Sygnaly_Lab2_145204

December 15, 2021

1 ZADANIE 2

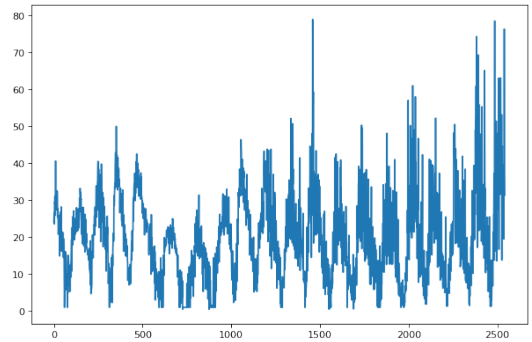
[102]:
```
import matplotlib.pyplot as plt
from pylab import *
from numpy import *
import math

from ipywidgets import *

import warnings
warnings.filterwarnings('ignore')
```

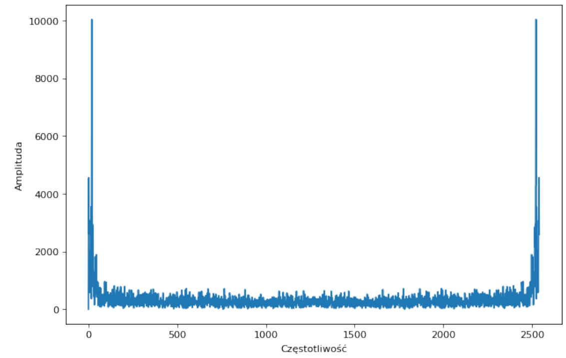
1.0.1 1. Plik spots.txt zawiera wartości aktywności Słońca w kolejnych miesiącach. Wykreśl ten sygnał oraz jego spektrum. Za pomocą FFT, oblicz częstotliwość cyklu aktywności słonecznej. Przydatne mogą być następujące konstrukcje:

- array = np.genfromtxt('spots.txt')
- x = max(array)

[103]:
```
array = np.genfromtxt('spots.txt')
len_array = len(array)

signal = array - np.mean(array)
signal = abs(fft.fft(signal))

fig = plt.figure(figsize=(20, 6), dpi=80)

plot = fig.add_subplot(121)
plot.plot(array);

plot = fig.add_subplot(122)
plot.set_xlabel('Częstotliwość')
plot.set_ylabel('Amplituda')
plot.plot(signal);

freq_sun = fft.fftfreq(len(np.genfromtxt('spots.txt')), 1/12)
print('Częstotliwość cyklu aktywności słonecznej to: ', 1/fft.
 ↪fftfreq(len_array, 1/12)[np.argmax(signal)])
```

```
Częstotliwość cyklu aktywności słonecznej to:  11.149122807017545
```

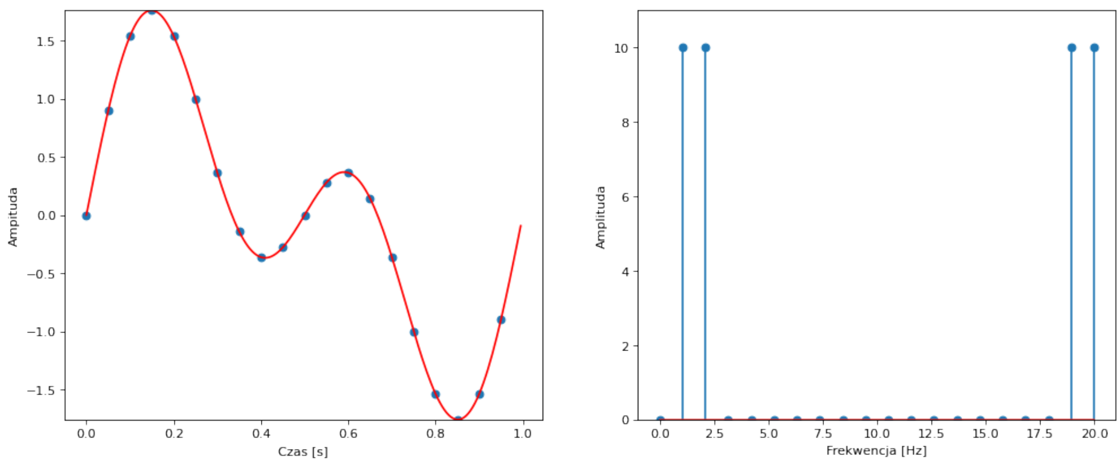




1.0.2 2. Proste filtrowanie. Wykreśl sygnał sin(2*pi*t) + sin(4*pi*t), T=1s, w=20Hz. Za pomocą FFT, przekształć sygnał do dziedziny częstotliwości. Następnie usuń składowe o częstotliwości 2Hz. Tak zmodyfikowany sygnał przekształć do dziedziny czasu i wykreśl go.

