Višebrzinska obrada signala

Name: Slađan Kantar
Date: 22.01.2018.

Instrukcije

Potrebno je poslati odgovore na pitanja, grafike i kodove (.m fajlove). Odgovore na pitanja i grafike treba poslati u posebnom fajlu (u .doc, .docx ili .pdf formatu), a kodove kao posebne .m fajlove. Početna faza obe komponente je 0.

Zadatak I

Generisati signal

$$x(t) = \sin(2\pi 130t) + \sin(2\pi 288t)$$

koji je sastavljen od dve prostoperiodične komponente. Frekvencija odabiranja signala je $f_s = 1200$Hz. Početna faza obe komponente je 0.

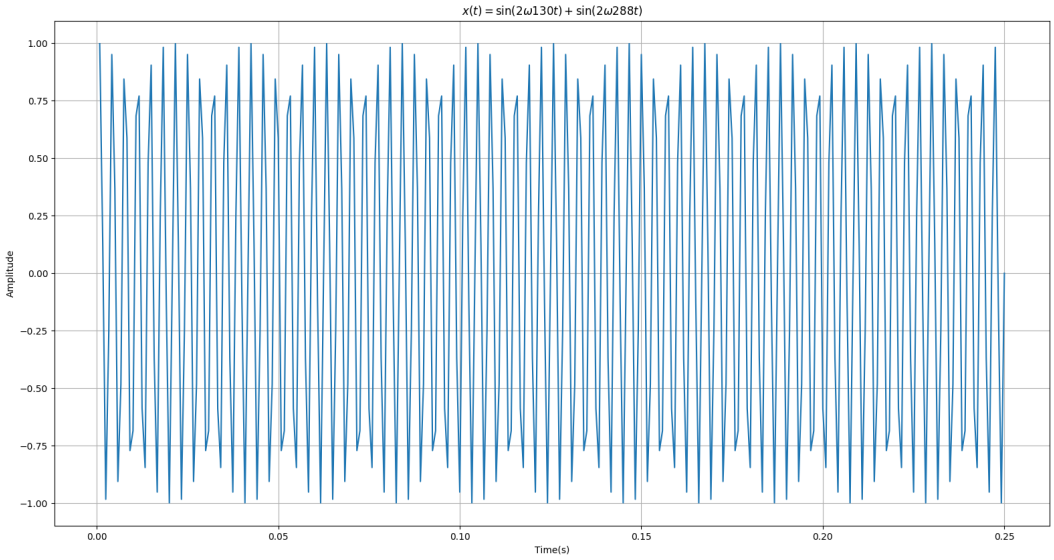

Fig. 1: Graički prikaz signala

1. Odrediti da li će pri dužini niza N=300 doći do curenja spektra.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Fs = 1200       # Sample frequence
N = 300         # Number of sample points
T = 1.0 / Fs    # Sample spacing
x = np.linspace(T, N * T, N)

