```
import matplotlib.pyplot as plt
import math
from math import sin, cos, pi, sqrt, log
from cmath import exp
import time
#czestotliwosc probkowania
f_s = 8000
#przesuniecie w fazie
phi = pi/120
#funkcja z tabeli
funkcja_x = lambda t: sin(2*pi*f_s*t*cos(3*pi*t) + t*phi)

#sekundy
T = 0.5

def generate_signal(funkcja, f_s, T):
  ''' Funkcja generujaca sygnal dla funkcji: funkcja, czestotliwosci: f
_s i czasu trwania T'''
  return [funkcja(n/f_s) for n in range(int(T*f_s))]

#wygererowany sygnal
x = generate_signal(funkcja_x, f_s, T)

#funkcja y z tabeli
funkcja_y = lambda t: (2*t*sin(0.5*t*pi) + 1.5)*cos(9*pi*t+pi*t)
y = generate_signal(funkcja_y, f_s, T)

#funkcja z z tabeli
funkcja_z = lambda t: funkcja_x(t)*funkcja_y(t)+ abs(funkcja_x(t)+2)*(f
unkcja_y(t)**2 + 0.32)
z = generate_signal(funkcja_z, f_s, T)

#funkcja v z tabeli
funkcja_v = lambda t: sqrt(abs(funkcja_x(t)*funkcja_z(t)+10))*(abs(funk
cja_y(t) + 1.2))*sin(2*pi*t)
v = generate_signal(funkcja_v, f_s, T)

#czas trwania sygnalu wynikajacy ze wzoru funkcji u
T_u = 3.1

def funkcja_u(t):
  '''Definicja funkcji u na przedzialach'''
  if t<0 or t >= 3.1:
    raise Exception("Wrong time!!")
  if t < 1.2:
    return (-t**2 + 0.5)*sin(30*pi*t)*log(t**2+1,2)
  if t < 2:
    return (1/t)*0.8*sin(24*pi*t)-0.1*t
  if t < 2.4:
```

```
    abs(sin(2*pi*t*2))**0.8
  if t < 3.1:
    return 0.23*sin(20*pi*t)*sin(12*pi*t)


#funkcja u z tabeli
u = generate_signal(funkcja_u, f_s, T_u)

#czestotliosc dla b1, b2, b3
f_s = 22.05
#czas dla b1, b2, b3
T = 1

from functools import partial

def funkcja_b_k(t, k):
  '''definicja funkcji b_k dla k=1,2,3'''
  H = [2,20,40]
  return sum([(cos( 2*pi *h*t + sin(6*pi*t)))*((-
1)**h)/(3*h**2) for h in range(1, H[k-1])])


#sygnaly b1 do b3
b_1 = generate_signal(partial(funkcja_b_k,k=1), f_s, T_u)
b_2 = generate_signal(partial(funkcja_b_k,k=2), f_s, T_u)
b_3 = generate_signal(partial(funkcja_b_k,k=3), f_s, T_u)
```

Zad 1

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i \cdot \frac{2\pi \cdot k \cdot n}{N}}, \qquad k = 0, \ldots, N-1,$$

gdzie:

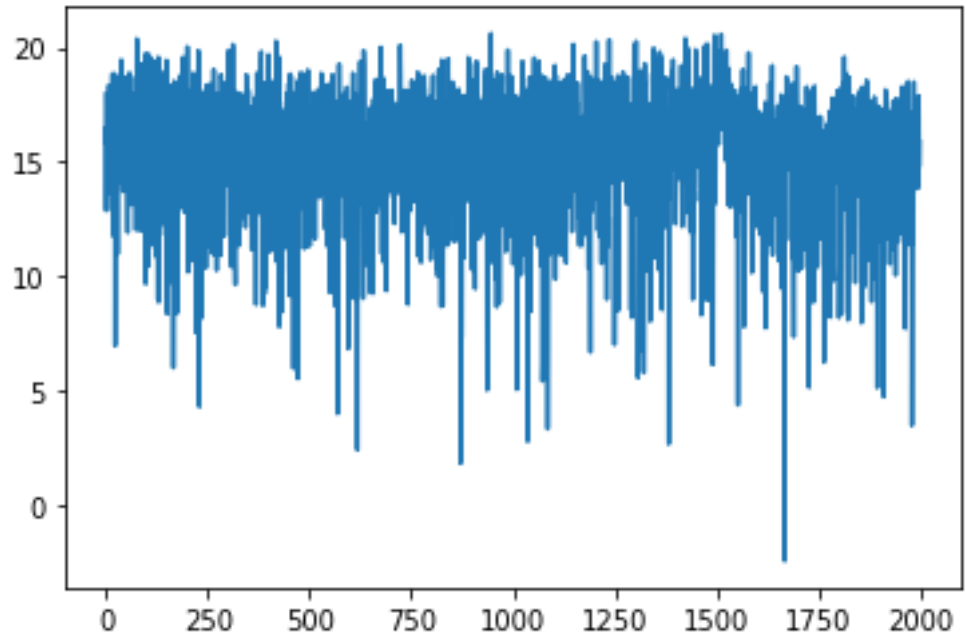
$X(k)$ - reprezentacja w dziedzinie częstotliwości (wektor liczb zespolonych postaci $a + ib$),
$x(n)$ - próbki reprezentujące sygnał w dziedzinie czasu,
i - jednostka urojona ($i^2 = -1$),
N - liczba próbek sygnałów w dziedzinie czasu i częstotliwości.

```
def DFT(x):
  '''Funkcja realizujaca Dyskretna Transformate Fouriera dla sygnalu
wejsciowego: x. Funkcja oblicza dft tylko dla polowy sygnalu ze wzgledu
 na symetrie'''
  #dlugosc sygnalu
  N = len(x)
  #ze wzgledu na symetrie DFT
  if N % 2 == 0:
    dft = [sum([x[n] * exp(complex(0,-
(2*pi*k*n)/N)) for n in range(N)]) for k in range(int(N/2))]
    #kopia pierwszej polowy ciagu
```

```
    dft.extend(reversed(dft))
  else:
    dft = [sum([x[n] * exp(complex(0,-
(2*pi*k*n)/N)) for n in range(N)]) for k in range(int((N+1)/2))]
    #kopia pierwszej polowy ciagu
    dft.extend(reversed(dft[:-1]))
  return dft

#czas obliczen dla DFT i sygnalu x
#rozpoczecie liczenia czasu
start = time.time()

#obliczenia
DFT_x = DFT(x)

#zatrzymanie liczenia czasu
end = time.time()
#obliczenie roznicy w czasie
x_dft_time = end - start
print(x_dft_time)
#wynik wydruku: 7.458267450332642
```