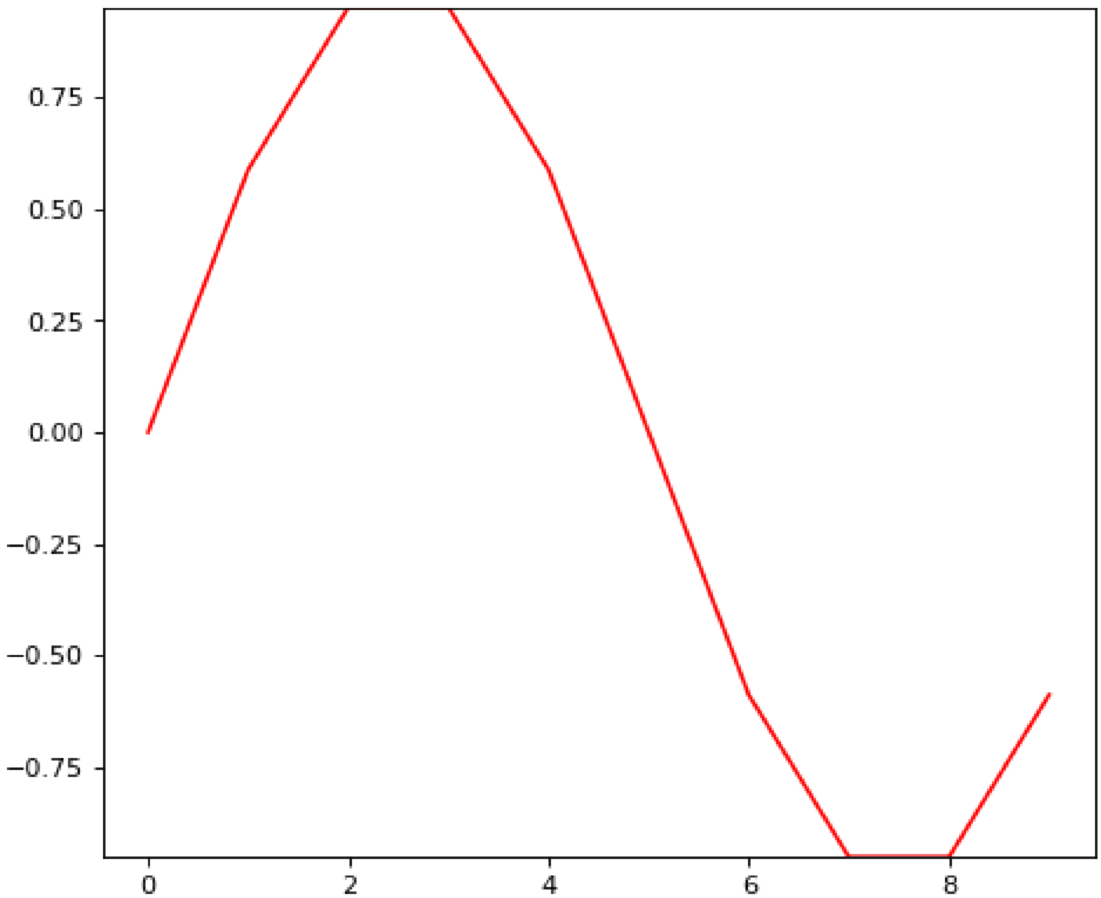
[104]:
```
def wykresy_sygnalow_funkcja(f, F = 2.0, LP = 1, w = 40):
  #--- Definiujemy sygnal wejsciowy
  # A = 1        # Amplituda sygnalu
  # F = 2.0      # Czestotliwosc sygnalu [Hz]
  T = 1/F      # Okres sygnalu [s]

  #--- Probkujemy sygnal
  # LP = 1       # Liczba analizowanych pełnych okrebsów sygnalu (okresow)
  # w = 40       # Częstotliwość probkowania [Hz]
  TW = 1/w     # Okres probkowania [s] (co ile sekund pobieramy próbkę)

  t = np.arange(0, LP*T, TW) # Momenty, w których pobieramy próbki (oś OX)
  n = len(t)                 # Liczba próbek

  signal = f(t)

  #--- Rysujemy sygnał (niebieskie kółka)
  fig = plt.figure(figsize=(15, 6), dpi=80)
  ax = fig.add_subplot(121)
  ax.plot(t, signal, 'o')

  #--- Rysujemy sygnał przed spróbkowaniem (dla wizualizacji)
  base_t = np.arange(0, LP * T, 1/200)
  base_signal = f(base_t)
  ax.plot(base_t, base_signal, linestyle='-', color='red')
  ax.set_ylim([min(base_signal), max(base_signal)])
  ax.set_xlabel("Czas [s]")
```

```
  ax.set_ylabel("Ampituda")


  #--- Wykonujemy FFT
  signal1 = fft.fft(signal)
  signal1 = abs(signal1) # moduł

  #--- Rysujemy FFT
  ax = fig.add_subplot(122)
  ymax = max(signal1)
  ax.set_ylim([0.0, max(1.1*ymax, 3.0)])
  ax.set_xlabel("Frekwencja [Hz]")
  ax.set_ylabel("Amplituda")

  freqs = np.linspace(0, w, len(signal1))
  stem(freqs, signal1, '-*', use_line_collection=True);



wykresy_sygnalow_funkcja(lambda t: (sin(2*pi*t)+sin(4*pi*t)), F = 1, LP = 1, w = 20)
```