# I Component
f1 = 130
y1 = np.sin(f1 * 2.0 * np.pi * x)

# II Component
f2 = 288
y2 = np.sin(f2 * 2.0 * np.pi * x)

# Complete Component
y = y1 + y2

# Calculate FFT
Y = fft(y)

xf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

# Display FFT
plt.figure(1)
plt.stem(xf, 2.0 / N * np.abs(Y[0:N//2]))
plt.title('FFT Spectrum\n$x(t) = \sin(2\pi130t) + \sin(2\pi288t)$')
plt.xlabel('Frequency [Hz]')
plt.grid()
plt.show()
```

Listing 1: Kod za prikazivanje amplitudskog spektra za N = 300 napisan u programskom jeziku python. Kod je podeljen na sekcije: Generisanje pojedinačnih komponenti signala, sabiranje komponenti(ulazni signal), računanje DFT ulaznog signala, prikazivanje amplitudski spektra

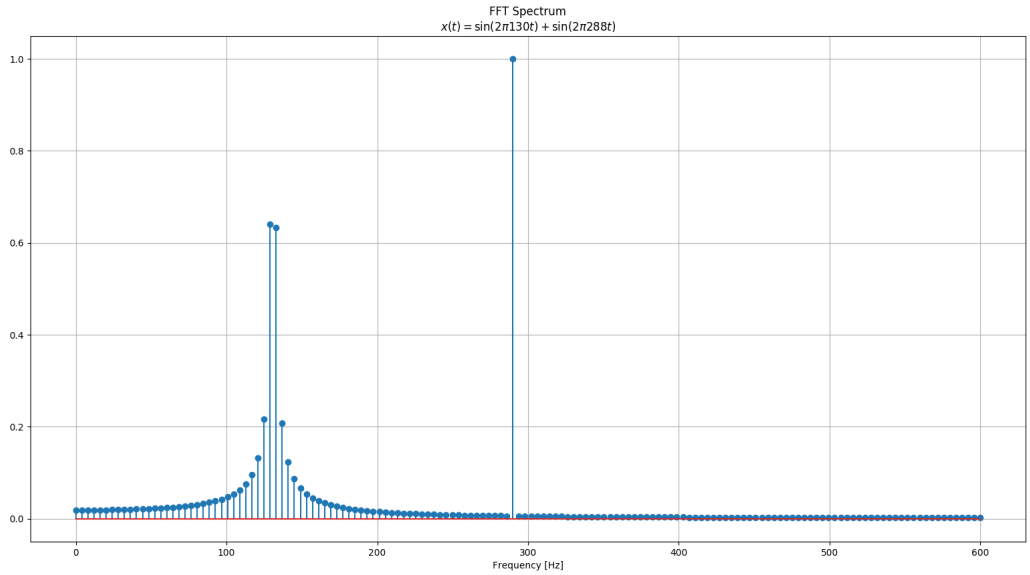

Fig. 2: Amplitudni spektar signala za 300 odbiraka

Ans: Kao što možemo primetiti sa grafika, pri dužini niza od 300 odbiraka dolazi do curenja spektra oko komponente na učestanosti 130Hz, stoga 300 odbiraka ne predstavlja dobar odabir.

2. Odrediti računski kolika je najmanja dužina niza koja obezbeđuje da ne dođe do curenja spektra i prikazati grafički takav spektar.

$$f_1 = \frac{F_s * k_1}{N_1} \qquad f_2 = \frac{F_s * k_2}{N_2}$$

$$f_1 = 130\text{Hz} \wedge F_s = 1200\text{Hz} \qquad f_2 = 288\text{Hz} \wedge F_s = 1200\text{Hz}$$

$$\frac{130}{1200} = \frac{k_1}{N_1} \qquad \frac{288}{1200} = \frac{k_2}{N_2}$$

$$\frac{13}{120} = \frac{k_1}{N_1} \qquad \frac{6}{25} = \frac{k_2}{N_2}$$

$$LCM(120, 25) = 600$$

Ans: Minimalan broj odbiraka pri kom neće doći do curenja spektra je 600. Do curenja neće doći ukoliko je broj odbiraka jednak:

$$N = N_{min} * k, \forall k = 1, 2, 3... \wedge N_{min} = 600$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

def GCD(A, B):
    """ Greatest common divisor """
    while B:
        A, B = B, A % B
    return A


def LCM(A, B):
    """ Lowest common denominator """
    return A * B / GCD(A, B)


Fs = 1200    # Sample frequency
f1 = 130     # Frequency of 1st component
f2 = 288     # Frequency of 2nd component

# Number of sample points
N = int(LCM(Fs / GCD(Fs, f1), Fs / GCD(Fs, f2)))     # 600

T = 1.0 / Fs     # Sample spacing
x = np.linspace(T, N * T, N)

# I Component
y1 = np.sin(f1 * 2.0 * np.pi * x)

# II Component
y2 = np.sin(f2 * 2.0 * np.pi * x)

# Complete Signal
y = y1 + y2
yf = fft(y)

xf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

# Display FFT
plt.figure()
plt.stem(xf, 2.0 / N * np.abs(yf[0:N//2]))
plt.title('FFT Spectrum\n$x(t) = \sin(2\pi130t) + \sin(2\pi288t)$')
plt.xlabel('Frequency [Hz]')
plt.grid()
plt.show()
```

Listing 2: Kod za prikazivanje amplitudskog spektra za minimalan broj odbiraka takav da ne dođe do curenja spektra, napisan u programskom jeziku python. Kod je podeljen na sekcije: Proračunavanje minimalnog broja odbiraka (bez curenja spektra), Generisanje pojedinačnih komponenti signala, sabiranje komponenti(ulazni signal), računanje DFT ulaznog signala, prikazivanje amplitudski spektra

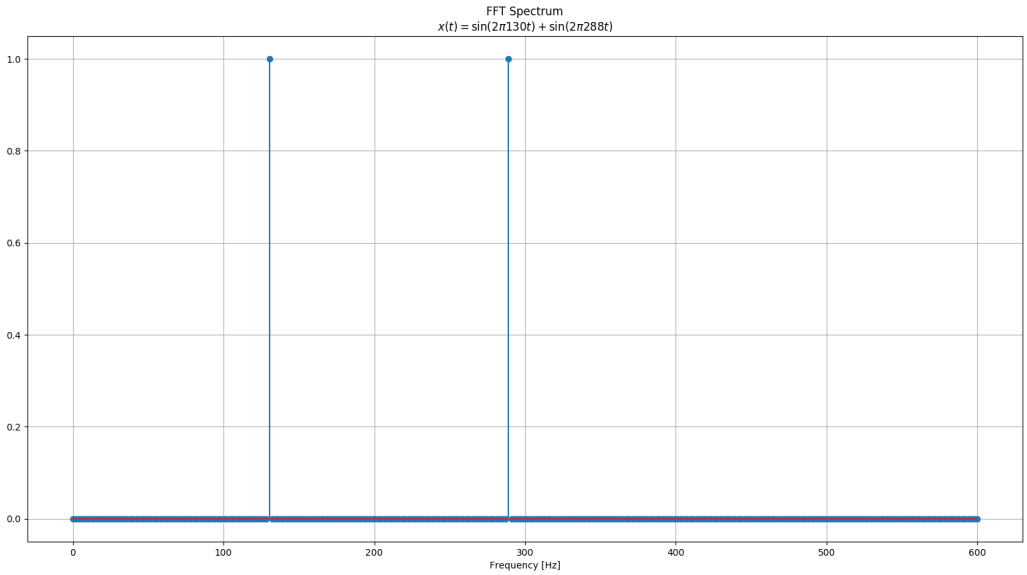

Fig. 3: Amplitudni spektar signala za minimalan broj odbiraka, tako izabran da ne dođe do curenja spektra

3. Odrediti da li se na osnovu DFT-a niza u N=50 tačaka može precizno odrediti odnos amplituda prostoperiodičnih komponenti koje čine posmatrani signal.

