# Ćwiczenie4

December 19, 2020

**Wydział Informatyki Politechniki Białostockiej Przetwarzanie Sygnałów i Obrazów – Pracownia specjalistyczna**

**Ćwiczenie 4. Analiza widmowa sygnałów**

**Imię i nazwisko studenta: Sylwia Mościcka.**

**Grupa: PS 5**

**Data realizacji ćwiczenia 24.11.2020r.**

## Zadanie 4.1

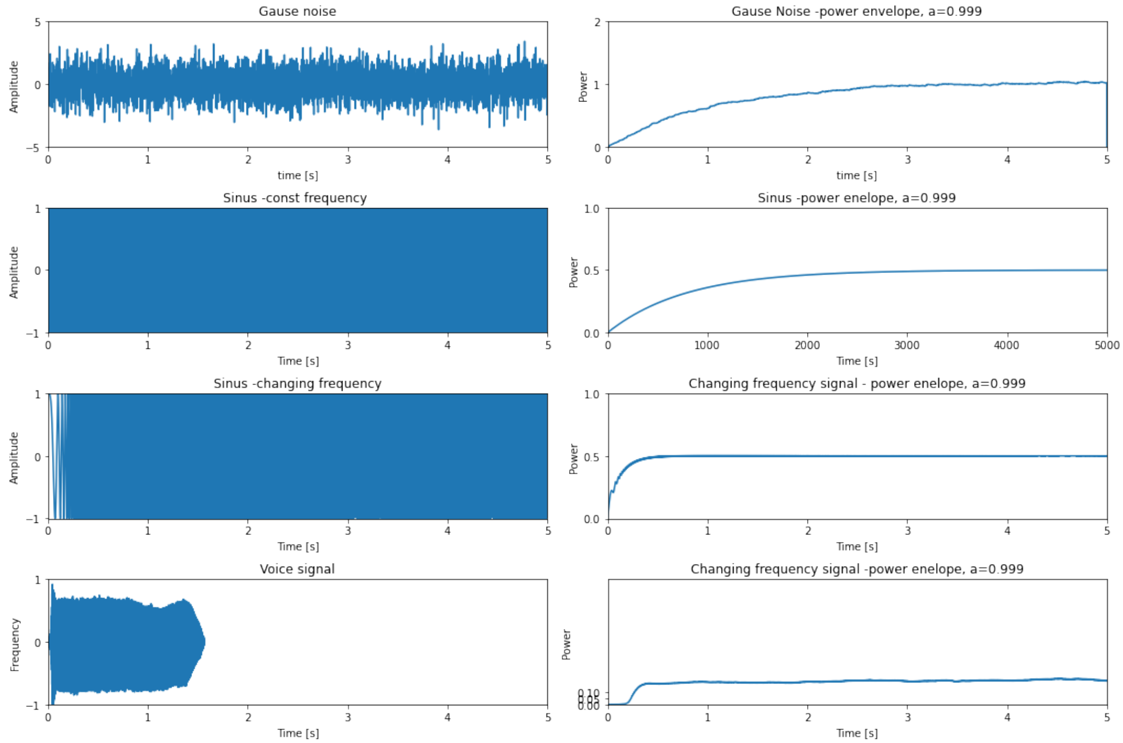
**Wygenerować/nagrać następujące sygnały (dł. 5s każdy, tempo próbkowania fs = 8kHz): szum gaussowski, sygnał sinusoidalny o stałej częstotliwości 1kHz, sygnał o zmiennej częstotliwości w zakresie od 0Hz (0s) do 1kHz (5s) (patrz funkcja chirp) oraz sygnał mowy. Następnie, dla każdego z sygnałów wykreślić obwiednię mocy w czasie (dla uzyskania lepszej przejrzystości zamiast funkcji stem można użyć funkcji plot). W celu oszacowania mocy sygnału w czasie Px[n] zastosować uśrednianie wykładnicze/rekursywne, zgodnie ze wzorem: Px[n] = Px[n-1] + (1- )x[n] 2 , gdzie x[n] - n-ta próbka sygnału oraz 0< <1 parametr uśredniający. Sprawdzić, jaki wpływ na obwiednię mocy ma dobór parametru alfa? Co możesz powiedzieć o stacjonarności sygnałów na podstawie kształtu obwiedni?**

[177]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import wave


x = np.linspace(0,5000,5001)
noise = np.random.normal(0,1,5001)
plt.figure(figsize=(15,10))
plt.subplot(4,2,1)
plt.plot(x,noise)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-5,0,5])
plt.ylim(-5,5)
plt.xlabel("time [s]")
plt.ylabel("Amplitude")
```

```
plt.title("Gause noise")
plt.subplot(4,2,2)
alpha = 0.999
y = (1 - alpha) * (noise * noise)
for i in range(0,5000):
    y[i] = y[i] + y[i - 1] * alpha

plt.plot(x,y)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([0,1,2])
plt.ylim(0,2)
plt.xlabel("time [s]")
plt.ylabel("Power")
plt.title("Gause Noise -power envelope, a=0.999")
plt.subplot(4,2,3)
x = np.linspace(0,5000 * 2 * np.pi,5 * 8000)
sin1 = np.sin(x)
plt.plot(x,sin1)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.title("Sinus -const frequency ")
plt.subplot(4,2,4)
alpha = 0.999
y = (1 - alpha) * (sin1 * sin1)
for i in range(0,int(5000 * 2 * np.pi)):
    y[i] = y[i] + y[i - 1] * alpha
plt.plot(x,y)
plt.xticks
(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Sinus -power enelope, a=0.999")
plt.subplot(4,2,5)
x = np.linspace(0,5,5 * 8000)
sin2 = signal.chirp(x, f0 = 0, f1 = 1000, t1 = 5)
plt.plot(x,sin2)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
```

```
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.title("Sinus -changing frequency")
plt.subplot(4,2,6)
alpha=0.999
y=(1 - alpha)*(sin2 * sin2)
for i in range(0,5 * 8000):
    y[i]=y[i]+y[i - 1]*alpha
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Changing frequency signal - power enelope, a=0.999")
plt.subplot(4,2,7)
y= wave.open('klarnet.wav')
signal=y.readframes(-1)
signal= np.frombuffer(signal, dtype = 'int16')
signal=signal/30000
fs=y.getframerate()
time=np.linspace(0, len(signal) / fs, len(signal))
plt.plot(time,signal)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Frequency")
plt.title("Voice signal")
plt.subplot(4,2,8)
alpha=0.999
y=(1-alpha)*(signal*signal)
for i in range(1,len(signal)):
     y[i]=y[i]+y[i-1]*alpha
y=y[:len(x)]
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([0,0.05,0.1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Changing frequency signal -power enelope, a=0.999")
plt.tight_layout()
```

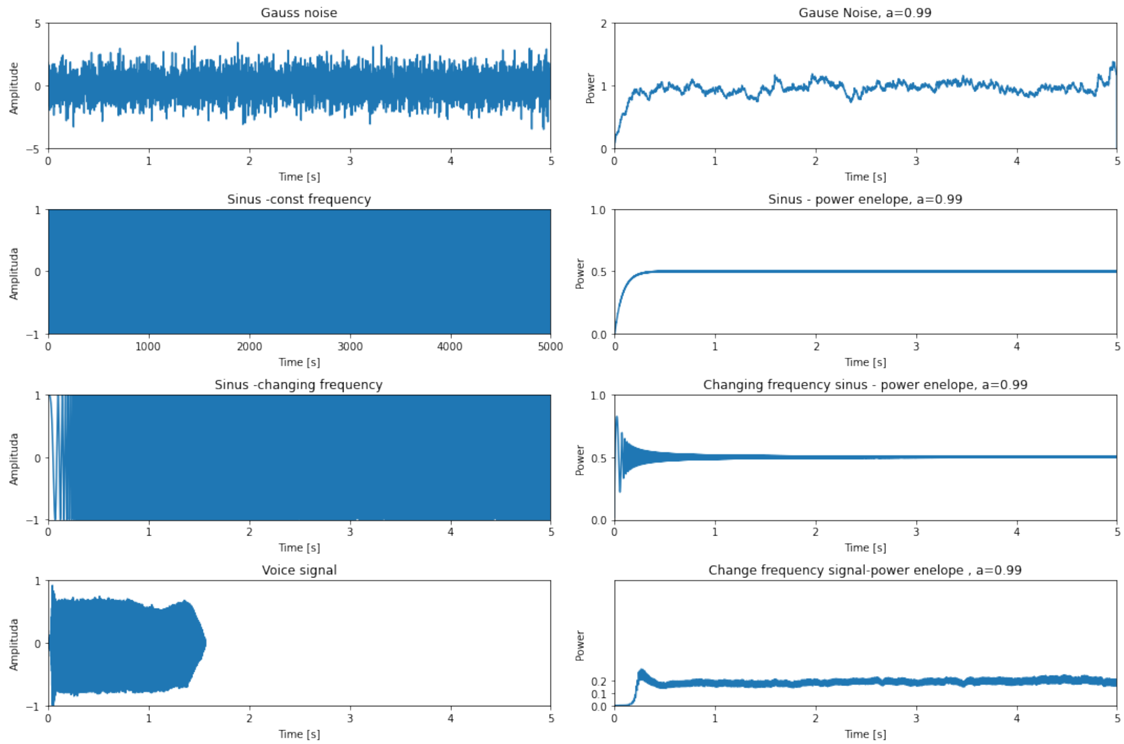
```
plt.show()
```




[178]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import wave
x = np.linspace(0,5000,5001)
noise = np.random.normal(0,1,5001)
plt.figure(figsize=(15,10))
plt.subplot(4,2,1)
plt.plot(x,noise)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-5,0,5])
plt.ylim(-5,5)
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.title("Gauss noise")
plt.subplot(4,2,2)
alpha=0.99
y=(1 - alpha)*(noise * noise)
for i in range(0,5000):
    y[i]=y[i]+y[i - 1]*alpha
```

```
plt.plot(x,y)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([0,1,2])
plt.ylim(0,2)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Gause Noise, a=0.99")
plt.subplot(4,2,3)
x=np.linspace(0,5000 * 2 * np.pi,5 * 8000)
sin1=np.sin(x)
plt.plot(x,sin1)
plt.xticks
(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Sinus -const frequency")
plt.subplot(4,2,4)
alpha=0.99
y=(1-alpha)*(sin1*sin1)
for i in range(0,int(5000*2*np.pi)):
    y[i]=y[i]+y[i-1]*alpha
plt.plot(x,y)
plt.axis([-5, 5, 0, 5]);
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim
(0,5000)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Sinus - power enelope, a=0.99")
plt.subplot(4,2,5)
x=np.linspace(0,5,5*8000)
sin2=signal.chirp(x, f0=0, f1=1000, t1=5)
plt.plot(x,sin2)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Sinus -changing frequency")
plt.subplot(4,2,6)
```

```
alpha=0.99
y=(1-alpha)*(sin2*sin2)
for i in range(0,5*8000):
    y[i]=y[i]+y[i-1]*alpha
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Changing frequency sinus - power enelope, a=0.99")
plt.subplot(4,2,7)
y= wave.open('klarnet.wav')
signal=y.readframes(-1)
signal= np.frombuffer(signal, dtype='int16')
signal=signal/30000
fs=y.getframerate()
time=np.linspace(0, len(signal)/fs, len(signal))
plt.plot(time,signal)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Voice signal")
plt.subplot(4,2,8)
alpha=0.99
y=(1-alpha)*(signal*signal)
for i in range(0,len(signal)):
    y[i]=y[i]+y[i-1]*alpha
y=y[:len(x)]
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([0,0.1,0.2])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Change frequency signal-power enelope , a=0.99")
plt.tight_layout()
plt.show()
```