[105]:
```
def fftfreq2(N, w=1):
    if N == 1: return [0]
    n = N if N % 2 == 0 else N - 1
    q = -1 if N % 2 == 0 else 0
    a = np.arange(0, n/2+q+1, 1)
    b = np.arange(-n/2, 0, 1)
    return np.hstack((a, b)) / (n*w)

def wykres_bez_frekwencji(func, x1=0, LP=1, T=1, w=20, removed_freq=[]):
```

```
    x2 = LP*T
    xs1 = np.arange(x1, x2, 1 / w)
    ys1 = func(xs1)

    N = len(ys1)

    xs2 = fftfreq2(N, 1 / w)
    ys2 = fft.fft(ys1)

    ys2 = ys2[np.where(xs2>=0)]
    xs2 = xs2[np.where(xs2>=0)]
    ys2[removed_freq]=0
    ys3 = np.fft.ifft(ys2)

    fig = plt.figure(figsize=(15, 6), dpi=80)
    ax = fig.add_subplot(121)
    ax.plot(xs2, ys3, linestyle='-', color='red');
    ax.set_ylim([min(ys3), max(ys3)])
    plt.show()


func = (lambda t: sin(2*pi*t) + sin(4*pi*t))
wykres_bez_frekwencji(func, T=1, w=20, removed_freq=[2])
```