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

from scipy.fftpack import fft

Fs = 1200       # Sample frequence
N = 50          # Number of sample points
T = 1.0 / Fs    # Sample spacing
x = np.linspace(T, N * T, N)

# I Component
f1 = 130
y1 = np.sin(f1 * 2.0 * np.pi * x)

# II Component
f2 = 288
y2 = np.sin(f2 * 2.0 * np.pi * x)

# Complete Component
y = y1 + y2
Y = fft(y)

xf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

# Display FFT
plt.figure(1)
plt.stem(xf, 2.0 / N * np.abs(Y[0:N//2]))
plt.title('FFT Spectrum\n$x(t) = \sin(2\pi130t) + \sin(2\pi288t)$')
plt.xlabel('Frequency [Hz]')
plt.grid()
plt.show()
```

Listing 3: Kod za prikazivanje amplitudskog spektra za N = 50 napisan u programskom jeziku python. Kod je podeljen na sekcije: Generisanje pojedinačnih komponenti signala, sabiranje komponenti(ulazni signal), računanje DFT ulaznog signala, prikazivanje amplitudski spektra

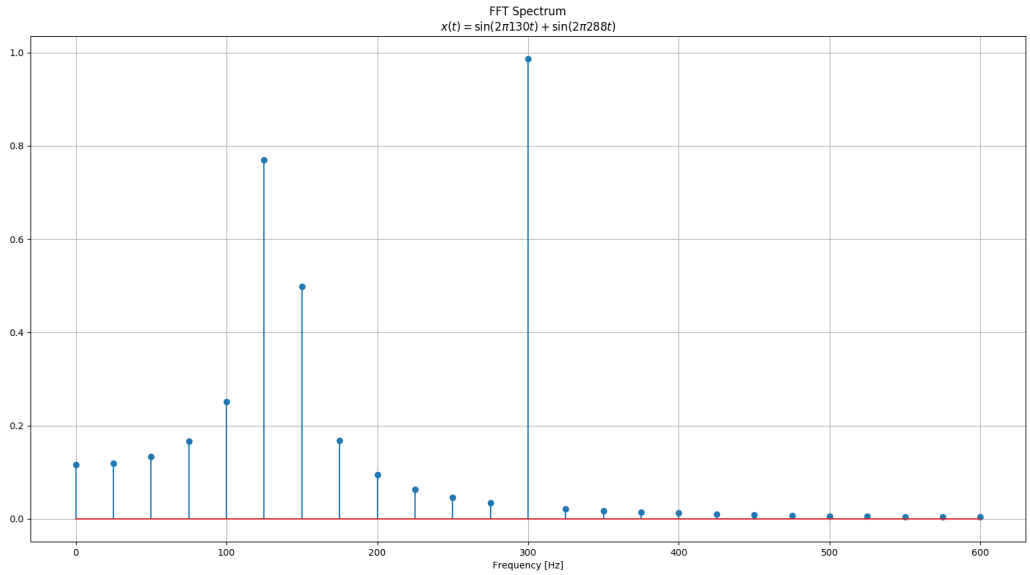

Fig. 4: Amplitudni spektar signala za 50 odbiraka

Ans: Nemoguće je precizno odrediti odnos amplituda upravo zbog toga što imamo izraženo spektra oko prve komponente.

4. Menjati fazu jedne od komponenti signala za dužinu niza N=1200 i ustanoviti na koji način se menjaju modul, realni i imaginarni deo DFT niza. Prikazati grafički promenu za najmanje tri različite faze.

   Ans: Vršena je promena faze druge komponente i to redom sa veličinama: $0, \pi/4, \pi/2, \pi$. U nastavku su prikazani grafici sa navedenim pomerajima redom. Sa grafika možemo doći do sledećih zaključaka:

   Pomeranje faze ne utiče na amplitudski spektar.

   Pomeranje faze druge komponente ne utiče na realni i imaginarni deo prve komponente.

   Menjanjm faze druge komponente utičemo na vrednosti realnog i imaginarnog dela te komponente

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft

phases, labels = [0, np.pi / 4, np.pi / 2, np.pi], ['', '$\pi / 4$', '$\pi / 2$', '$\pi$']
Fs = N = 1200
T = 1.0 / Fs
x = np.linspace(T, N * T, N)

# I Component
f1, f2 = 130, 288
y1 = np.sin(f1 * 2.0 * np.pi * x)
y1f = fft(y1)

# II Component
for i, phase in enumerate(phases):
    y2 = np.sin(phase + f2 * 2.0 * np.pi * x)
    y2f = fft(y2)

    # Complete Component
    y = y1 + y2
    yf = fft(y)
    xf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    plt.figure(i + 1)
    # Module
    plt.subplot(3, 1, 1)
    plt.stem(xf, 2.0 / N * np.abs(yf[0:N//2]))
    plt.yticks([-1, -0.5, 0, 0.5, 1])
    plt.title('Module {}'.format(labels[i]), position=(0.9, 0.8))
    plt.grid()
    # Real
    plt.subplot(3, 1, 2)
    plt.stem(xf, 2.0 / N * np.real(yf[0:N // 2]))
    plt.yticks([-1, -0.5, 0, 0.5, 1])
    plt.title('Real {}'.format(labels[i]), position=(0.9, 0.8))
    plt.grid()
    # Imag
    plt.subplot(3, 1, 3)
    plt.stem(xf, 2.0 / N * np.imag(yf[0:N // 2]))
    plt.yticks([-1, -0.5, 0, 0.5, 1])
    plt.title('Imag {}'.format(labels[i]), position=(0.9, 0.8))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show(block=False)
```