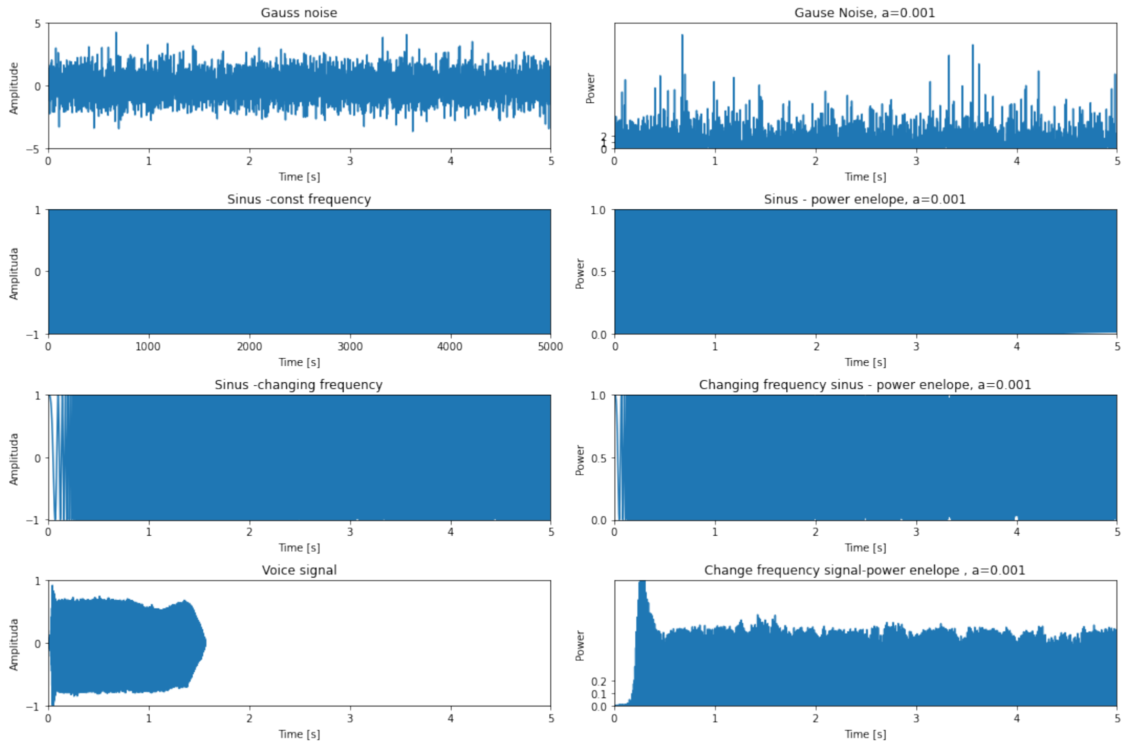



[179]:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import wave
x = np.linspace(0,5000,5001)
noise = np.random.normal(0,1,5001)
plt.figure(figsize=(15,10))
plt.subplot(4,2,1)
plt.plot(x,noise)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-5,0,5])
plt.ylim(-5,5)
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.title("Gauss noise")
plt.subplot(4,2,2)
alpha=0.001
y=(1 - alpha)*(noise * noise)
for i in range(0,5000):
    y[i]=y[i]+y[i - 1]*alpha
plt.plot(x,y)
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
```

```
plt.xlim(0,5000)
plt.yticks([0,1,2])
plt.ylim(0,20)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Gause Noise, a=0.001")
plt.subplot(4,2,3)
x=np.linspace(0,5000 * 2 * np.pi,5 * 8000)
sin1=np.sin(x)
plt.plot(x,sin1)
plt.xticks
(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim(0,5000)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Sinus -const frequency")
plt.subplot(4,2,4)
alpha=0.001
y=(1-alpha)*(sin1*sin1)
for i in range(0,int(5000*2*np.pi)):
    y[i]=y[i]+y[i-1]*alpha
plt.plot(x,y)
plt.axis([-5, 5, 0, 5]);
plt.xticks(np.arange(0,6000,1000),["0","1","2","3","4","5"])
plt.xlim
(0,5000)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Sinus - power enelope, a=0.001")
plt.subplot(4,2,5)
x=np.linspace(0,5,5*8000)
sin2=signal.chirp(x, f0=0, f1=1000, t1=5)
plt.plot(x,sin2)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Sinus -changing frequency")
plt.subplot(4,2,6)
alpha=0.001
y=(1-alpha)*(sin2*sin2)
```

```
for i in range(0,5*8000):
    y[i]=y[i]+y[i-1]*alpha
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([0,0.5,1])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Changing frequency sinus - power enelope, a=0.001")
plt.subplot(4,2,7)
y= wave.open('klarnet.wav')
signal=y.readframes(-1)
signal= np.frombuffer(signal, dtype='int16')
signal=signal/30000
fs=y.getframerate()
time=np.linspace(0, len(signal)/fs, len(signal))
plt.plot(time,signal)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"])
plt.xlim(0,5)
plt.yticks([-1,0,1])
plt.ylim(-1,1)
plt.xlabel("Time [s]")
plt.ylabel("Amplituda")
plt.title("Voice signal")
plt.subplot(4,2,8)
alpha=0.001
y=(1-alpha)*(signal*signal)
for i in range(0,len(signal)):
    y[i]=y[i]+y[i-1]*alpha
y=y[:len(x)]
plt.axis([0, 1000, -1, 1])
plt.plot(x,y)
plt.xticks(np.linspace(0,5,6),["0","1","2","3","4","5"]) #zle
plt.xlim(0,5)
plt.yticks([0,0.1,0.2])
plt.ylim(0,1)
plt.xlabel("Time [s]")
plt.ylabel("Power")
plt.title("Change frequency signal-power enelope , a=0.001")
plt.tight_layout()
plt.show()
```




**Wnioski:**

**Wraz ze wzrostem parametru uśredniającego (alfa), coraz bardziej zmniejsza się maksymalna amplituda obwiedni mocy, którą otrzymujemy po zastosowaniu uśredniania wg podanego wzoru. Otrzymany kształt obwiedni pozwala na określenie stacjonarności sygnałów czyli stałości ich parametrów statycznych, np. wariancji. Im bardziej nieregularny kształt obwiedni, tym mniej stacjonarny jest wykres.**

**Zadanie 4.2**

**Wygenerować następujące sygnały (po N = 1024 próbek, każdy): szum gaussowski - w[n] z parametrami = 0, 2 = 1, kombinację sygnałów sinusoidalnych o częstotliwościach f1 = 500Hz i f2 = 1.2kHz zgodnie ze wzorem s[n] = 0.5sin(2 n f1/fs) + sin(2 n f2/fs) oraz sygnał y[n] = s[n] + 0.1w[n], (we wszystkich przypadkach założyć, że fs = 8kHz). Dla każdego z sygnałów oszacować widmową gęstość mocy (ang. Power Spectral Density - PSD) wykorzystując następujące metody:**

**a) periodogramu**

**b) zmodyfikowanego periodogramu (dla okna Hanna)**

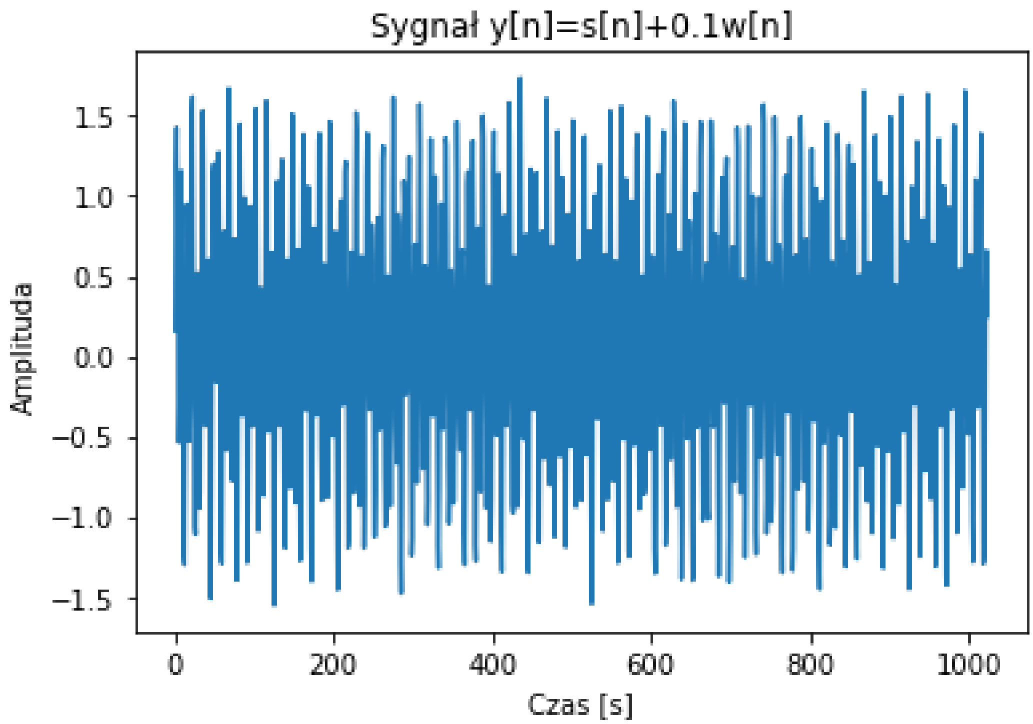
**c) Welcha (dla okna Hanna dł. 256, 128 i 64 próbki z 50% nakładaniem)**

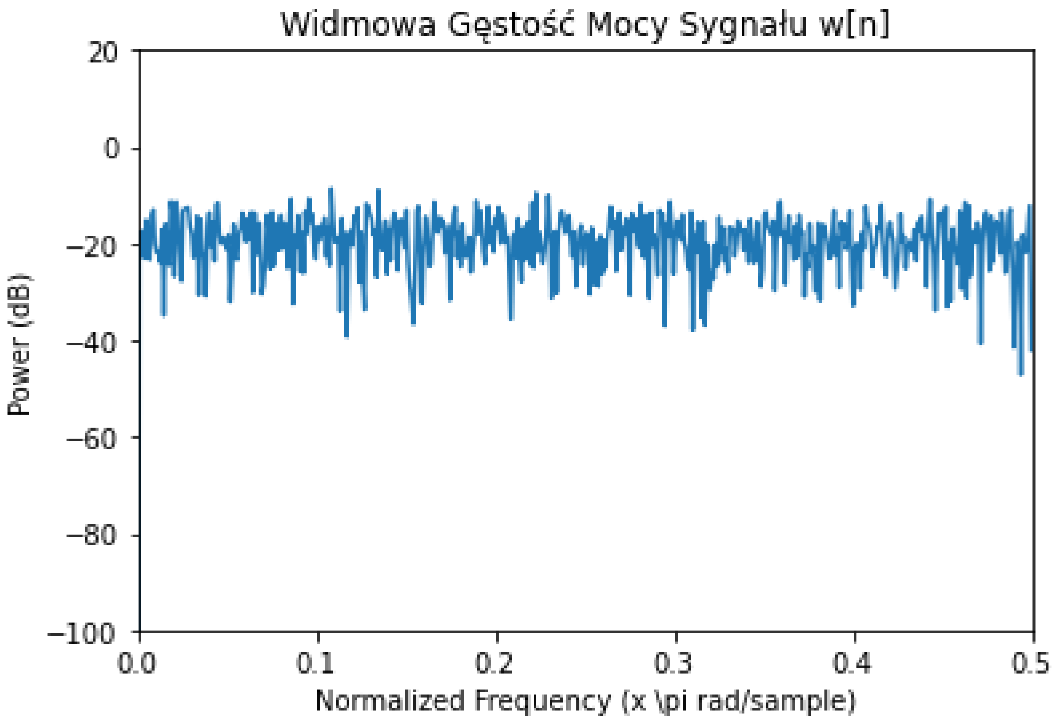
**Sporządzić wykresy widmowej gęstości mocy w skali decybelowej (wyniki dla podpunktu czaprezentować w tym samym oknie). Co możesz powiedzieć o wariancji i obciążeniu poszczególnych oszacowań (estymatorów)? Jakie warunki/założenia muszą spełniać analizowane sygnały? Czy możliwe byłoby oszacowanie PSD dla sygnału o zmiennej częstotliwości lub sygnału mowy z poprzedniego zadania?**

**a)**

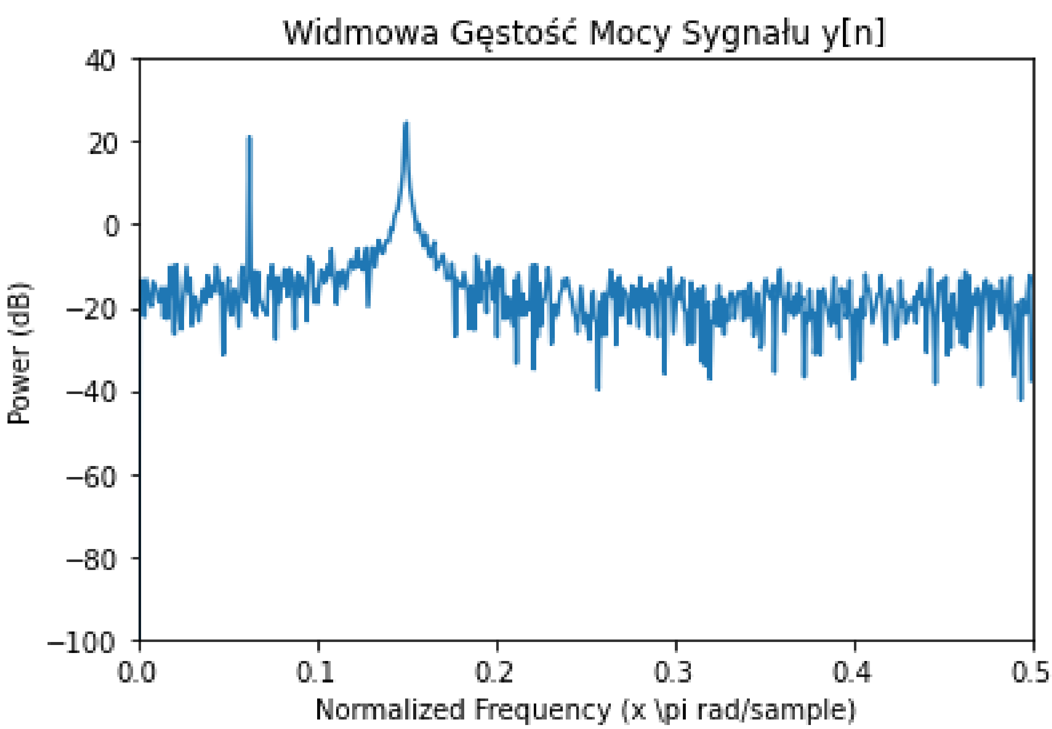
[4]:
```
%matplotlib inline
import numpy as np
import scipy.signal as signal
from scipy.signal import chirp, spectrogram
from scipy.io import wavfile
import matplotlib.pyplot as plt
# częstotliwość próbkowania
fs = 8000
sig_win = 1024
n = np.arange(sig_win)
w=np.random.randn(1024)+1
s = 0.5*np.sin(2*np.pi*n*500/fs) + np.sin(2*np.pi*n*1200/fs)
wone = 0.1*w
y =s+wone
plt.plot(n,y)
plt.title('Sygnał y[n]=s[n]+0.1w[n]')
plt.xlabel('Czas [s]')
plt.ylabel('Amplituda')
plt.show()
f = np.fft.rfftfreq(2048, 1/fs)
f,pxx = signal.periodogram(wone)
plt.plot(f,10*np.log10(pxx))
plt.ylim(-100,20)
plt.xlim(0,0.5)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału w[n]')
plt.show()
f = np.fft.rfftfreq(2048, 1/fs)
f,pxx = signal.periodogram(y)
plt.plot(f,10*np.log10(pxx))
plt.ylim(-100,40)
plt.xlim(0,0.5)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału y[n]')
plt.show()
f = np.fft.rfftfreq(2048, 1/fs)
f,pxx = signal.periodogram(s)
plt.plot(f,10*np.log10(pxx))
```