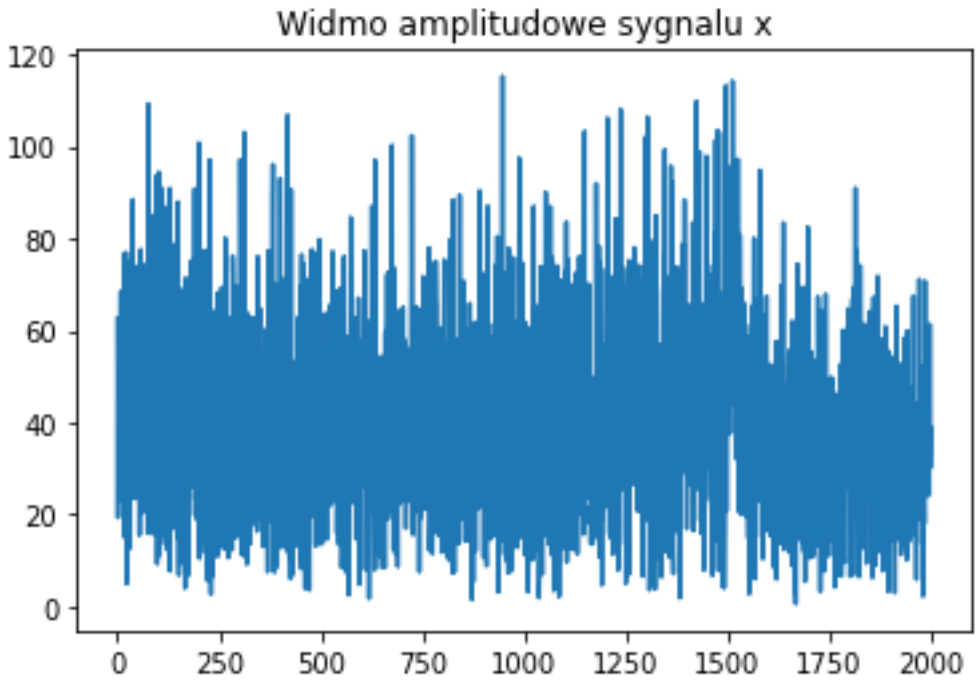
Zad 2

```
def widmo_amplitudowe(X):
  '''Obliczamy widno amplitudowe dla ciagu X'''
  return [ sqrt(X[k].real**2 + X[k].imag**2) for k in range(len(X))]


#dlugosc ciagu
N = len(x)
#widmo dla ciagu
Widmo_x = widmo_amplitudowe(DFT_x[:int(N/2 - 1)])


#wartosc amplitudy w sklai decybelowej
M_prim = [10*log(Widmo_x[k],10) for k in range(int(N/2 - 1))]


#skala czestotliwosci
skala_czestotliwosci = [k*(f_s/N) for k in range(int(N/2 - 1))]


#wykres widma amplitudowego
plt.plot(range(int(N/2 - 1)),M_prim)
```

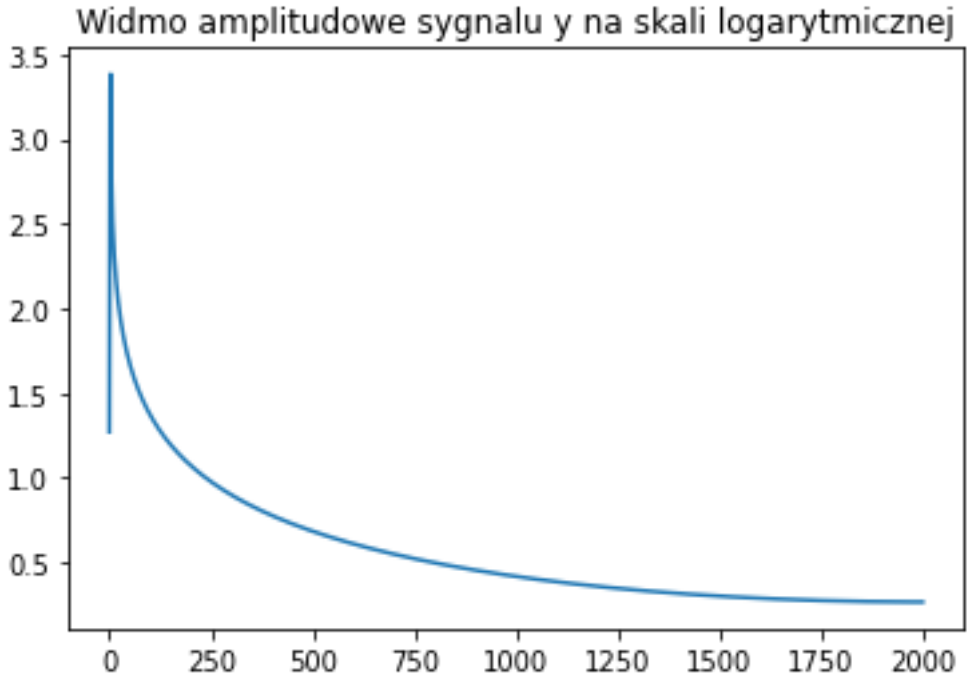



```
#czas obliczen dla DFT i sygnalu y
start = time.time()

DFT_y = DFT(y)

end = time.time()
y_dft_time = end - start
print("y", y_dft_time)
#czas obliczen dla DFT i sygnalu z
start = time.time()

DFT_z = DFT(z)

end = time.time()
z_dft_time = end - start
print("z",z_dft_time)
#czas obliczen dla DFT i sygnalu v
start = time.time()

DFT_v = DFT(v)

end = time.time()
v_dft_time = end - start
print("v",v_dft_time)
#czas obliczen dla DFT i sygnalu u
start = time.time()

DFT_u = DFT(u)

end = time.time()
u_dft_time = end - start
```