Listing 4: Kod za prikazivanje amplitudskog spektra za N = 1200, i fazne pomeraje od $0, \pi/4, \pi/2, \pi$, napisan u programskom jeziku python

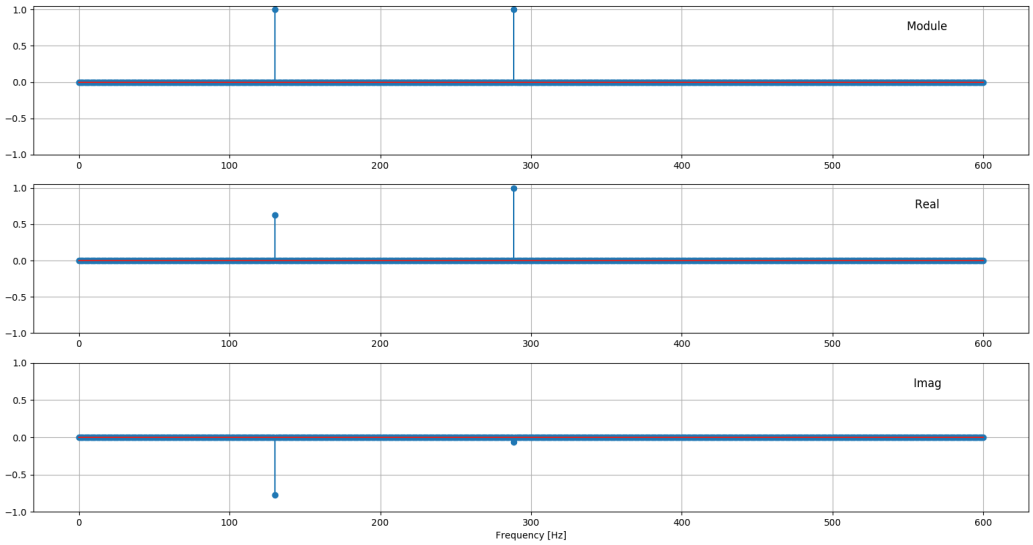

Fig. 5: Amplitudni spektar signala za pomeraj komponente 0

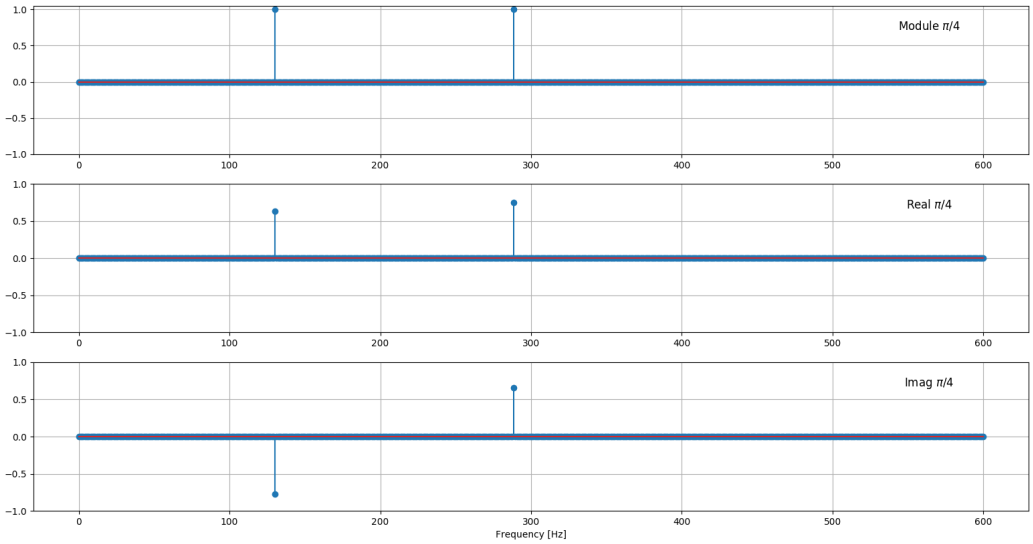

Fig. 6: Amplitudni spektar signala za pomeraj komponente $\pi/4$

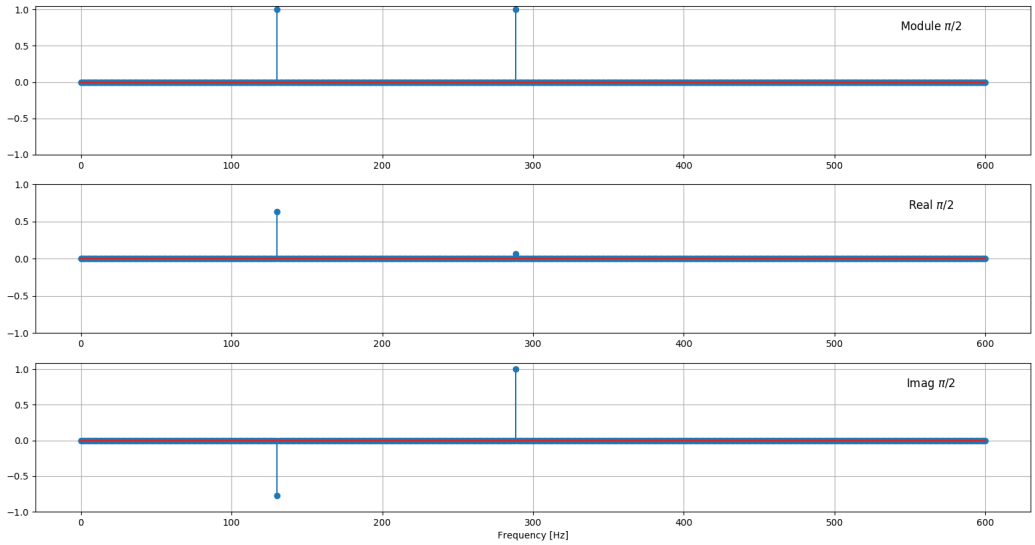

Fig. 7: Amplitudni spektar signala za pomeraj komponente $\pi/2$

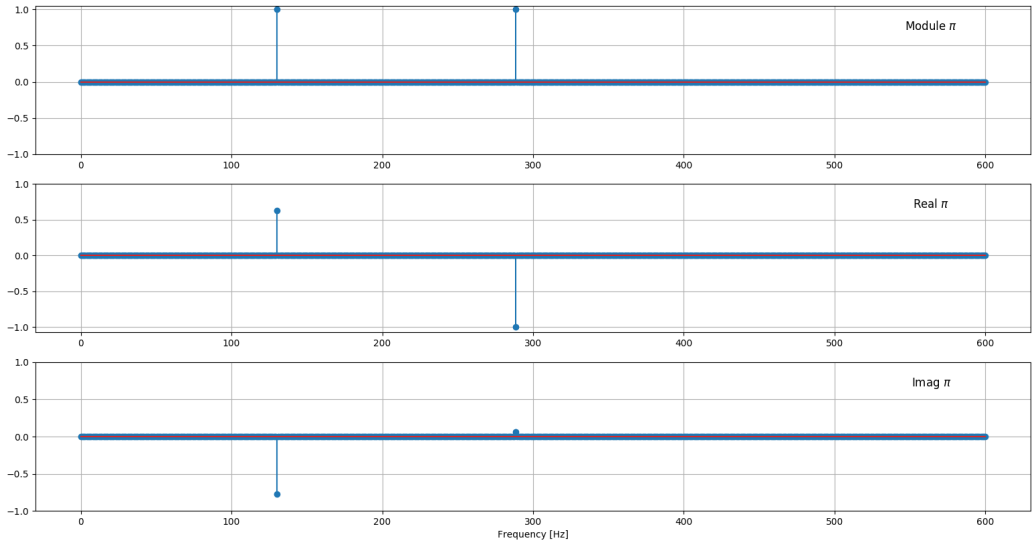

Fig. 8: Amplitudni spektar signala za pomeraj komponente π

5. Niz dužine 600 dopuniti nulama do dužina 800, 1000 i 1200, te na osnovu odgovarajućih DFT-a nizova objasniti da li postoje elementi u DFT-u koji odgovaraju frekvencijama prostoperiodičnih komponenti. Šta se postiže dopunjavanjem niza nulama (veća rezolucija ili veća gustina spektra)?

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

new_sizes = [800, 1000, 1200]

Fs, SN = 1200, 600
T = 1.0 / Fs

sx = np.linspace(T, SN * T, SN)

for i, N in enumerate(new_sizes):
    x = np.pad(sx, (0, N - SN), 'constant')

    # I Component
    f1 = 130
    y1 = np.sin(f1 * 2.0 * np.pi * x)
    y1f = fft(y1)

    # II Component
    f2 = 288
    y2 = np.sin(f2 * 2.0 * np.pi * x)
    y2f = fft(y2)

    # Complete Component
    y = y1 + y2
    yf = fft(y)

    xf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    plt.figure(i + 1)
    plt.stem(xf, 2.0 / N * np.abs(yf[0:N//2]))
    plt.title('FFT Spectrum N = {}'.format(N))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 5: Kod za prikazivanje amplitudskog spektra za N = 600 uz dopunu nulama do N = 800, 1000 i 1200, napisan u programskom jeziku python