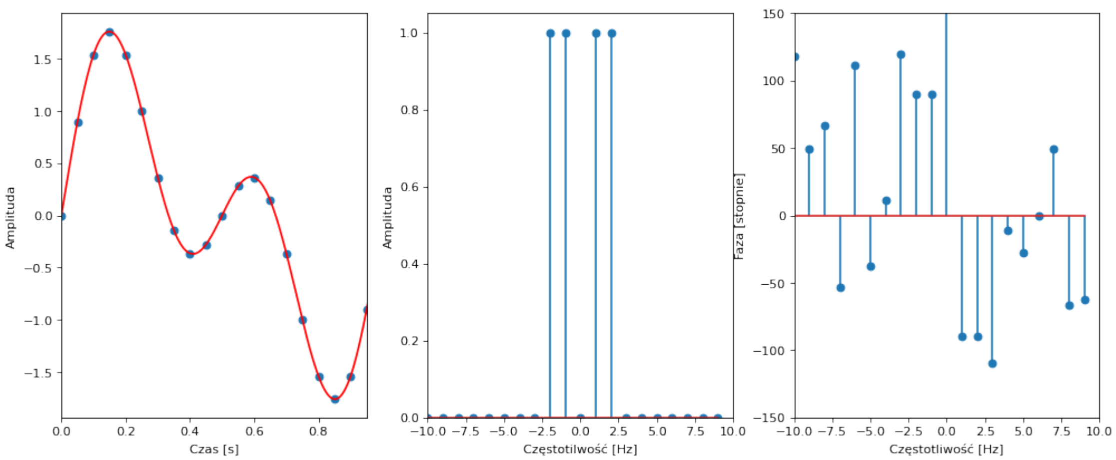


1.0.3 3. Informacja o fazie. Wykreśl sygnał sin(2*pi*t) + sin(4*pi*t), T=1s, w=20Hz. Tym razem oprócz spektrum, wykreśl wykres z informacją o fazie poszczególnych częstotliwości (faza = arg(z)), gdzie z=a+bi. To samo wykonaj dla sin(2*pi*t) + cos(4*pi*t). Porównaj otrzymane wykresy. Przydatne funkcje (działają także dla tablic):

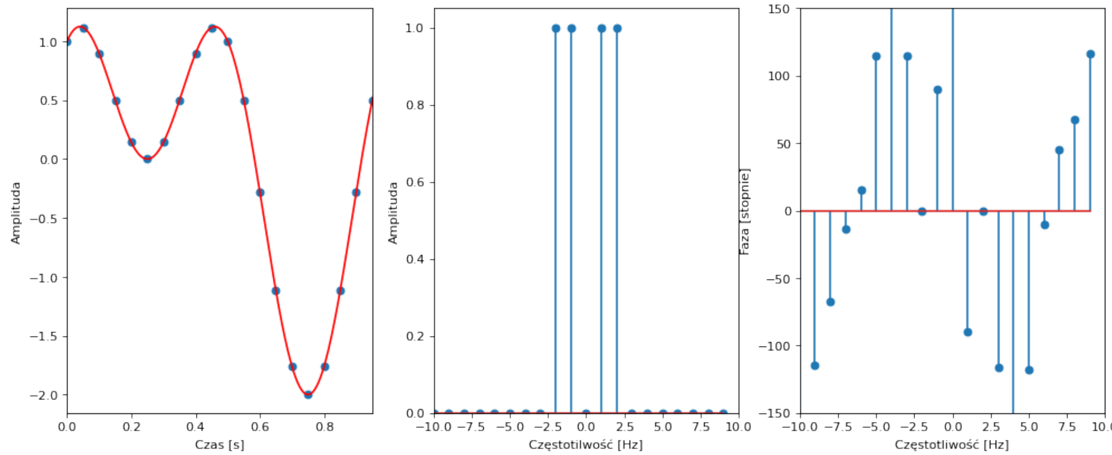
- atan2(z)
- z.imag
- z.real

[106]:
```
def wykres_faza(func, x1=0, LP=1, T=1, w=20):
    x2 = LP*T
    xs1 = np.arange(x1, x2, 1/w)
    ys1 = func(xs1)
    N = len(ys1)

    xs2 = np.linspace(x1, x2, 200)
    ys2 = [func(x) for x in xs2]
```

```
    xs3 = fftfreq2(N, 1 / w)
    ys3 = np.abs(fft.fft(ys1)) * 2 / N
    ys4 = angle(fft.fft(ys1), deg=True)

    fig = plt.figure(figsize=(15, 6), dpi=80)
    ax = fig.add_subplot(131)
    ax.plot(xs1, ys1, 'o');
    ax.plot(xs2, ys2, '-', color='red');
    ax.set_xlim(0, xs1[-1])
    ax.set_xlabel('Czas [s]')
    ax.set_ylabel('Amplituda')

    ax = fig.add_subplot(132)
    plt.stem(xs3, ys3, '-*')
    ax.set_xlim(-w/2, w/2)
    ax.set_ylim(0)
    ax.set_xlabel('Częstotilwość [Hz]')
    ax.set_ylabel('Amplituda')

    ax = fig.add_subplot(133)
    plt.stem(xs3, ys4, '-*');
    ax.set_xlim(-w/2, w/2)
    ax.set_ylim(-150,150)
    ax.set_xlabel('Częstotliwość [Hz]')
    ax.set_ylabel('Faza [stopnie]')
    plt.show()


func = lambda t: sin(2*pi*t) + sin(4*pi*t)
wykres_faza(func, T=1, w=20)
```