```
plt.ylim(-100,40)
plt.xlim(0,0.5)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału w[n]')
plt.show()
```

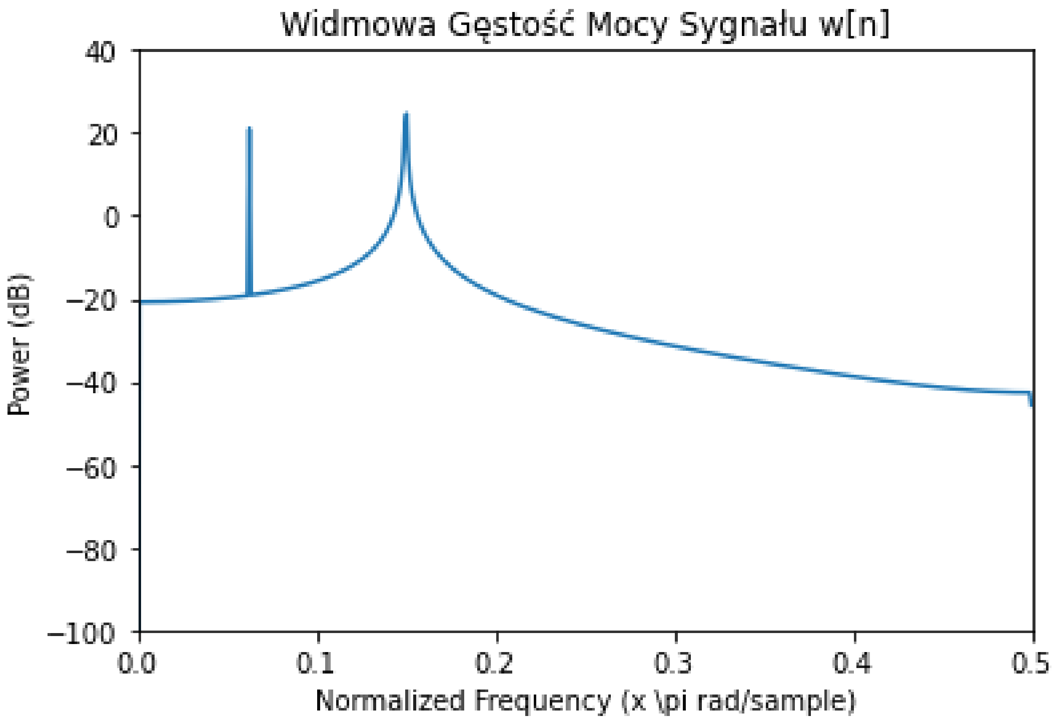




Widmowa Gęstość Mocy Sygnału w[n]
20
0
−20
−40
−60
−80
−100
Power (dB)
0.0
0.1
0.2
0.3
0.4
0.5
Normalized Frequency (x \pi rad/sample)

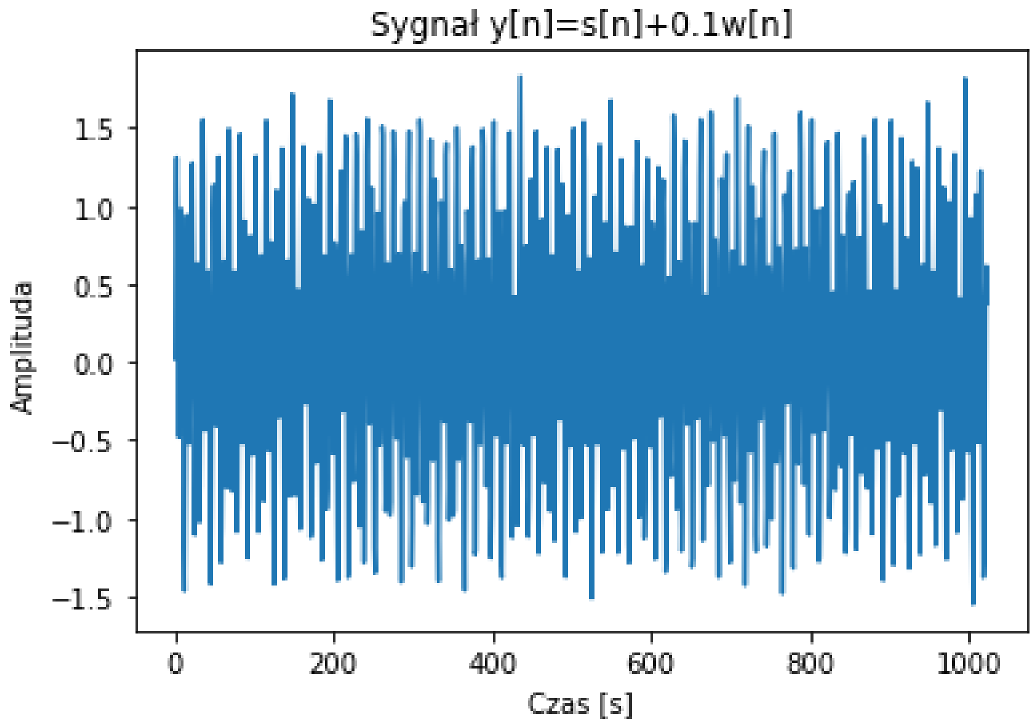
Widmowa Gęstość Mocy Sygnału y[n]
40
20
0
−20
−40
−60
−80
−100
Power (dB)
0.0
0.1
0.2
0.3
0.4
0.5
Normalized Frequency (x \pi rad/sample)

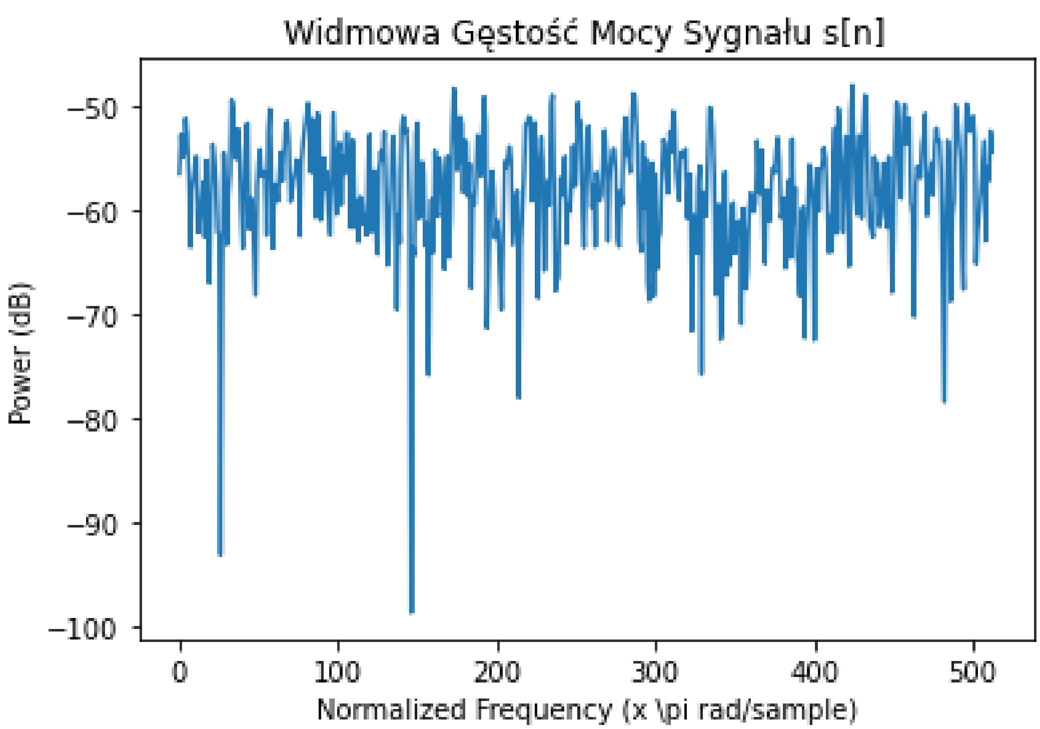


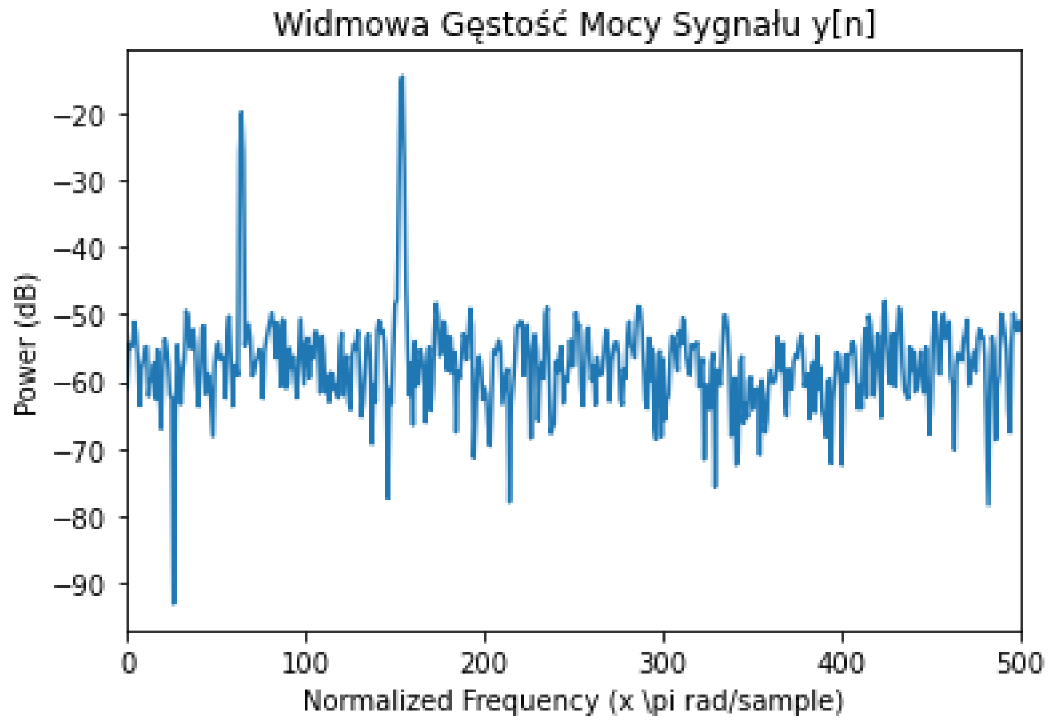

**b)**

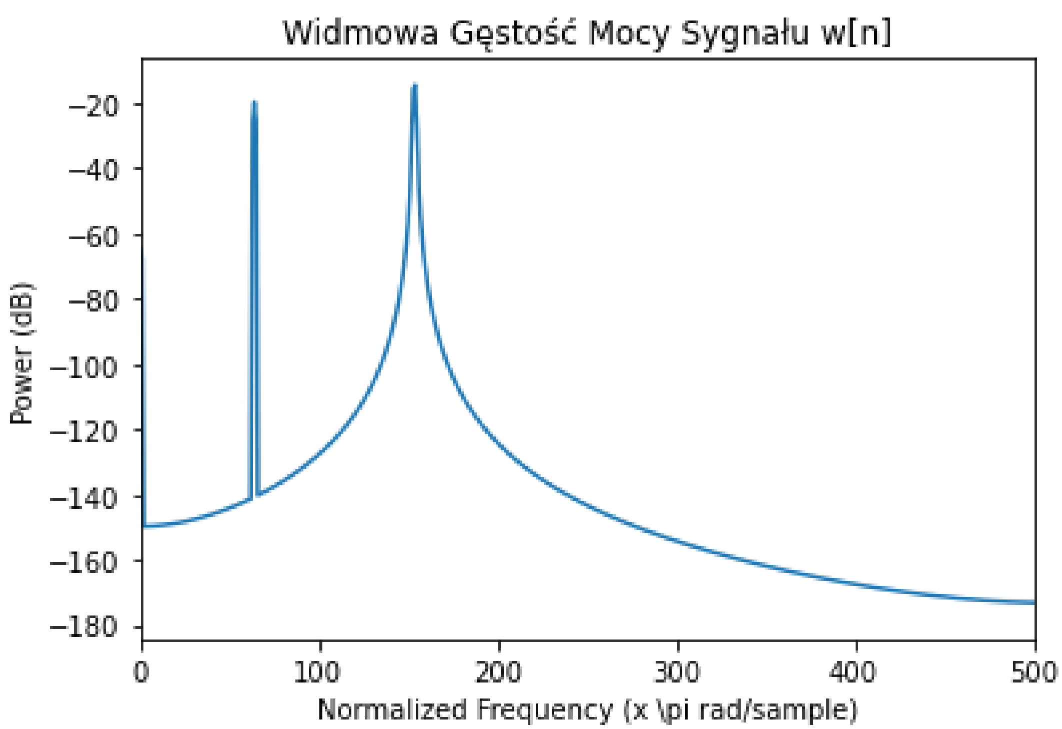
[6]:
```
%matplotlib inline
import numpy as np
import scipy.signal as sig
from scipy.signal import chirp, spectrogram
from scipy.fft import fftshift
from scipy.io import wavfile
import matplotlib.pyplot as plt
# częstotliwość próbkowania
fs = 8000
sig_win = 1024
n = np.arange(sig_win)