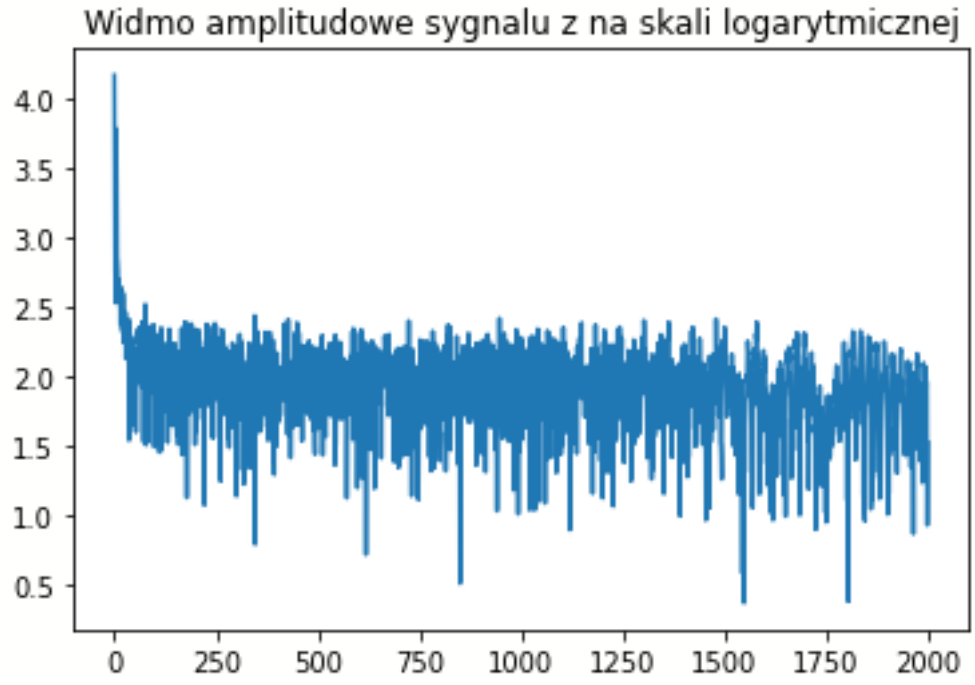
```
print("yu",u_dft_time)
#czas obliczen dla DFT i sygnalu b_1
start = time.time()

DFT_b1 = DFT(b_1)

end = time.time()
b1_dft_time = end - start
print("b_1",b1_dft_time)
#czas obliczen dla DFT i sygnalu b_2
start = time.time()

DFT_b2 = DFT(b_2)

end = time.time()
b2_dft_time = end - start
print("b_2",b2_dft_time)
#czas obliczen dla DFT i sygnalu b_3
start = time.time()

DFT_b3 = DFT(b_3)

end = time.time()
b3_dft_time = end - start
print("b_3",b3_dft_time)

#czasy wydrukowane przez program
y 5.45585298538208
z 5.350531101226807
v 5.4098687171936035
yu 211.88939332962036
b_1 0.0016765594482421875
b_2 0.001634359359741211
b_3 0.0016238689422607422



#widma amplitudowe dla sygnalow x, y, z, v, u, b_1, b_2, b_3
Widmo_y = widmo_amplitudowe(DFT_y[:int(len(DFT_y)/2 - 1)])
Widmo_z = widmo_amplitudowe(DFT_z[:int(len(DFT_z)/2 - 1)])
Widmo_v = widmo_amplitudowe(DFT_v[:int(len(DFT_v)/2 - 1)])
Widmo_u = widmo_amplitudowe(DFT_u[:int(len(DFT_u)/2 - 1)])
Widmo_b1 = widmo_amplitudowe(DFT_b1[:int(len(DFT_b1)/2 - 1)])
Widmo_b2 = widmo_amplitudowe(DFT_b2[:int(len(DFT_b2)/2 - 1)])
Widmo_b3 = widmo_amplitudowe(DFT_b3[:int(len(DFT_b3)/2 - 1)])


#wykres widma
plt.plot(range(len(Widmo_x)), Widmo_x)
plt.title("Widmo amplitudowe sygnalu x")
```