[107]:
```
func = lambda t: sin(2*pi*t) + cos(4*pi*t)
wykres_faza(func, T=1, w=20)
```

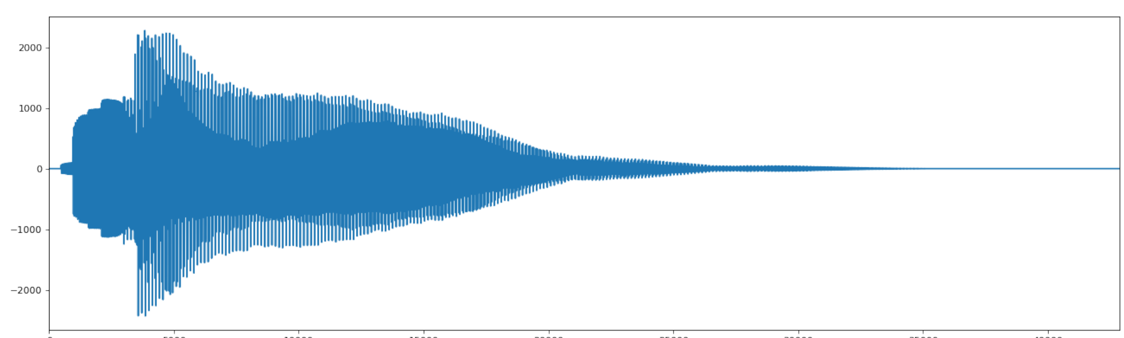


1.0.4 4. Wczytaj plik err.wav. Wykreśl jego spektrum. Spróbuj także skali logarytmicznej. Określ dominujące w sygnale częstotliwości. Przydatne:

- import scipy.io.wavfile
- w, signal = scipy.io.wavfile.read('err.wav')
- signal = [s[0] for s in signal] #Tylko pierwszy kanał
- yscale('log')
- spectrum[::10] # co 10ty element

[108]:
```
import scipy.io.wavfile
w, signal = scipy.io.wavfile.read('err.wav')
signal = [s[0] for s in signal]
fig = plt.figure(figsize=(20, 6), dpi=80)
plt.plot(signal)
plt.xlim(0, len(signal))
print(w, 'Hz')
```

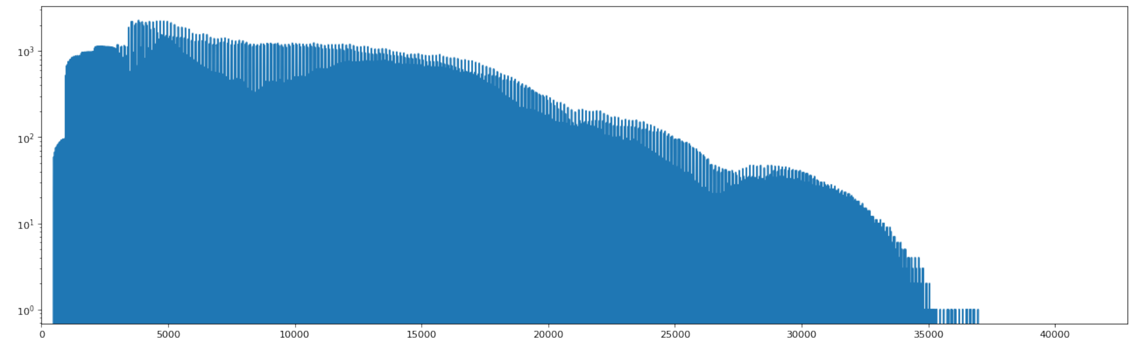
```
44100 Hz
```



[109]:

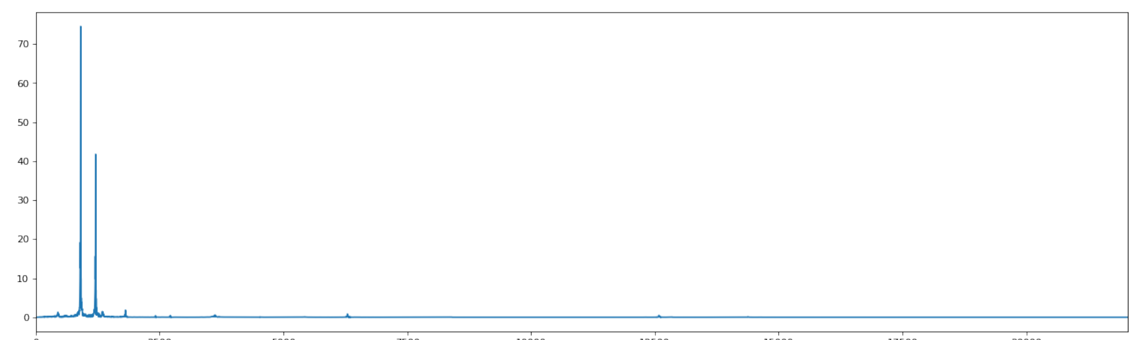
```
fig = plt.figure(figsize=(20, 6), dpi=80)
yscale('log')
plt.plot(signal)
plt.xlim(0, len(signal))
print(w, 'Hz')
```

```
44100 Hz
```



[110]:

```
import plotly.graph_objects as go

signal = abs(fft.fft(signal)) / len(signal) / 2

fig = plt.figure(figsize=(20, 6), dpi=80)
plt.xlim(0, 44100/2)
plt.plot(signal);
```

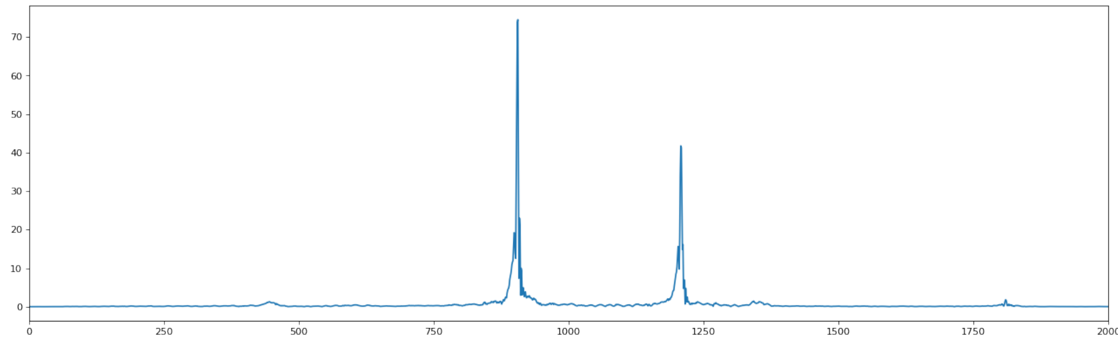


[111]:

```
fig = plt.figure(figsize=(20, 6), dpi=80)
plt.xlim(0, 2000)
plt.plot(signal)
```

```
print(np.argmax(signal), 'Hz')
```

906 Hz



1.0.5 5. Za pomocą fft można efektywnie mnożyć duże liczby (lub np. wielomiany). Sprawdź poniższ obliczenia. Jaka jest złożoność obliczeniowa następującej operacji? Uwaga: aby wykonać obliczenia dla większych cyfr, trzeba zaprogramować "promocję" np. [1,2]*[0,6]=[0,6,12], co znaczy [0,6+1,2]=[0,7,2].

[112]:
```
a=[3,2,3,4]
b=[4,2,3]
print(3234*423)

# Należy odpowiednio dobrać "padding". W przeciwnym wypadku na końcu wyniku␣
↪pojawią się dodatkowe zera.
A = fft.fft(a,6)
B = fft.fft(b,6)
C = A*B
c = abs(ifft(C))

print(c)

c_len = len(c) - 1

while c_len >= 0:
  while c[c_len] > 9:
    c[c_len]-=10
    if c_len == 0 and flag:
      c.resize(c.shape[0]+1)
      for i in reversed(range(1, len(c))):
        c[i] = c[i-1]
      c[0] = 0
      c_len += 1
```

```python
    c[c_len-1]+=1
  c_len -= 1

print(c)
```

```
1367982
[12. 14. 25. 28. 17. 12.]
[1. 3. 6. 7. 9. 8. 2.]
```