w=np.random.randn(1024)+1
s = 0.5*np.sin(2*np.pi*n*500/fs) + np.sin(2*np.pi*n*1200/fs)
wone = 0.1*w
y =s+wone
plt.plot(n,y)
plt.title('Sygnał y[n]=s[n]+0.1w[n]')
plt.xlabel('Czas [s]')
plt.ylabel('Amplituda')

plt.show()
```

```
f = np.fft.rfftfreq(1024, 1/fs)
f,pxx= sig.periodogram(wone, fs, window='hann', nfft=1024, detrend='constant',␣
↪return_onesided=True, scaling='density')
plt.plot(10*np.log10(pxx))
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału s[n]')


plt.show()

f = np.fft.rfftfreq(1024, 1/fs)
f, pxx = sig.periodogram(y, fs, window='hann', nfft=1024, detrend='constant',␣
↪return_onesided=True, scaling='density')
plt.plot(10*np.log10(pxx))
plt.xlim(0,500)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału y[n]')

plt.show()
f = np.fft.rfftfreq(1024, 1/fs)
f, pxx = sig.periodogram(s, fs, window='hann', nfft=1024, detrend='constant',␣
↪return_onesided=True, scaling='density')
plt.plot(10*np.log10(pxx))
plt.xlim(0,500)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału w[n]')

plt.show()
```


Sygnał y[n]=s[n]+0.1w[n]
Amplituda
Czas [s]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000

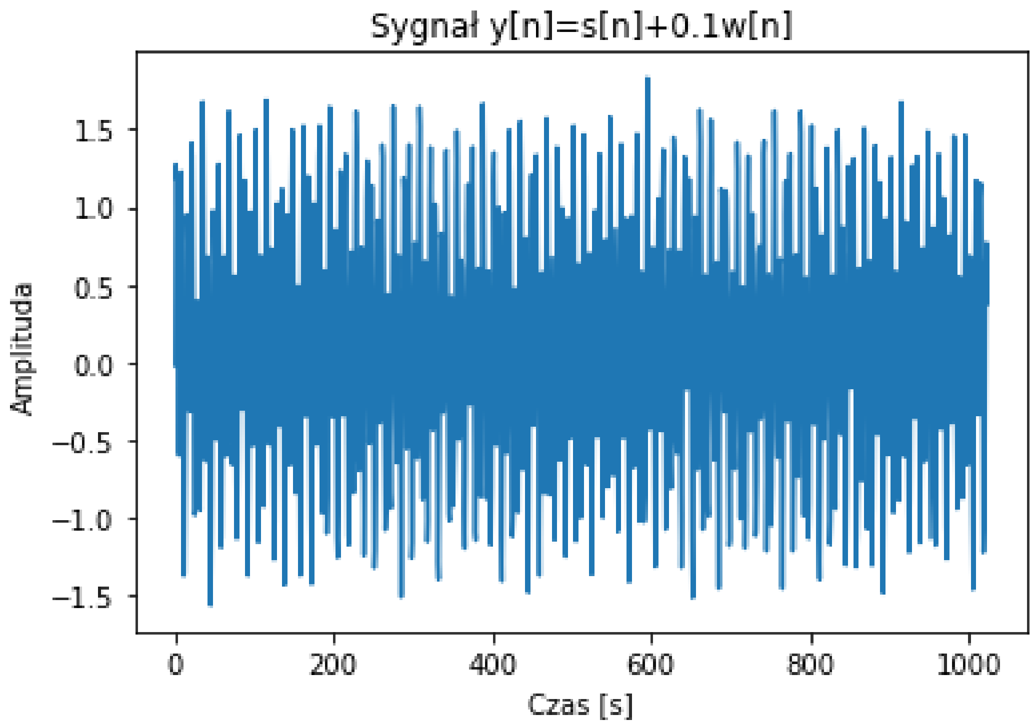

Widmowa Gęstość Mocy Sygnału s[n]
Power (dB)
Normalized Frequency (x \pi rad/sample)
−50
−60
−70
−80
−90
−100
0
100
200
300
400
500

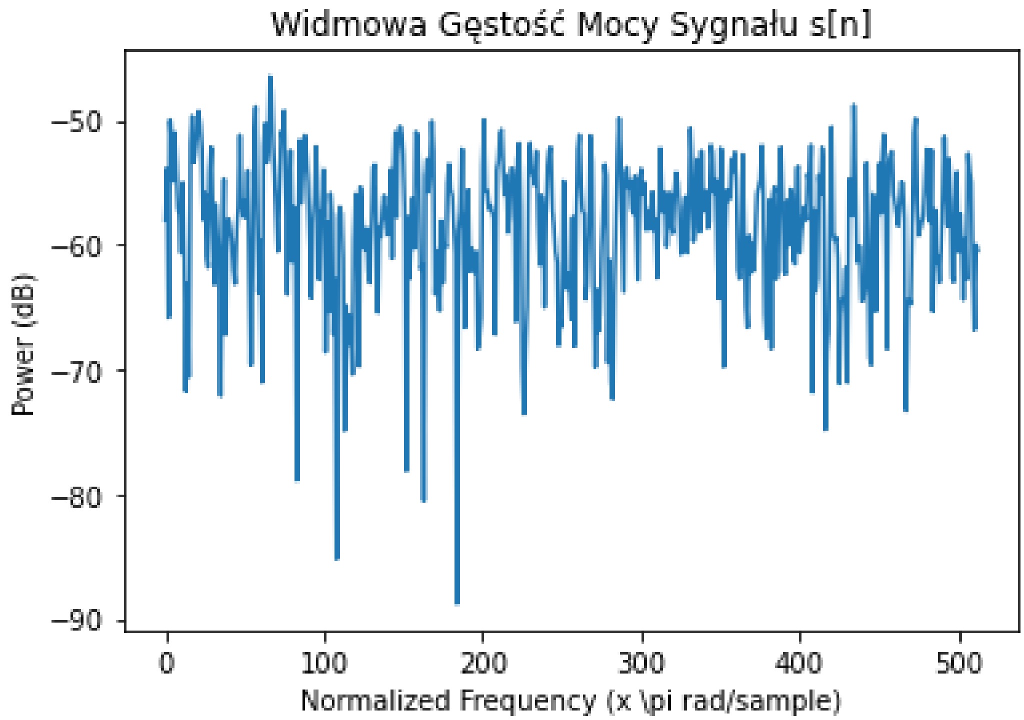

Widmowa Gęstość Mocy Sygnału y[n]
Power (dB)
Normalized Frequency (x \pi rad/sample)
−20
−30
−40
−50
−60
−70
−80
−90
0
100
200
300
400
500

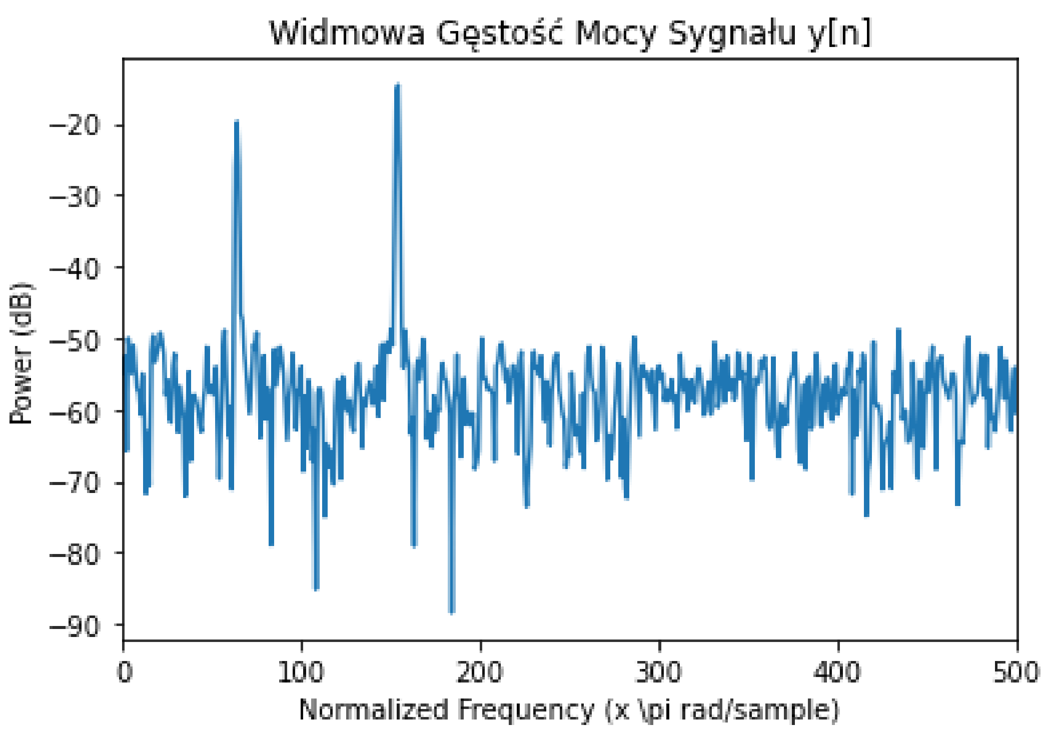

Widmowa Gęstość Mocy Sygnału w[n]
Power (dB)
Normalized Frequency (x \pi rad/sample)
−20
−40
−60
−80
−100
−120
−140
−160
−180
0
100
200
300
400
500

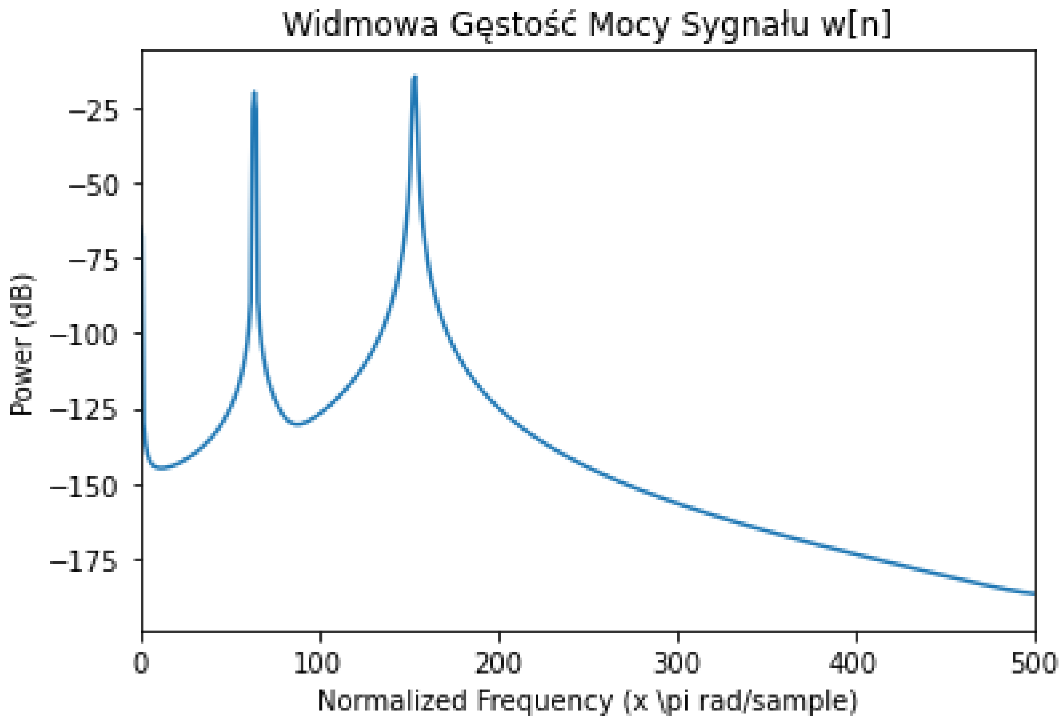
[36]:
```python
%matplotlib inline
import numpy as np
import scipy.signal as sig
from scipy.signal import hann
from scipy.signal import chirp, spectrogram
from scipy.fft import fftshift
from scipy.io import wavfile
import matplotlib.pyplot as plt
# częstotliwość próbkowania
fs = 8000
sig_win = 1024
n = np.arange(sig_win)

