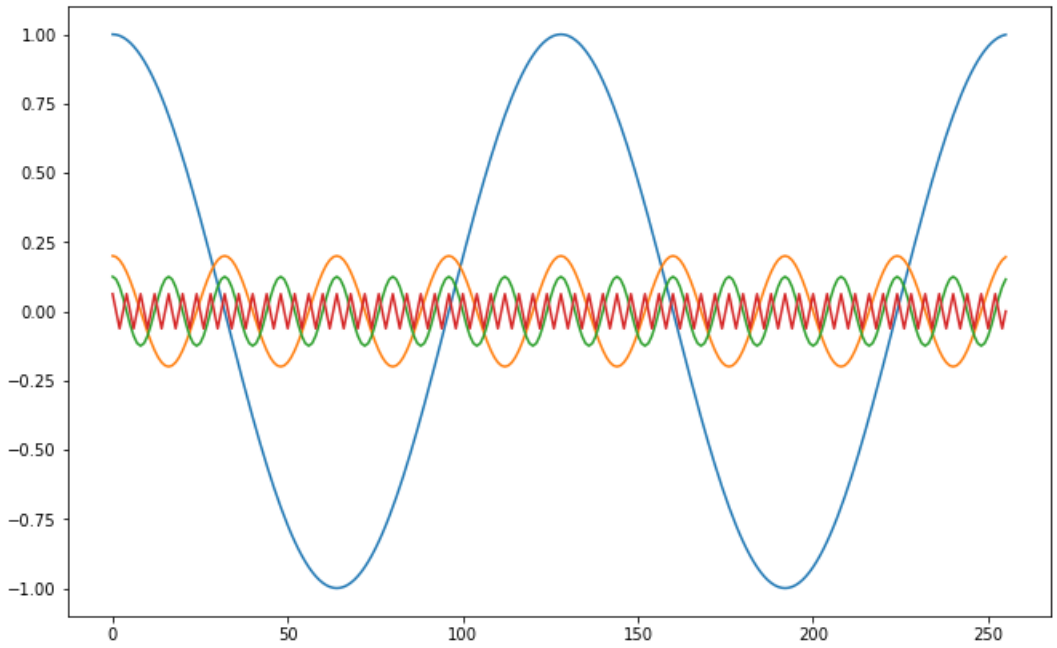
```
real, imag = transfor.real, transfor.imag
```

```
plt.figure(figsize=(15,10))
x = np.arange(len(real))
plt.bar(x, real , align='center', alpha=0.5)
plt.show()
```



Na powyższych wykresie widać związek z częstotliwościami i amplitudami cosinusów wchodzących w skład funkcji. Argumenty w powyższym wykresie odpowiadają połowom częstotliwości cosinusów. Dla częstotliwości 4 mamy argument 2, dla 16 to 8, dla 32 to 16, a dla 128 to 64. Natomiast wartości słupków odpowiadają amplitudy. Dla nie zmienionej jest to 128, kolejna to podzielona przez 5, następne to podzielona przez 8 – 8-krotnie mniejsza i ostatni słupek odpowiada tej podzielone przez 16 – jest 16krotnie mniejsza od pierwszej wartości.

Wykres cosinusów składających się na sygnał:



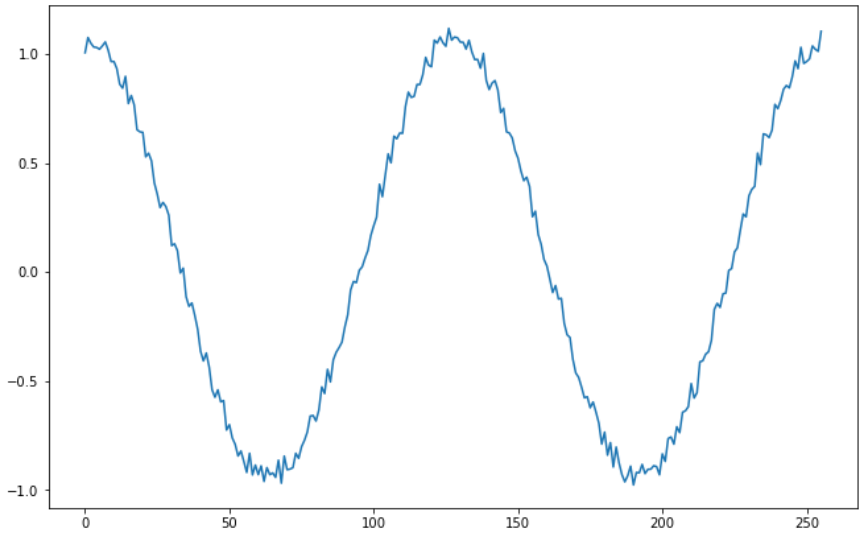
3. *Teraz wypełniamy tablicę wartościami funkcji cosinus zaburzonej niewielkim "szumem", np. korzystając z wzoru: data[i]=cos(4*Pi*i/N)+((float)rand())/RAND_MAX/8.0 Proszę narysować wykres tej funkcji.*