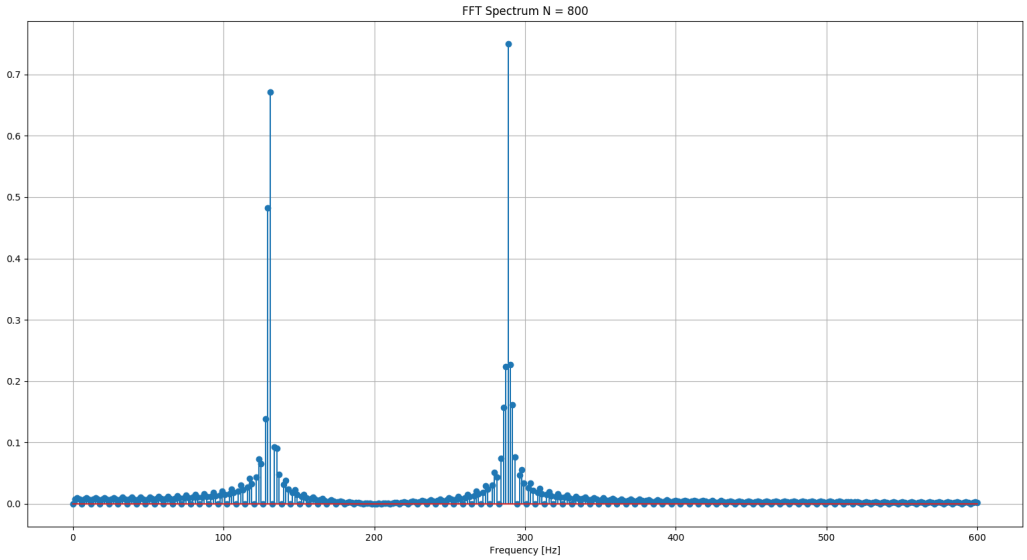

Fig. 9: Amplitudni spektar signala za N = 800

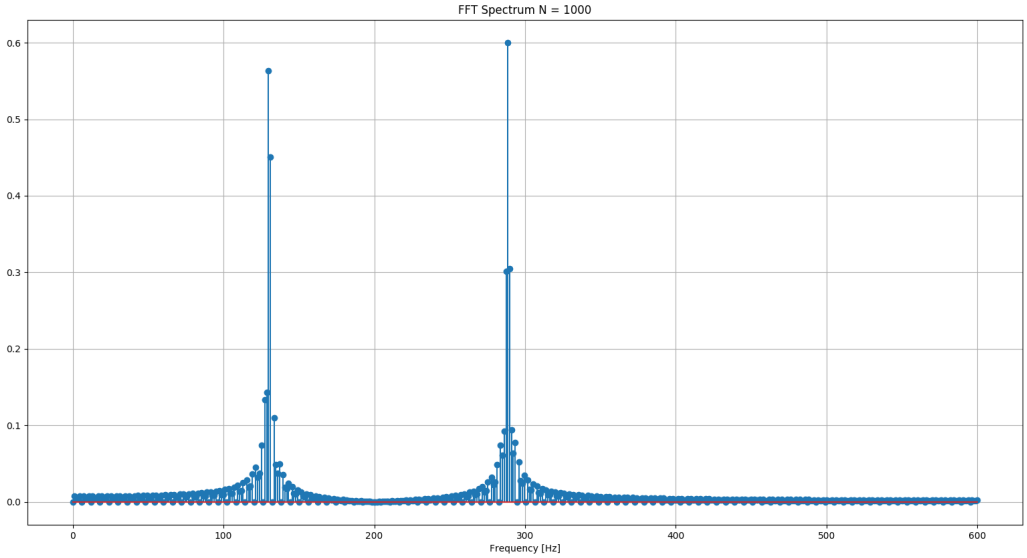

Fig. 10: Amplitudni spektar signala za N = 1000

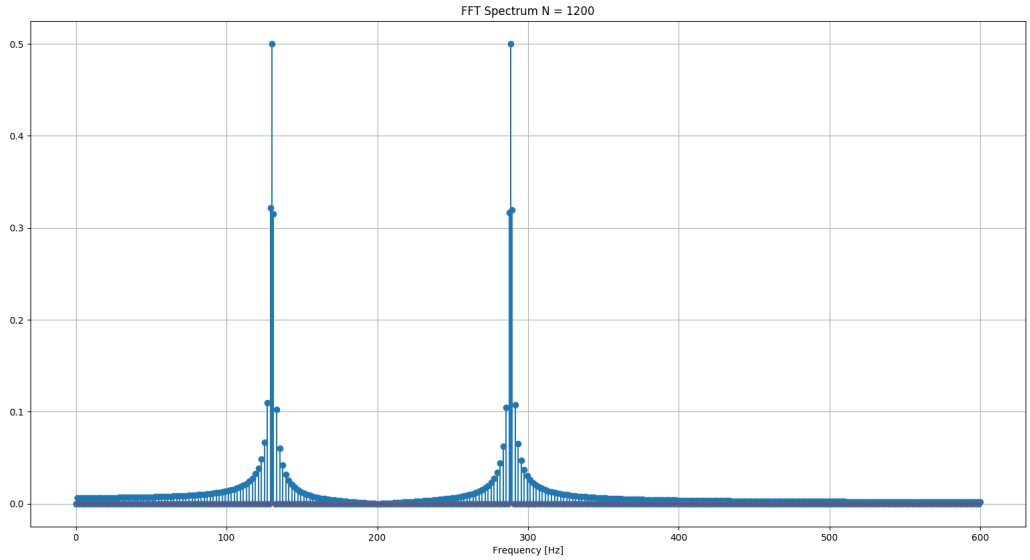

Fig. 11: Amplitudni spektar signala za N = 1200

Ans: Dopunjavanjem niza nulama pojavljuje se curenje spektra oko obe komponente. Takođe, rezultujući grafik ne sadrži realne komponente 130Hz i 288Hz, već su one malo pomerene(131.1, 289.2). Međutim popunjavanjem nulama postiže se veća rezolucija te se jasnije uočavaju spektralne komponente.

Zadatak II

Generisati signal

$$x(t) = 2.3\sin(2\pi 3t) + e(t)$$

gde je e(t) ABGŠ. Frekvencija odabiranja signala je $f_s = 200$Hz. Dužina niza je N=150. Amplituda signala je 2.3.