w=np.random.randn(1024)+1
s = 0.5*np.sin(2*np.pi*n*500/fs) + np.sin(2*np.pi*n*1200/fs)
wone = 0.1*w
y =s+wone
plt.plot(n,y)
plt.title('Sygnał y[n]=s[n]+0.1w[n]')
plt.xlabel('Czas [s]')
plt.ylabel('Amplituda')

plt.show()
f = np.fft.rfftfreq(1024, 1/fs)
f,pxx = sig.periodogram(wone,fs,hann(len(wone)))
plt.plot(10*np.log10(np.abs(pxx)))
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału s[n]')


plt.show()

f = np.fft.rfftfreq(1024, 1/fs)
f,pxx = sig.periodogram(y,fs,hann(len(wone)))
plt.plot(10*np.log10(pxx))
plt.xlim(0,500)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału y[n]')

plt.show()
f = np.fft.rfftfreq(124, 1/fs)
f,pxx = sig.periodogram(s,fs,hann(len(wone)))
plt.plot(10*np.log10(np.abs(pxx)))
plt.xlim(0,500)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
```

```
plt.ylabel('Power (dB)')
plt.title('Widmowa Gęstość Mocy Sygnału w[n]')

plt.show()
```




Widmowa Gęstość Mocy Sygnału s[n]
Power (dB)
−50
−60
−70
−80
−90
0
100
200
300
400
500
Normalized Frequency (x \pi rad/sample)


Widmowa Gęstość Mocy Sygnału y[n]
Power (dB)
−20
−30
−40
−50
−60
−70
−80
−90
0
100
200
300
400
500
Normalized Frequency (x \pi rad/sample)

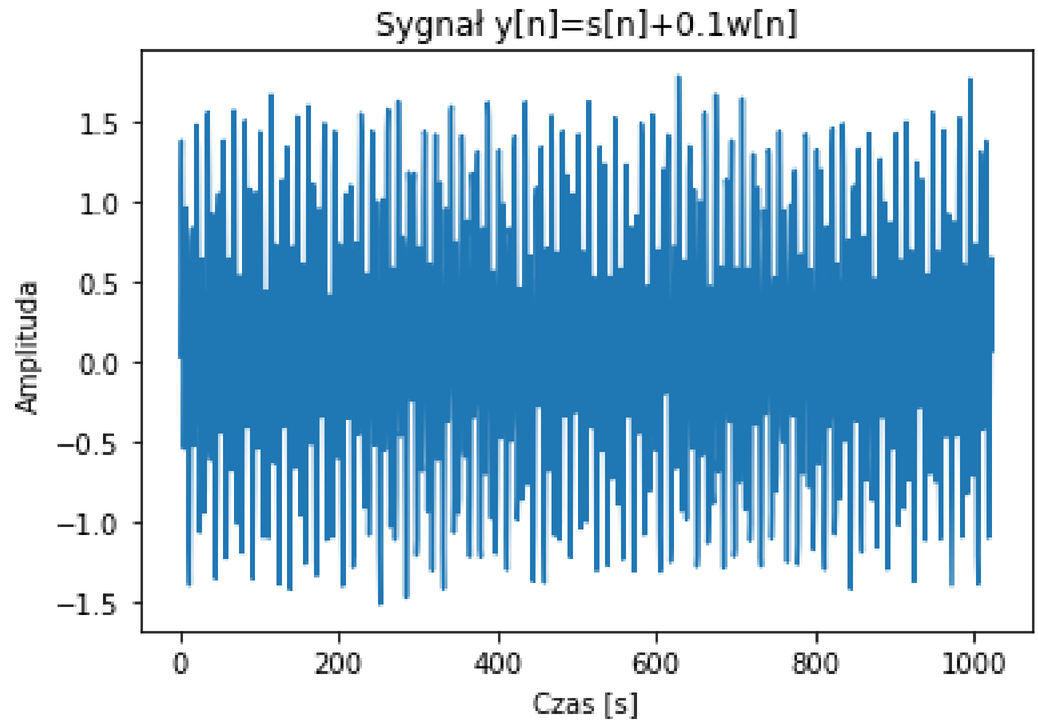


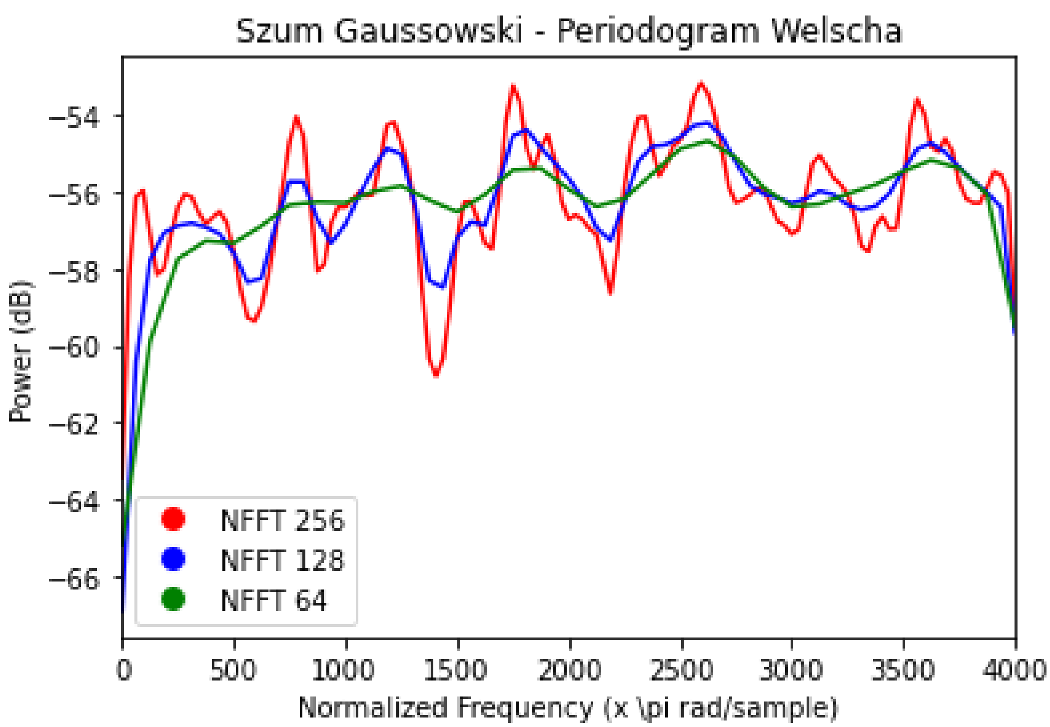

**c)**

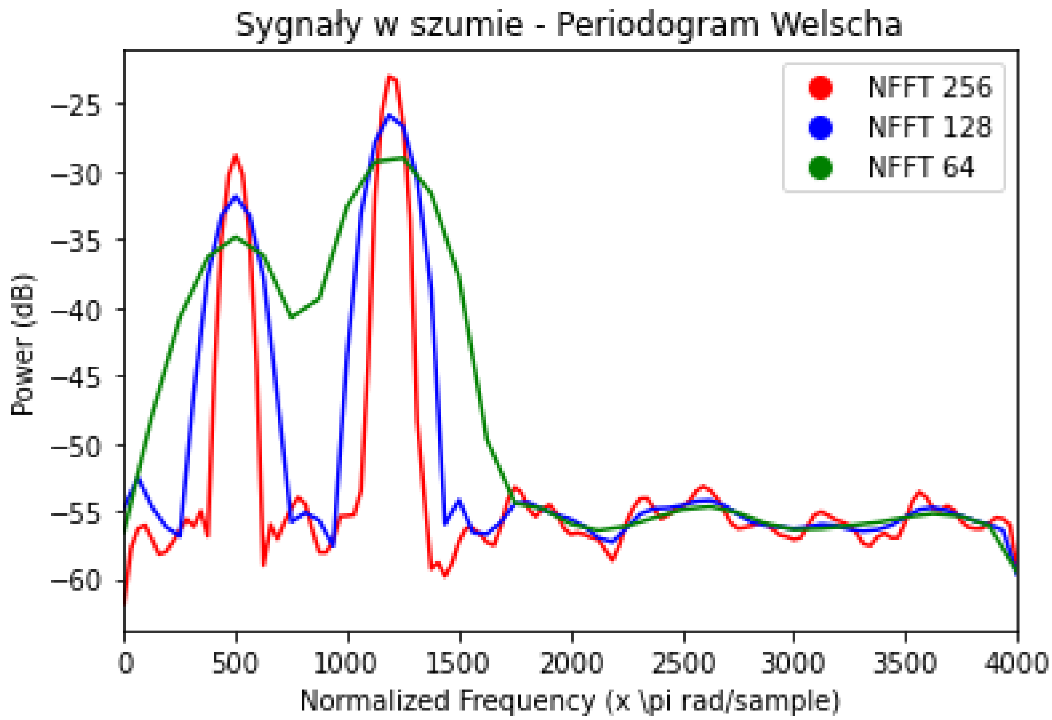
[4]:
```
%matplotlib inline
import numpy as np
from matplotlib.lines import Line2D
from scipy import signal
import scipy.signal as sig
from scipy.signal import welch
from scipy.signal import hann
from scipy.signal import chirp, spectrogram
from scipy.io import wavfile
import matplotlib.pyplot as plt
# częstotliwość próbkowania
fs = 8000
sig_win = 1024
n = np.arange(sig_win)
w=np.random.randn(1024)+1
s = 0.5*np.sin(2*np.pi*n*500/fs) + np.sin(2*np.pi*n*1200/fs)
wone = 0.1*w
y =s+wone
plt.plot(n,y)
plt.title('Sygnał y[n]=s[n]+0.1w[n]')
plt.xlabel('Czas [s]')
plt.ylabel('Amplituda')
```

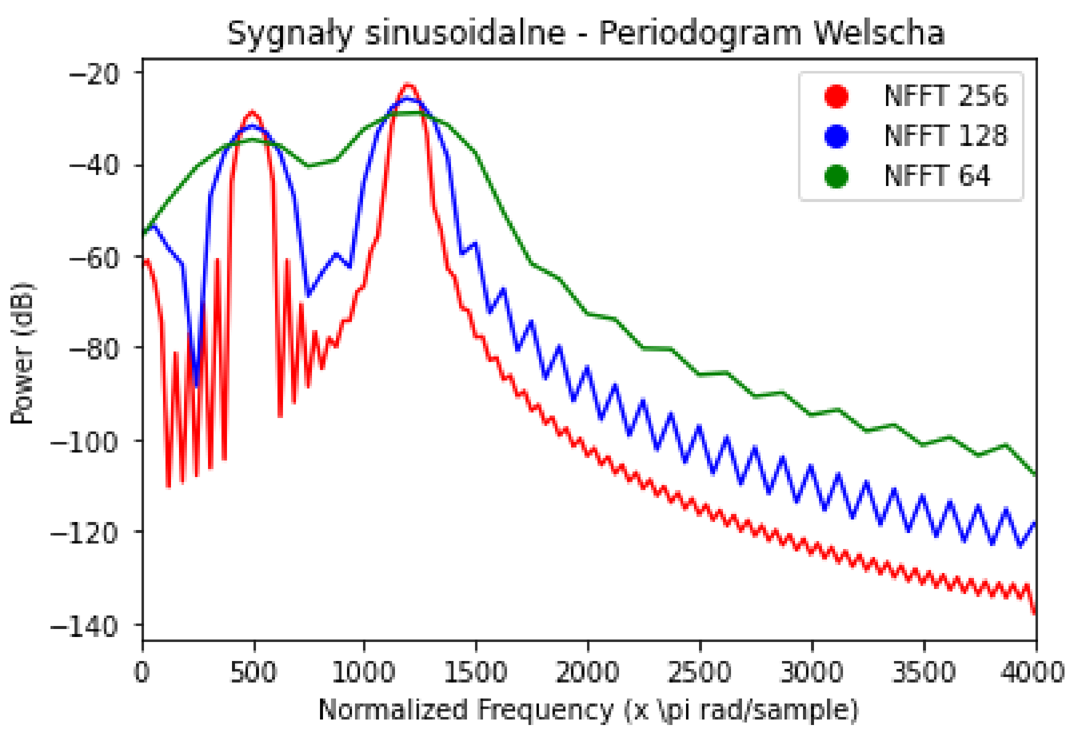
```
plt.show()
f = np.fft.rfftfreq(1024, 1/fs)
f,pxx = signal.welch(wone, fs, window='hann', nperseg=128,  nfft=256,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx)),'r')
f,pxx1 = signal.welch(wone, fs, window='hann',nperseg=64,  nfft=128,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'b')
f,pxx1 = signal.welch(wone, fs, window='hann', nperseg=32,  nfft=64,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'g')
plt.xlim(0,4000)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Szum Gaussowski - Periodogram Welscha')
legend_elements = [Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪256',markerfacecolor='r', markersize=10),
                  Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪128',markerfacecolor='b', markersize=10),
                   Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪64',markerfacecolor='g', markersize=10)
                  ]