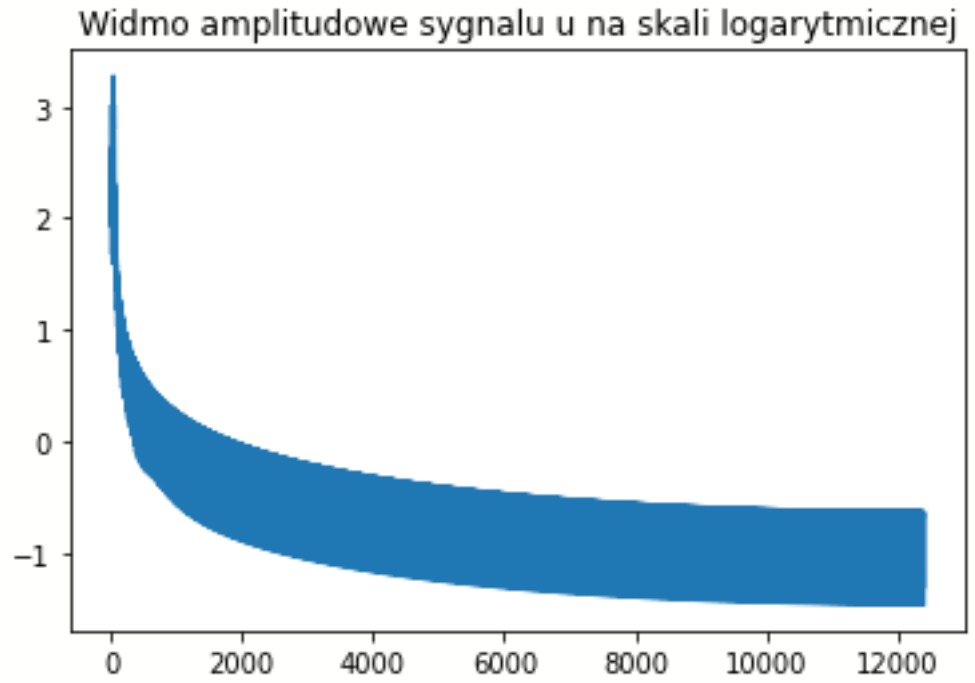

```
#funkcja pomocnicza dla skali logarytmicznej
def table_log(seq):
  return [log(el,10) for el in seq]


plt.plot(range(len(Widmo_y)), table_log(Widmo_y))
plt.title("Widmo amplitudowe sygnalu y na skali logarytmicznej")
```