1. Odrediti kako se menja spektar:
    - Kada nema šuma
    - Kada je srednja snaga šuma 1
    - Kada je srednja snaga šuma 3
    - Kada je srednja snaga šuma 6

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Fs, N = 200, 150        # Sample frequence, Number of sample points
T = 1.0 / Fs            # Sample spacing
t = np.linspace(T, N * T, N)

A, f = 2.3, 3
x_clear = A * np.sin(f * 2.0 * np.pi * t)

powers, colors = [0, 1, 3, 6], ['r', 'g', 'b', 'y']
for i, power in enumerate(powers):
    e = np.random.normal(0, 1, N) * np.sqrt(power)
    x = x_clear + e
    xf = fft(x)

    yf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    # Display FFT
    plt.figure(i + 1)
    plt.stem(yf, 2.0 / N * np.abs(xf[0:N // 2]), colors[i],
             markerfmt='{}o'.format(colors[i]))
    plt.title('FFT Spectrum AWGN Power {}'.format(powers[i]))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 6: Kod za prikazivanje amplitudskog spektra signala sa aditivnim belim gausovim šumom snage 0, 1, 3, 6, napisan u programskom jeziku python

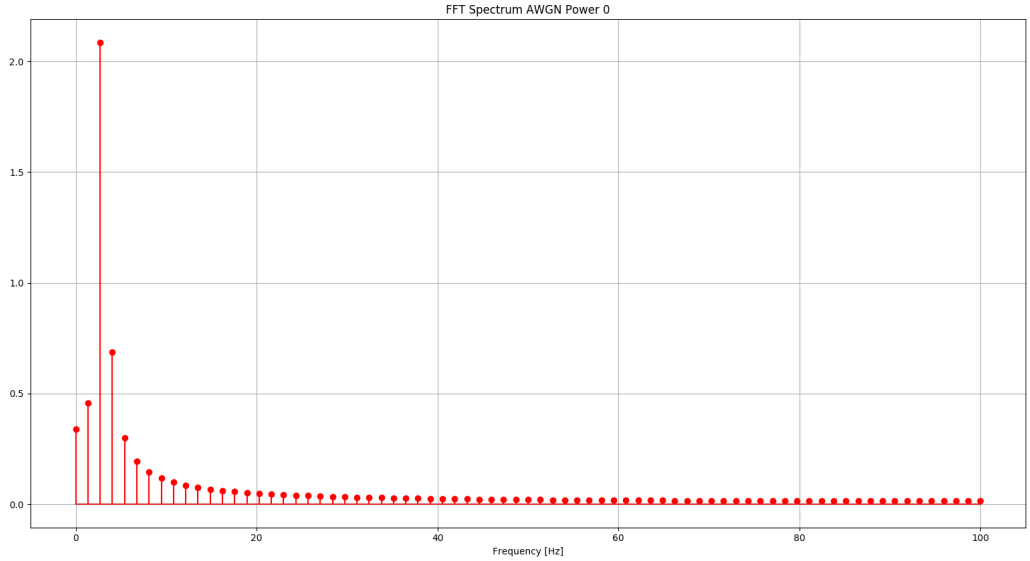

Fig. 12: Amplitudni spektar signala za snagu gausovog šuma 0

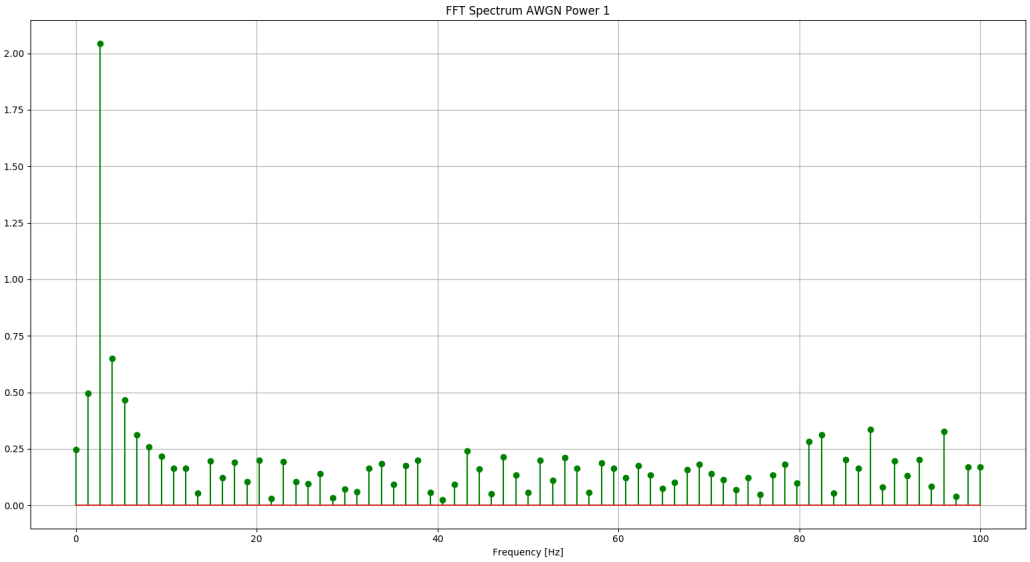

Fig. 13: Amplitudni spektar signala za snagu gausovog šuma 1

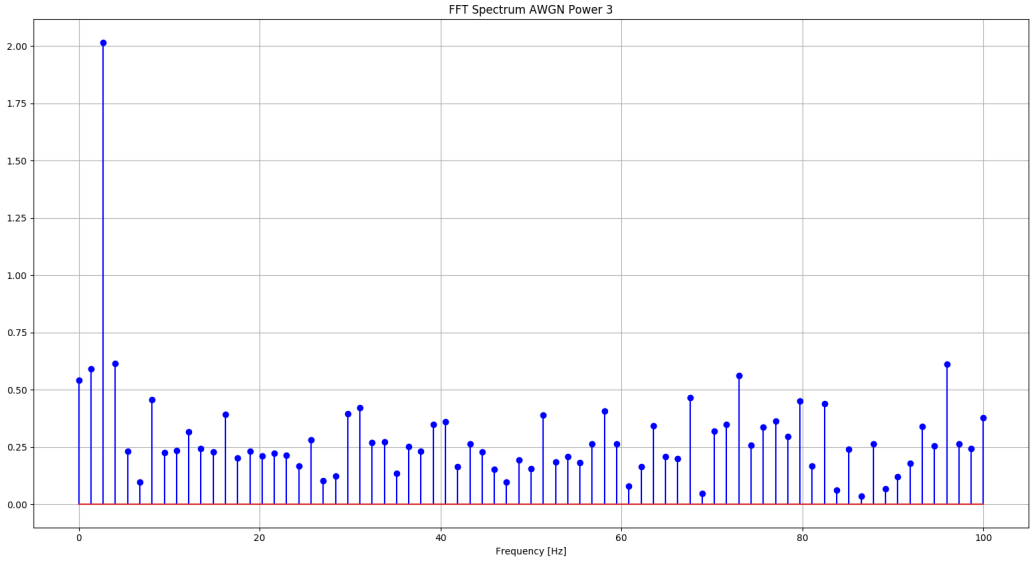