# Make legend
plt.legend(handles=legend_elements, loc='best')
plt.show()
f = np.fft.rfftfreq(1024, 1/fs)
f,pxx = signal.welch(y, fs, window='hann',  nperseg=128,  nfft=256,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx)),'r')
f,pxx1 = signal.welch(y, fs, window='hann', nperseg=64,  nfft=128,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'b')
f,pxx1 = signal.welch(y, fs, window='hann', nperseg=32,  nfft=64,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'g')
plt.xlim(0,4000)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Sygnały w szumie - Periodogram Welscha')
```

```
legend_elements = [Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪256',markerfacecolor='r', markersize=10),
                  Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪128',markerfacecolor='b', markersize=10),
                   Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪64',markerfacecolor='g', markersize=10)
                  ]

# Make legend
plt.legend(handles=legend_elements, loc='best')
plt.show()
plt.show()
f,pxx = signal.welch(s, fs, window='hann',  nperseg=128, ␣
↪nfft=256,detrend='constant', return_onesided=True, scaling='density', axis=-␣
↪1, average='mean')
plt.plot(f,10*np.log10(np.abs(pxx)),'r')
f,pxx1 = signal.welch(s, fs, window='hann', nperseg=64,  nfft=128,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'b')
f,pxx1 = signal.welch(s, fs, window='hann', nperseg=32,  nfft=64,␣
↪detrend='constant', return_onesided=True, scaling='density', axis=- 1,␣
↪average='mean')
plt.plot(f,10*np.log10(np.abs(pxx1)),'g')
plt.xlim(0,4000)
plt.xlabel('Normalized Frequency (x \pi rad/sample)')
plt.ylabel('Power (dB)')
plt.title('Sygnały sinusoidalne - Periodogram Welscha')
legend_elements = [Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪256',markerfacecolor='r', markersize=10),
                  Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪128',markerfacecolor='b', markersize=10),
                   Line2D([0], [0], marker='o', color='w', label='NFFT␣
↪64',markerfacecolor='g', markersize=10)
                  ]

# Make legend
plt.legend(handles=legend_elements, loc='best')
plt.show()
plt.show()
```


Sygnał y[n]=s[n]+0.1w[n]
Amplituda
Czas [s]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
200
400
600
800
1000


Szum Gaussowski - Periodogram Welscha
Power (dB)
Normalized Frequency (x \pi rad/sample)
NFFT 256
NFFT 128
NFFT 64
−54
−56
−58
−60
−62
−64
−66
0
500
1000
1500
2000
2500
3000
3500
4000


Sygnały w szumie - Periodogram Welscha
NFFT 256
NFFT 128
NFFT 64
Power (dB)
Normalized Frequency (x \pi rad/sample)


Sygnały sinusoidalne - Periodogram Welscha
NFFT 256
NFFT 128
NFFT 64
Power (dB)
Normalized Frequency (x \pi rad/sample)

**Wnioski:**

- **Na widmowych wykresach gęstości mocy otrzymanych metodą periodogramu można zaobserwować większą wariancję niż dla wykresów widmowych gęstości mocy otrzymanych metodą periodogramu z użyciem okien Hanna.**

- **Porównując widmowe gęstości mocy metodą Welcha można dostrzec, że wraz z długością okien Hanna, zmniejsza sie wariancja, a wraz z nią maleje obciążenie, dzięki czemu można otrzymać oszacowanie z wiekszą dokładnością.**

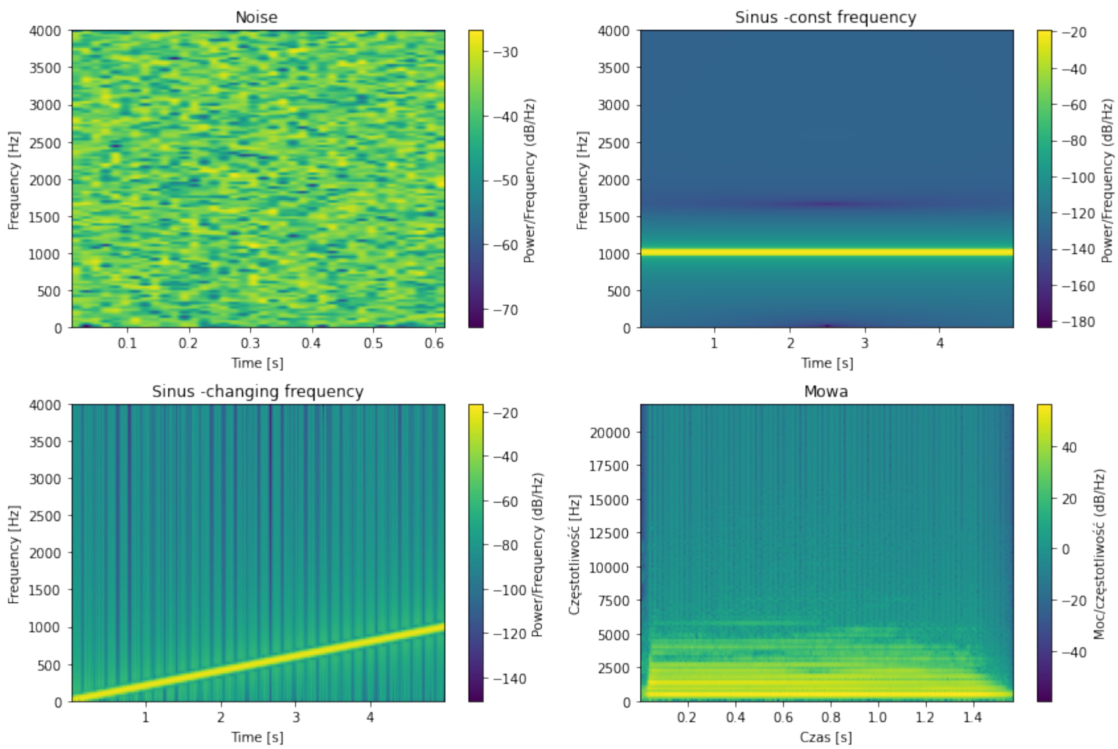
- **Zmienność oszacowania widmowej gęstości mocy dla metody Welcha i okna Hanna o długości 64 jest najmniejsza, zaś największa dla długości 256.**

**Zadanie 4.3**

**Wykorzystując funkcję spectrogram sporządzić spektrogramy dla sygnałów z zadania 4.1. Założyć, że oknem analizy jest okno Hamminga długości 256 próbek. Zwróć uwagę na to, by osie wykresu były opisane przy użyciu jednostek fizycznych (a nie znormalizowanych).Co możesz powiedzieć o rozkładzie energii w czasie i częstotliwości analizowanych sygnałów? Jaka jest relacja pomiędzy modułem widma krótkookresowego a widmową gęstością mocy w skali decybelowej?**

[7]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile
fs = 8000
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
x=np.linspace(0,5000,5001)
noise = np.random.normal(0,1,5001)
plt.specgram(noise, 256, fs, window=np.hamming(256))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Noise")

plt.subplot(2,2,2)
x2=np.linspace(0,5000*2* np.pi,5*8000)
sin1=np.sin(x2)
plt.specgram(sin1, 256, fs, window= np.hamming(256))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Sinus -const frequency")
plt.subplot(2,2,3)
x3=np.linspace(0,5,5*8000)
sin2 = signal.chirp(x3, f0=0, f1=1000, t1=5)
plt.specgram(sin2, 256, fs, window=np.hamming(256))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
```

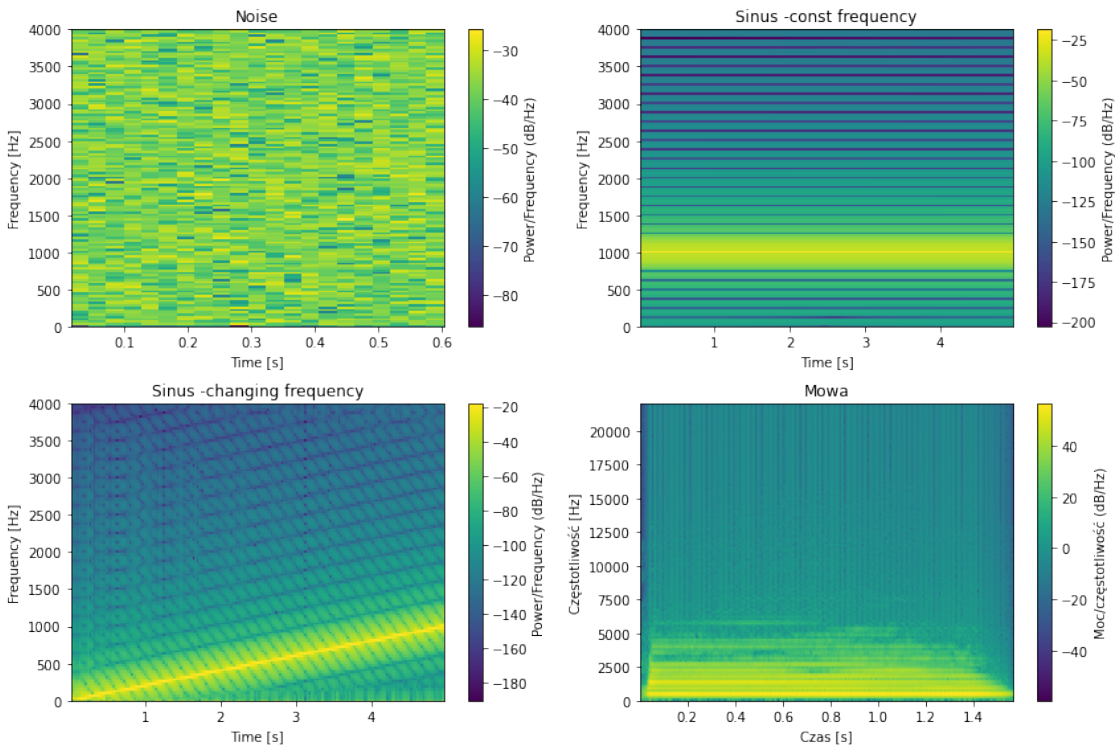
```
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Sinus -changing frequency")
plt.subplot(2,2,4)
sample_rate, samples = wavfile.read('klarnet.wav')
plt.specgram(samples, 256, sample_rate, window=np.hamming(256))
plt.ylabel('Częstotliwość [Hz]')
plt.xlabel('Czas [s]')
plt.colorbar(label= 'Moc/częstotliwość (dB/Hz)')
plt.title("Mowa")
plt.tight_layout()
plt.show()
```