Wypełnienie tablicy danymi:

```
import random
data2 = []
for i in range(256):
    data2.append(np.cos(4*np.pi*i/256)
                 +random.random()/8.0)
```

Wykres:

```
plt.figure(figsize=(11,7))
plt.plot(data2)
plt.show()
```



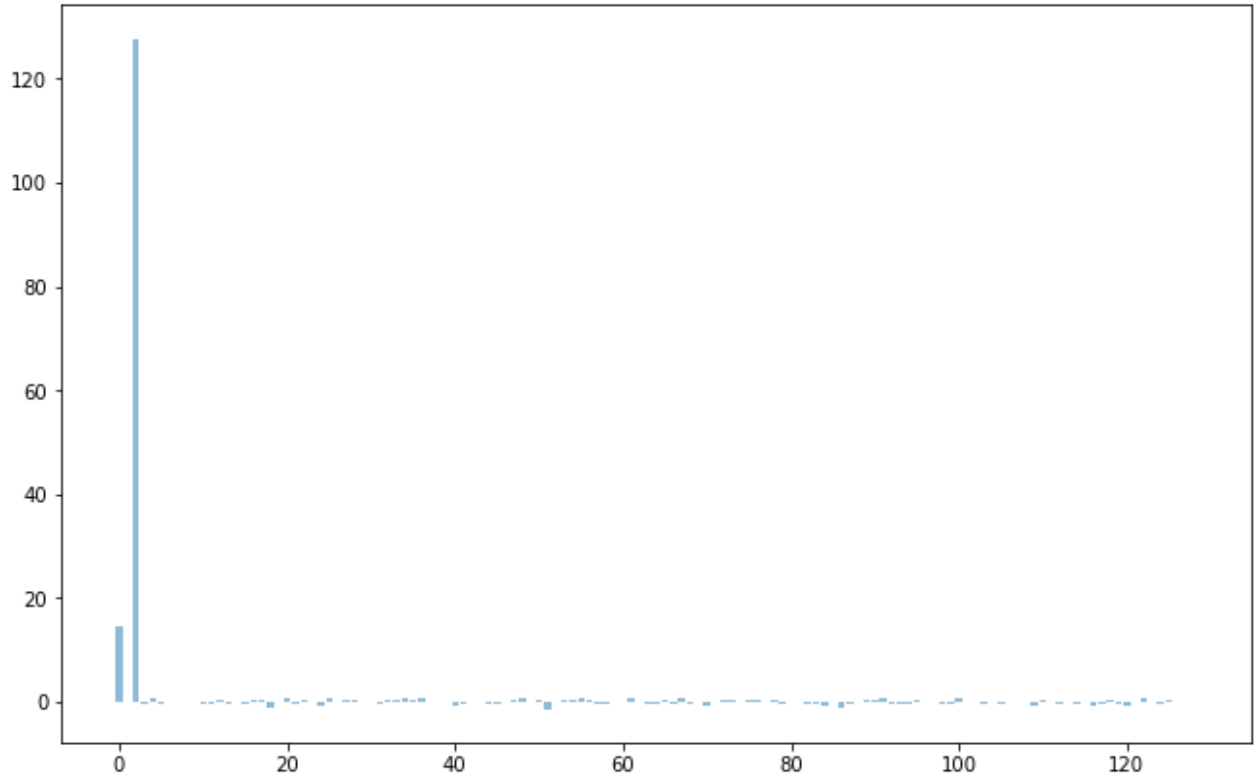
4. *Narysować wykres transformaty Fouriera tego sygnału (jak w punkcie 2).*

Wykonanie transformaty Fouriera:

```
transfor2 = np.fft.rfft(data2)
```

Stworzenie wykresu słupkowego:

```
plt.figure(figsize=(11,7))
x2 = np.arange(len(transfor2.real))
plt.bar(x2, transfor2.real , align='center', alpha=0.5)
plt.show()
```