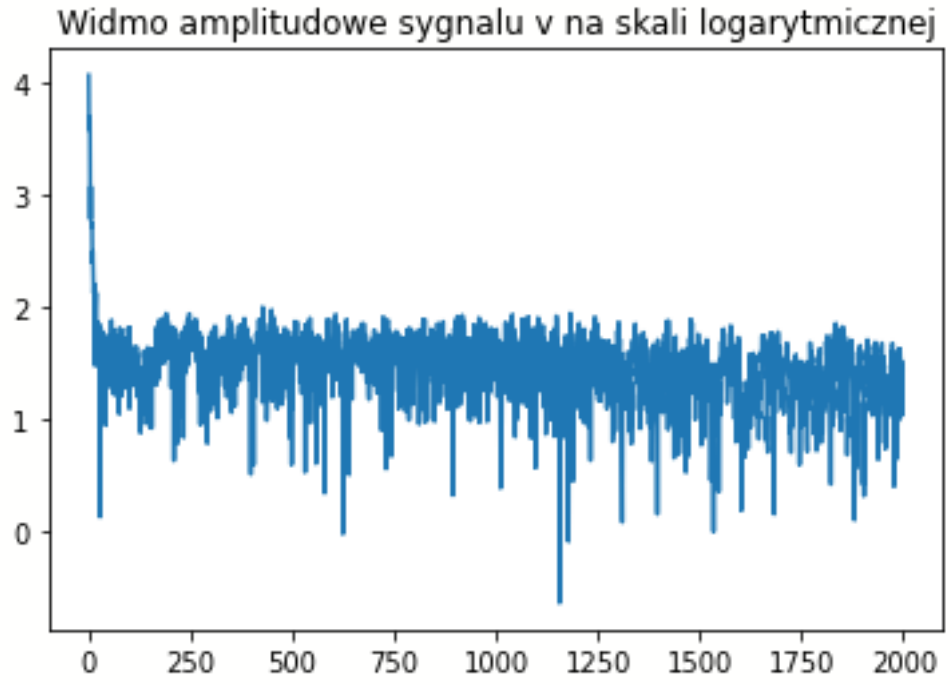
```
plt.plot(range(len(Widmo_z)), table_log(Widmo_z))
plt.title("Widmo amplitudowe sygnalu z na skali logarytmicznej")
```



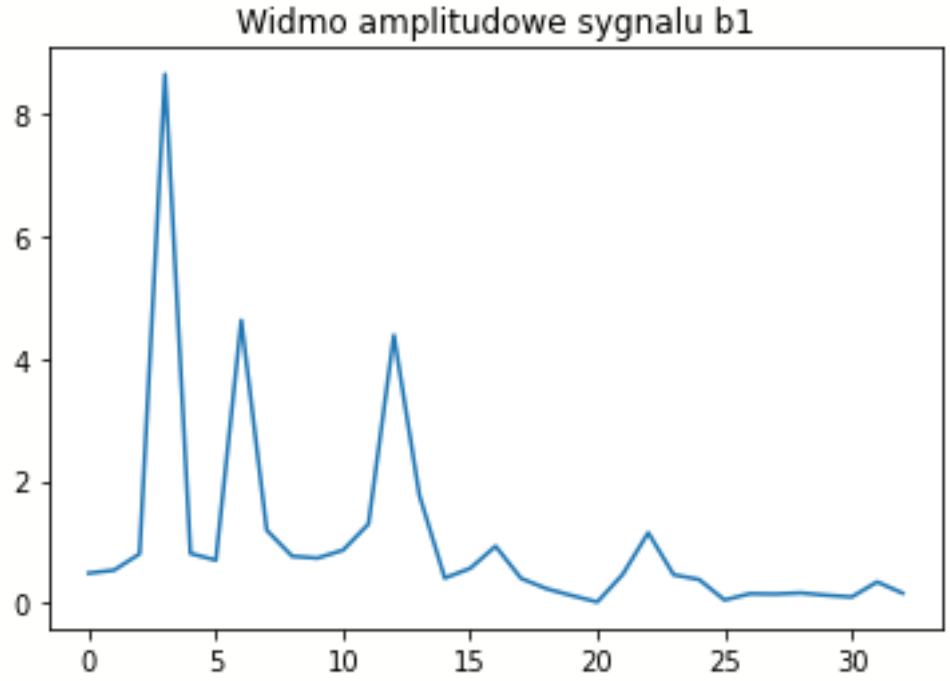

```
plt.plot(range(len(Widmo_u)), table_log(Widmo_u))
plt.title("Widmo amplitudowe sygnalu u na skali logarytmicznej")
```