### Wykresy 2D

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile
fs = 8000
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
x=np.linspace(0,5000,5001)
noise = np.random.normal(0,1,5001)
```

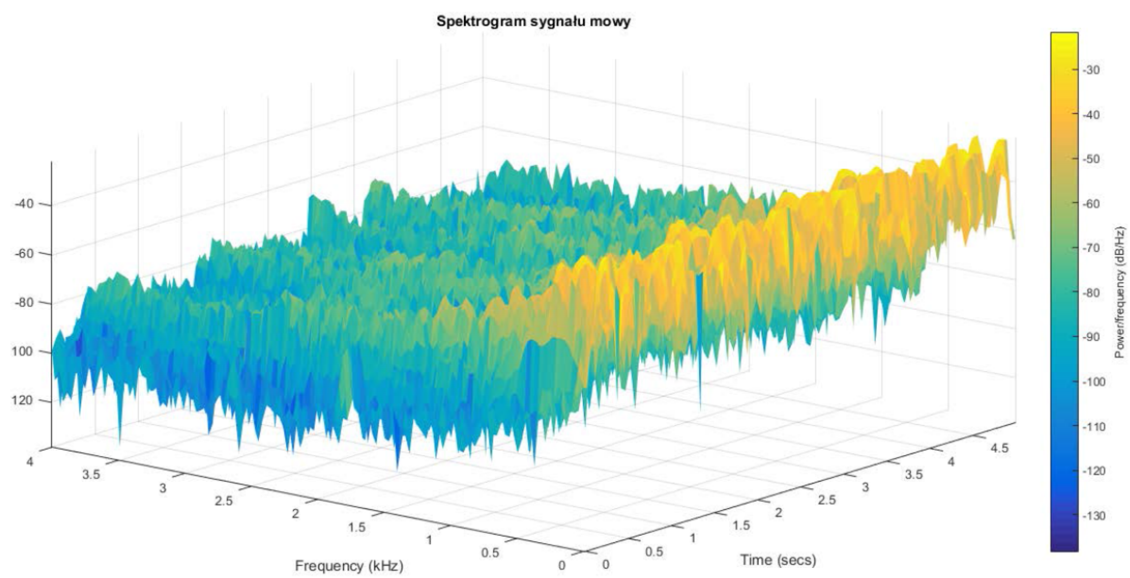
```
f, t, Sxx = signal.spectrogram(noise, fs)
plt.pcolormesh(t, f, 10*np.log10(Sxx))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Noise")