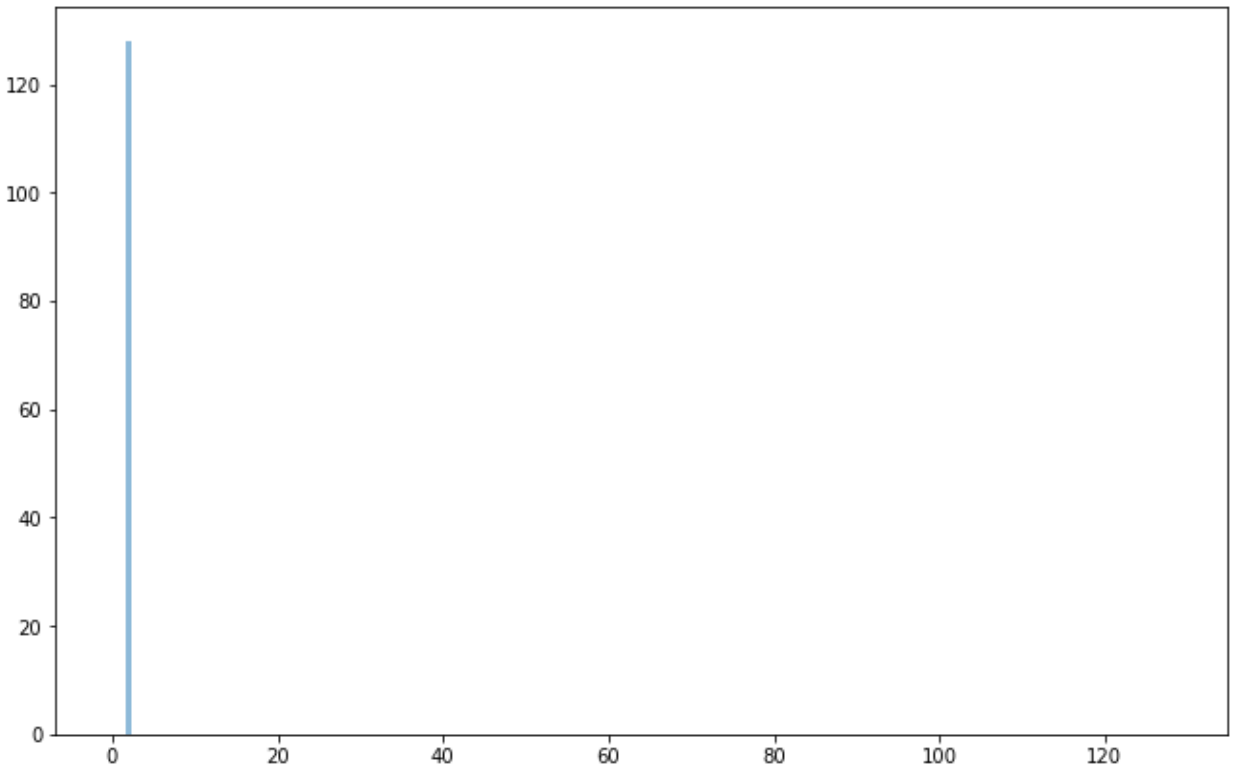
5. *Po transformacie wyzerować w widmie wszystkie elementy, których wartość bezwzględna jest mniejsza niż 50. W ten sposób usuwamy "szumy" z sygnału.*

Usuwanie szumów:

```
filter_data = [x if x > 50 else 0 for x in transfor2.real]
```

Wykres:

```
plt.figure(figsize=(11,7))
x2 = np.arange(len(filter_data))
plt.bar(x2, filter_data , align='center', alpha=0.5)
plt.show()
```



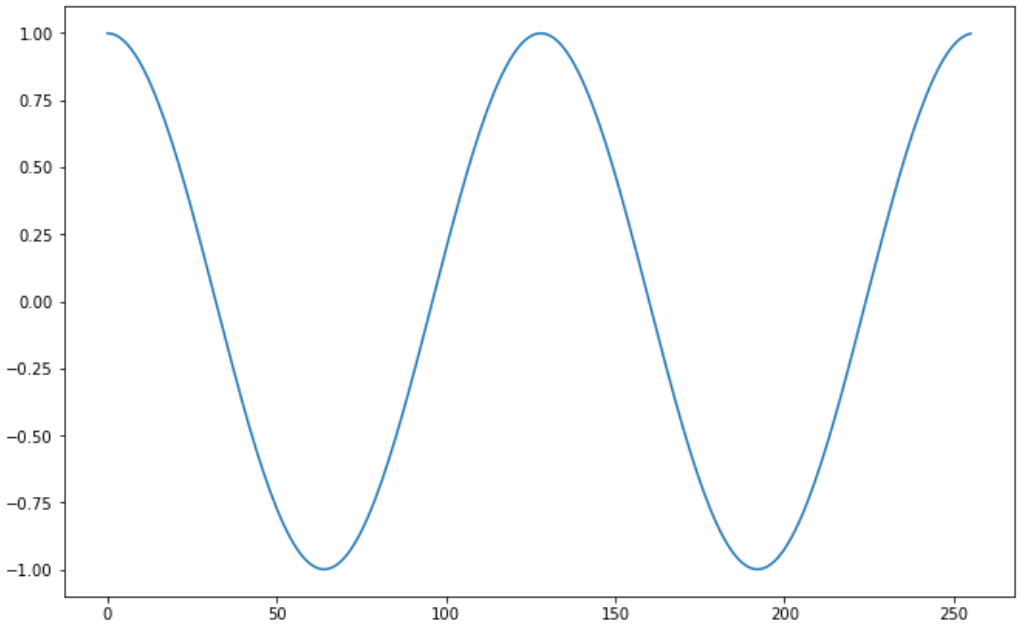
6. *Użyć funkcji gsl_fft_halfcomplex_radix2_inverse do przeprowadzenia odwrotnej transformaty. Narysować wykres otrzymanej funkcji. Czym różni się ona od wyjściowego sygnału?*

Odwrotna transformata Fouriera:

```
inverse_furier = np.fft.ifft(filter_data)
```

Wykres:

```
plt.figure(figsize=(11,7))
plt.plot(inverse_furier.real)
plt.show()
```



Otrzymałem wersję bez szumów funkcji cosinus. Funkcja już nie zawiera w sobie „zanieczyszczeń” z poprzedniej wersji.