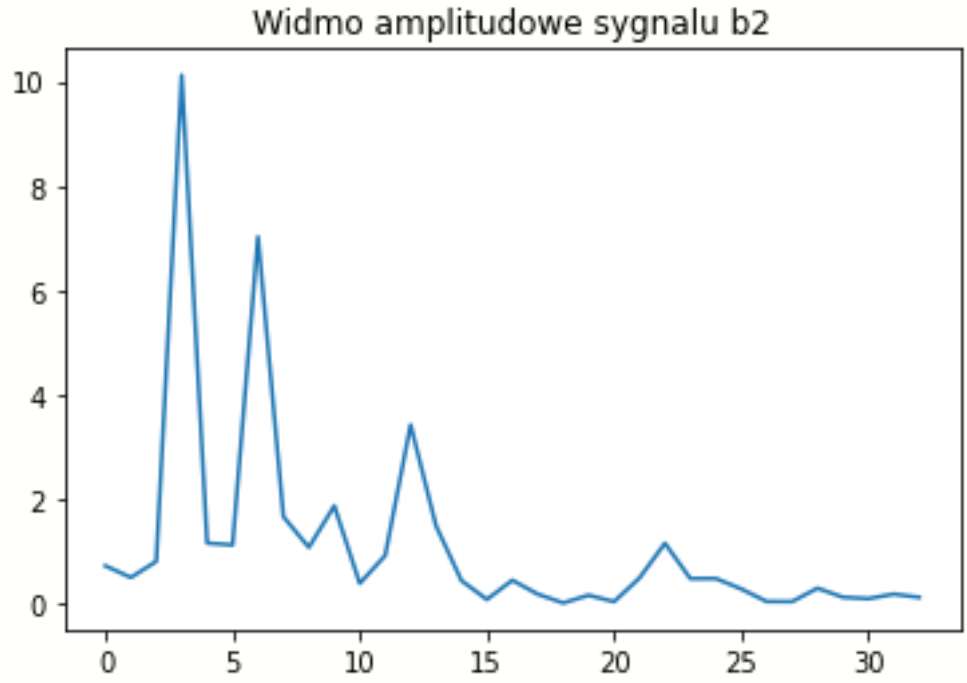
```
plt.plot(range(len(Widmo_v)), table_log(Widmo_v))
plt.title("Widmo amplitudowe sygnalu v na skali logarytmicznej")
```




```
plt.plot(range(len(Widmo_b1)), Widmo_b1)
plt.title("Widmo amplitudowe sygnalu b1")
```



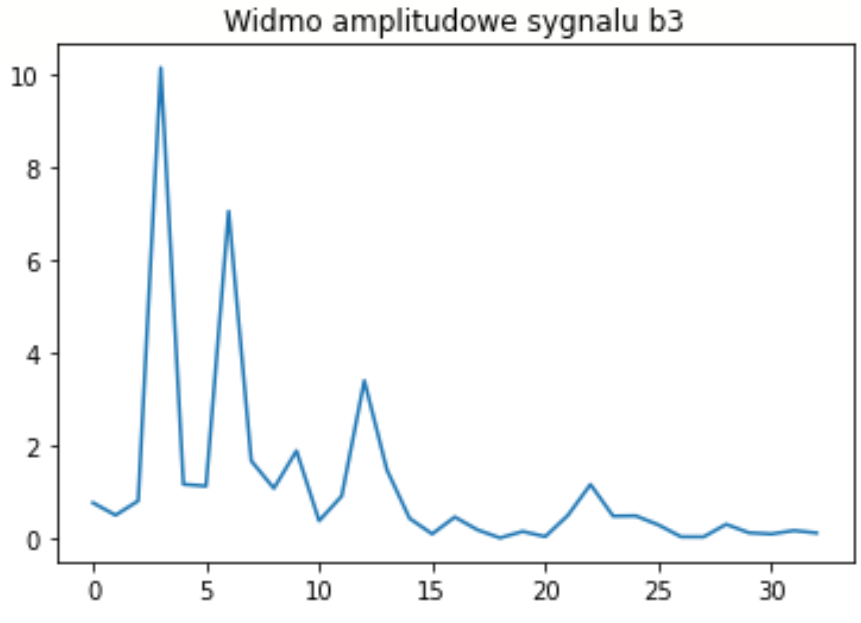
```
plt.plot(range(len(Widmo_b2)), Widmo_b2)
plt.title("Widmo amplitudowe sygnalu b2")
```