plt.subplot(2,2,2)
x2=np.linspace(0,5000*2* np.pi,5*8000)
sin1=np.sin(x2)
f, t, Sxx = signal.spectrogram(sin1, fs)
plt.pcolormesh(t, f, 10*np.log10(Sxx))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Sinus -const frequency")
plt.subplot(2,2,3)
x3=np.linspace(0,5,5*8000)
sin2 = signal.chirp(x3, f0=0, f1=1000, t1=5)
f, t, Sxx = signal.spectrogram(sin2, fs)
plt.pcolormesh(t, f, 10*np.log10(Sxx))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [s]')
plt.colorbar(label= 'Power/Frequency (dB/Hz)')
plt.title("Sinus -changing frequency")
plt.subplot(2,2,4)
sample_rate, samples = wavfile.read('klarnet.wav')
plt.specgram(samples, 256, sample_rate, window=np.hamming(256))
plt.ylabel('Częstotliwość [Hz]')
plt.xlabel('Czas [s]')
plt.colorbar(label= 'Moc/częstotliwość (dB/Hz)')
plt.title("Mowa")
plt.tight_layout()
plt.show()
```




### Wykresy 3D

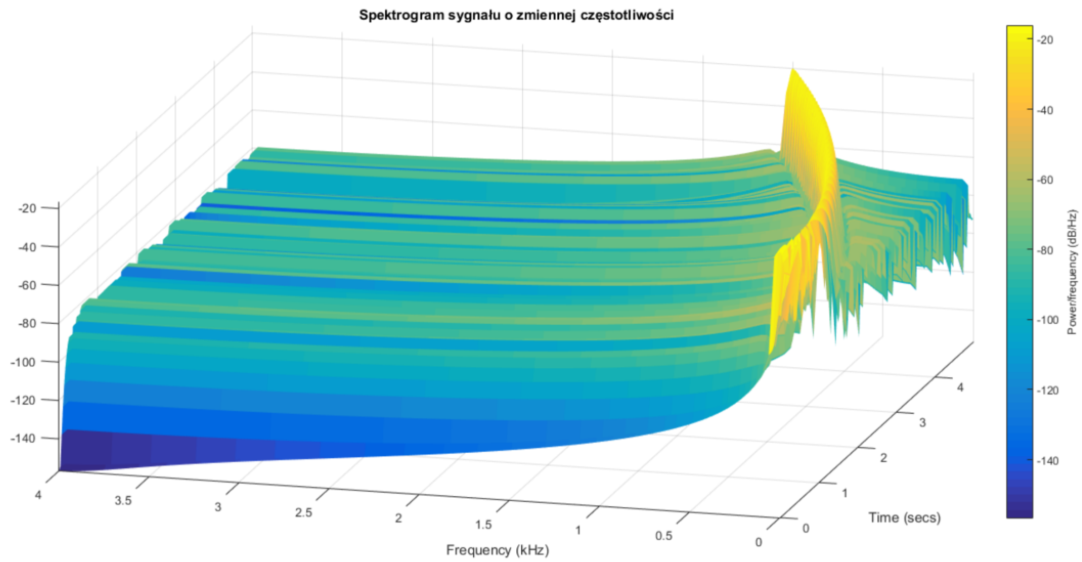
[16]:
```python
from IPython.display import Image
Image(filename='Spectogram sygnału mowy.png')
```

[16]:



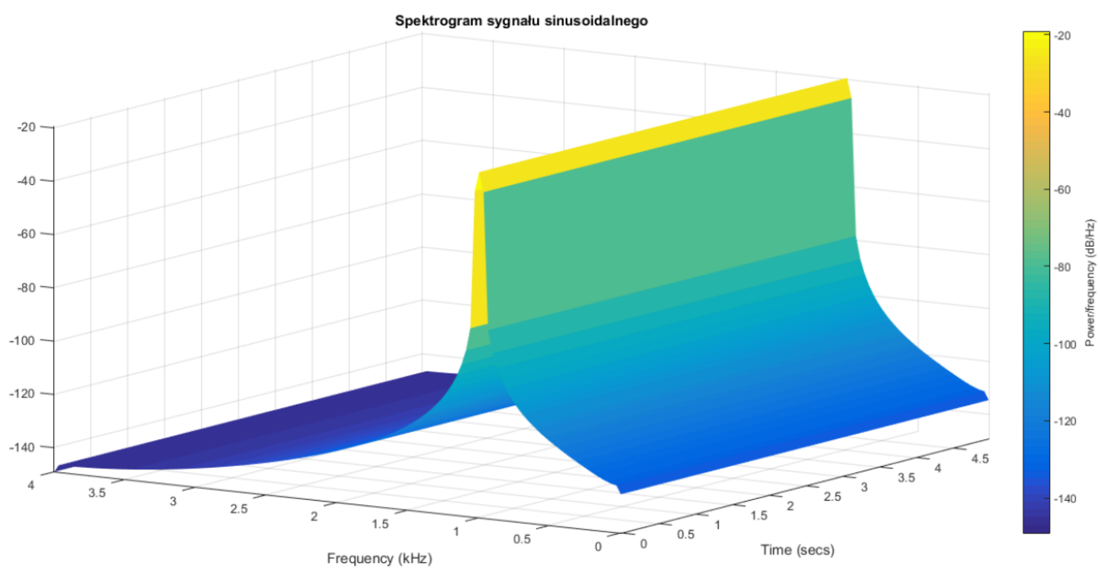
[17]:
```python
from IPython.display import Image
Image(filename='Spectogram sygnału o zmiennej częstotliwosci.png')
```

[17]:



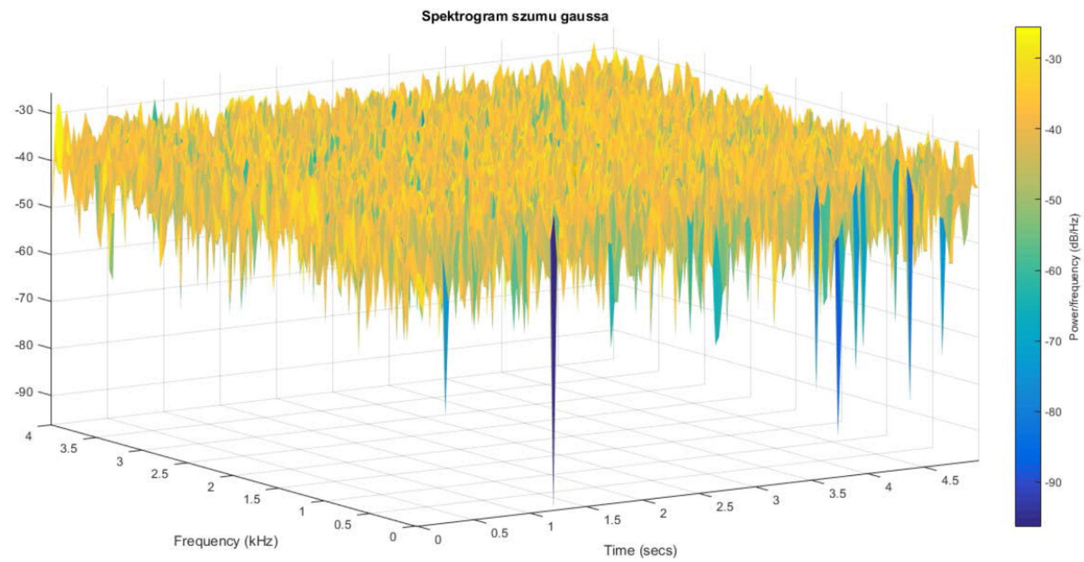

[18]:
```python
from IPython.display import Image
Image(filename='Spectogram sygnału sinusoidalnego.png')
```

[18]:



[19]:
```python
from IPython.display import Image
Image(filename='Spectogram szumu gaussa.png')
```

[19]:



**Wnioski:**

**Na podstawie wykresu szumu można dostrzeć równomiernie rozłożoną energię oraz wysoką wartość mocy.**

**Otrzymany sygnał sinusoidalny zmienna wartośd energii oscylujaca między wysoką a niską.**

**Obserwując sygnał zmienny można zauważyć,że w całej rozpiętości czasu i częstotliwości wartości energii jest niska.**

**Ostatni spektrogram pokazuje zmiany widma dźwięku klarnetu. W początkowej fazie dźwięku, podczas wprawiania powietrza wewnątrz instrumentu w drgania - widmo jest w tej fazie niestabilne. Następnie można dostrzec fazę podtrzymania. Widać harmoniczną strukturę dźwięku - częstotliwość podstawową 440 Hz i jej całkowite wielokrotności - harmoniczne. Nastepnie dźwięk zaczyna wygasać Wszystkie te informacje możemy odczytać z histogramu dzięki analizie STFT.**

**Zadanie 4.4**

**Napisać własną wersję funkcji spectrogram, wykorzystującą, jako narzędzie analizy widmowej metodę uśrednianych periodogramów (ang. smoothed periodograms) i rysującą wykres czasowo częstotliwościowy w postaci siatki 3D (funkcja mesh). Przyjąć, że oś X jest osią czasu, oś Y - osią częstotliwości oraz oś Z - osią widmowej gęstości mocy (w skali decybelowej). Podpowiedź: w celu implementacji metody uśrednianych periodogramów wykorzystać wzór z zadania 4.1 zastępując moc chwilową w czasie x[n] 2 wartościami krótkookresowego widma mocy |X(k,l)| 2, gdzie X(k,l) - k-ty prążek widma zespolonego DFT oraz l - indeks ramki sygnału. Sporządzić wykresy analogicznie jak w zadaniu 4.3, porównać wyniki.**

[ ]:
```
Funkcja wygenerowana przy użyciu Matlaba.
```

[ ]:
```
function [stft, f, t] = spektrogram( x, N, fs ) % N - liczba punktów FFT
    if size(x, 2) > 1
        x = x'
    end
    x = x/max(abs(x))
    win= hann(256, 'periodic')
    row = 0
    column = 1
    stft = zeros(ceil((1+N)/2), 1+fix((length(x)-256)/16))
    % STFT
    while row + 256 <= length(x)
        xw = x(row+1:row+256).*win
        X = fft(xw, N)
        stft(:, column) = X(1:ceil((1+N)/2))
        row = row + 16
        column = column + 1
```

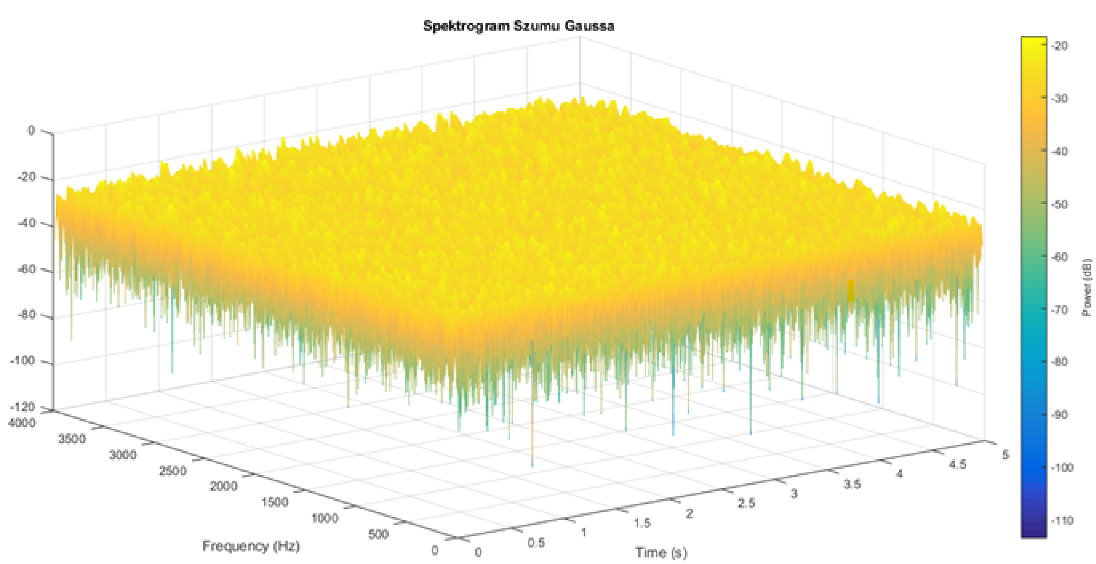
```
    end
    S = sum(hamming(256, 'periodic'))/256
    f = (0:ceil((1+N)/2)-1)*fs/N
    t = (256/2:16:256/2+(1+fix((length(x)-256)/16)-1)*16)/fs
    stft = abs(stft)/256/S
    if rem(N, 2)
        stft(2:end, :) = stft(2:end, :).*2
    else
        stft(2:end-1, :) = stft(2:end-1, :).*2
    end
        stft = 20*log10(stft + 1e-6)
end
```

[ ]:
```
Szum Gaussa
```

[ ]:
```
time=5
fs=8000
y=randn(1, time*fs)
[sp, f, t] =spektrogram(y, 2048, fs)
mesh(t, f, sp)
xlabel('Time (s)')
ylabel('Frequency (Hz)')
col = colorbar
ylabel(col, 'Power (dB)')
title('Spektrogram Szumu Gaussa')
```

[21]:
```
from IPython.display import Image
Image(filename='zad1.png')
```

[21]:

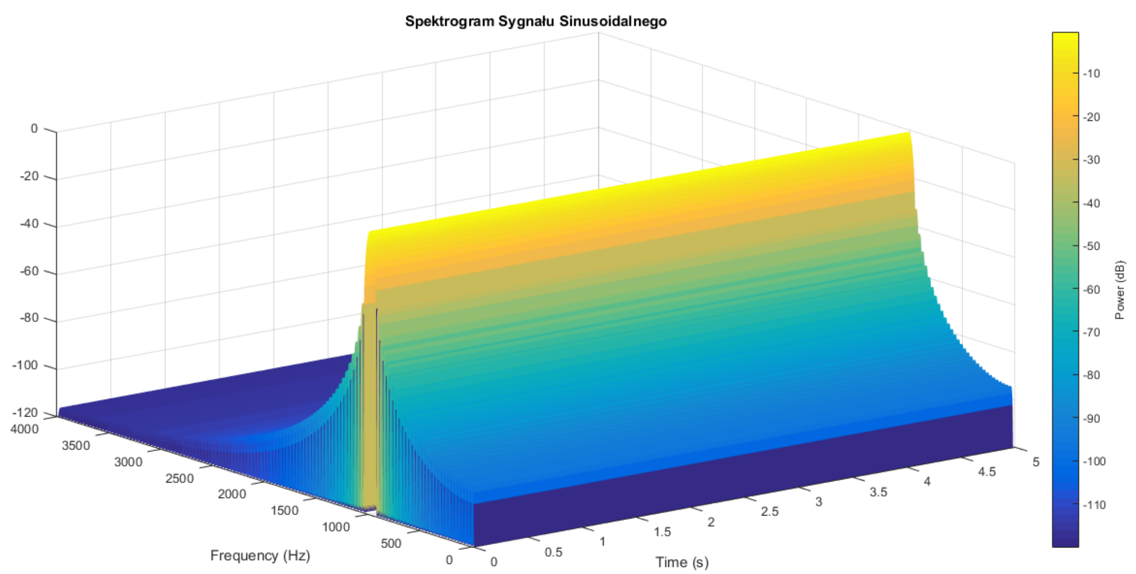


[ ]: Sinusoida o stałej częstotliwości:

[ ]:
```
time=5
fs=8000
t = [0:1/fs:time-1/fs]
f=1000
y=sin(2*pi*f*t)
[sp, f, t] = spektrogram(y, 2048, fs)
mesh(t, f, sp)
xlabel('Time (s)')
ylabel('Frequency (Hz)')
col = colorbar
ylabel(col, 'Power (dB)')
title('Spektrogram Sygnału Sinusoidalnego')
```

[22]:
```
from IPython.display import Image
Image(filename='zad2.png')
```

[22]:



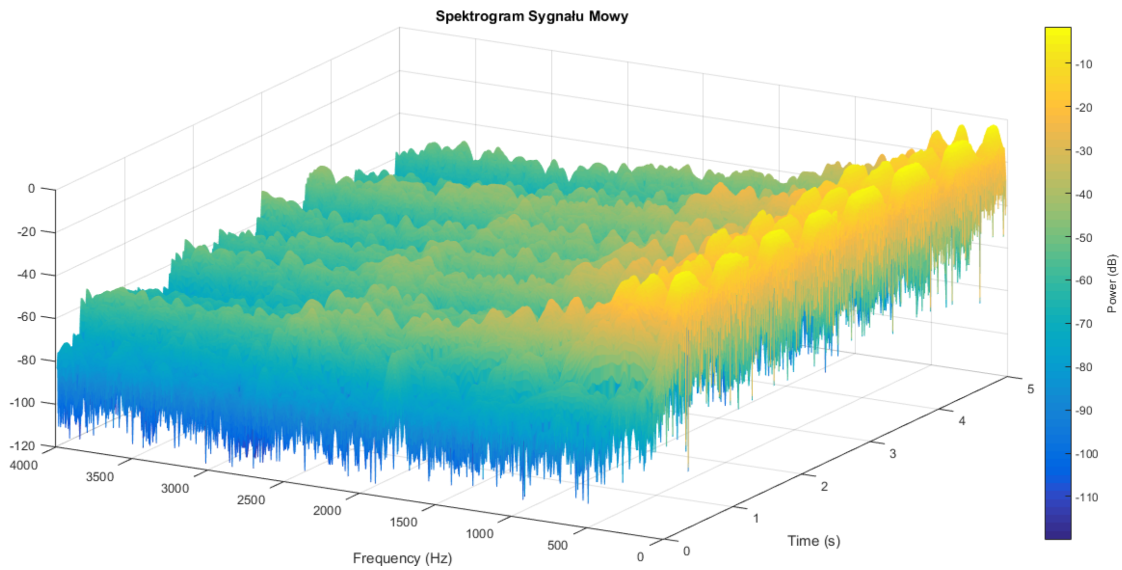

[ ]: Sygnał mowy

[ ]:
```
time=5
fs=8000
t=[0:1/fs:time-1/fs]
speech=audioread('klarnet.wav')
speech=speech(1:40000)
[s, f, t] = spektrogram(speech, 2048, fs)
mesh(t, f, s)
```

```
xlabel('Time (s)')
ylabel('Frequency (Hz)')
col = colorbar
ylabel(col, 'Power (dB)')
title('Spektrogram Sygnału Mowy')
```

[23]:
```
from IPython.display import Image
Image(filename='zad3.png')
```

[23]:

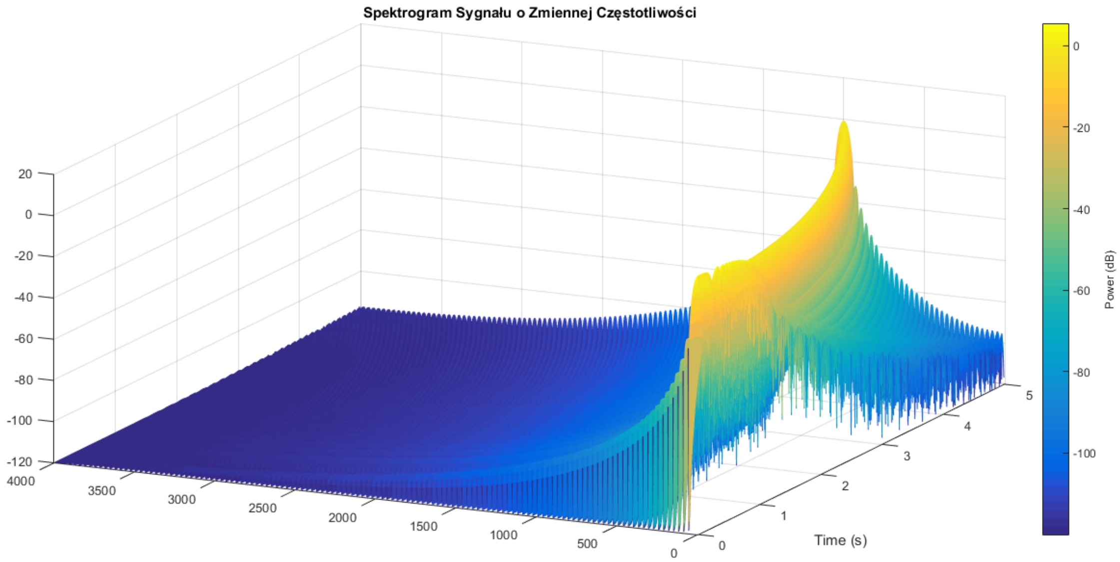



[ ]:
```
Sygnał o zmiennej częstotliwości
```

[ ]:
```
time=5
fs=8000
t=[0:1/fs:time-1/fs]
y=chirp(t, 0, 5, 1000, 'quadratic')
[s, f, t] = spektrogram(y, 2048, fs)
mesh(t, f, s)
xlabel('Time (s)')
ylabel('Frequency (Hz)')
col = colorbar
ylabel(col, 'Power (dB)')
title('Spektrogram Sygnału o Zmiennej Częstotliwości')
```

[24]:
```
from IPython.display import Image
Image(filename='zad4.png')
```

[24]:




**Wnioski:**

**Porównując spectogramy utworzone za pomocą własnej funkcji z zadaniem 4.3 można zauważyć ,iż dla odpowiadających sobie sygnałów są takie same.Podsumowując funkcja została napisana poprawnie.**