2. Ćwiczenia

1. Proszę zaimplementować przekształcenie DFT na podstawie wzoru przedstawionego w punkcie 1.
2. Dla uzyskanej reprezentacji w dziedzinie częstotliwości $X(k)$, $k = 0, \ldots, N/2 - 1$:
   - obliczyć widmo amplitudowe: $M(k) = \sqrt{Re[X(k)]^2 + Im[X(k)]^2}$,
   - wartości amplitudy przedstawić w skali decybelowej: $M'(k) = 10 \cdot \log_{10} M(k)$,
   - wyznaczyć skalę częstotliwości: $f_k = k \cdot \frac{f_s}{N}$, gdzie f_s - częstotliwość próbkowania syganału,
   - wykreślić wykres widma amplitudowego $M'(k)$ (f_k oznaczają częstotliwości prążków widma).
3. Dla sygnałów uzyskanych na poprzednich laboratoriach proszę obliczyć DFT i wygenerować wykresy widm amplitudowych. Należy tak dobrać skale osi poziomych i pionowych (liniowe lub logarytmiczne) aby jak najwięcej prążków widma było widocznych na wykresie.
4. Wykonać obliczenia dla sygnałów z poprzednich laboratoriów z wykorzystaniem szybkiej transformaty Fouriera (FFT) zamiast DFT. Porównać czasy obliczeń dla poszczególnych sygnałów oraz sumaryczne czasy obliczeń. Wyniki zestawić w tabeli.

```
plt.plot(range(len(Widmo_b3)), Widmo_b3)
plt.title("Widmo amplitudowe sygnalu b3")
```



FFT Scipy

```
#Szybkie Transofrmaty Fouriera dla sygnalow x,y,z,v,u,b1,b2,b3 z uzycie
m biblioteki scipy

from scipy.fft import fft

start = time.time()

FFT_x = fft(x)

end = time.time()
x_fft_time = end - start
print(x_fft_time)
start = time.time()

FFT_y = fft(y)

end = time.time()
y_fft_time = end - start
print(y_fft_time)
start = time.time()

FFT_z = fft(z)

end = time.time()
z_fft_time = end - start
print(z_fft_time)
start = time.time()

FFT_v = fft(v)

end = time.time()
v_fft_time = end - start
print(v_fft_time)
start = time.time()

FFT_u = fft(u)

end = time.time()
u_fft_time = end - start
print(u_fft_time)
start = time.time()

FFT_b1 = fft(b_1)

end = time.time()
b1_fft_time = end - start
```

```
print(b1_fft_time)
start = time.time()

FFT_b2 = fft(b_2)

end = time.time()
b2_fft_time = end - start
print(b2_fft_time)
start = time.time()

FFT_b3 = fft(b_3)

end = time.time()
b3_fft_time = end - start
print(b3_fft_time)
#czasy wydrukowane przez program
0.0006148815155029297
0.0005040168762207031
0.0008056163787841797
0.0006766319274902344
0.0021691322326660156
0.0001316070556640625
8.726119995117188e-05
9.179115295410156e-05

#podsumowanie wynikow
import numpy as np
import pandas as pd
czasy = pd.DataFrame(np.array([["DFT",x_dft_time,y_dft_time,z_dft_time,
u_dft_time,v_dft_time,b1_dft_time,b2_dft_time,b3_dft_time ],["FFT",x_ff
t_time,y_fft_time,z_fft_time,u_fft_time,v_fft_time,b1_fft_time,b2_fft_t
ime,b3_fft_time]]), columns=["transformata","x","y","z","u","v","b1","b
2","b3"]).set_index('transformata').astype(float)
czasy
```

transformata	x	y	z	u	v	b1	b2	b3
DFT	7.458267	5.455853	5.350531	211.889393	5.409869	0.001677	0.001634	0.001624
FFT	0.000615	0.000504	0.000806	0.002169	0.000677	0.000132	0.000087	0.000092

```
#Laczne czasy w sekundach
czasy.sum(axis=1)

transformata
DFT    235.568848
FFT      0.005081
dtype: float64
```