Fig. 14: Amplitudni spektar signala za snagu gausovog šuma 3

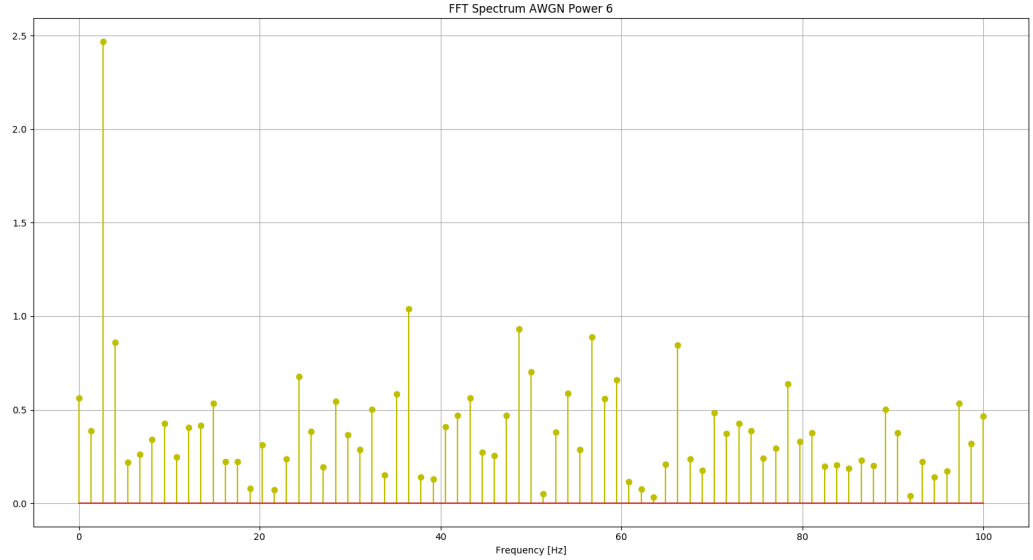

Fig. 15: Amplitudni spektar signala za snagu gausovog šuma 6

Ans: Povećanjem šuma sve je teže uočiti komponentu glavnog signala. Na ovom grafiku vidimo i curenje spektra.

2. Ponoviti postupak iz tačke 1 za frekvenciju signala od 20Hz i ustanoviti u kojem slučaju je teže odrediti frekvenciju korisnog, prostoperiodičnog signala.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Fs = 200          # Sample frequence
N = 150          # Number of sample points
T = 1.0 / Fs     # Sample spacing
t = np.linspace(T, N * T, N)

A = 2.3
f = 20
x_clear = A * np.sin(f * 2.0 * np.pi * t)

powers, colors = [0, 1, 3, 6], ['r', 'g', 'b', 'y']
for i, power in enumerate(powers):
    e = np.random.normal(0, 1, N) * np.sqrt(power)
    x = x_clear + e
    xf = fft(x)

    yf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    # Display FFT
    plt.figure(i + 1)
    plt.stem(
        yf,
        2.0 / N * np.abs(xf[0:N // 2]),
        colors[i],
        markerfmt='{}o'.format(colors[i])
    )
    plt.title('FFT Spectrum AWGN Power {}'.format(powers[i]))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 7: Kod za prikazivanje amplitudskog spektra signala sa aditivnim belim gausovim šumom snage 0, 1, 3, 6, napisan u programskom jeziku python

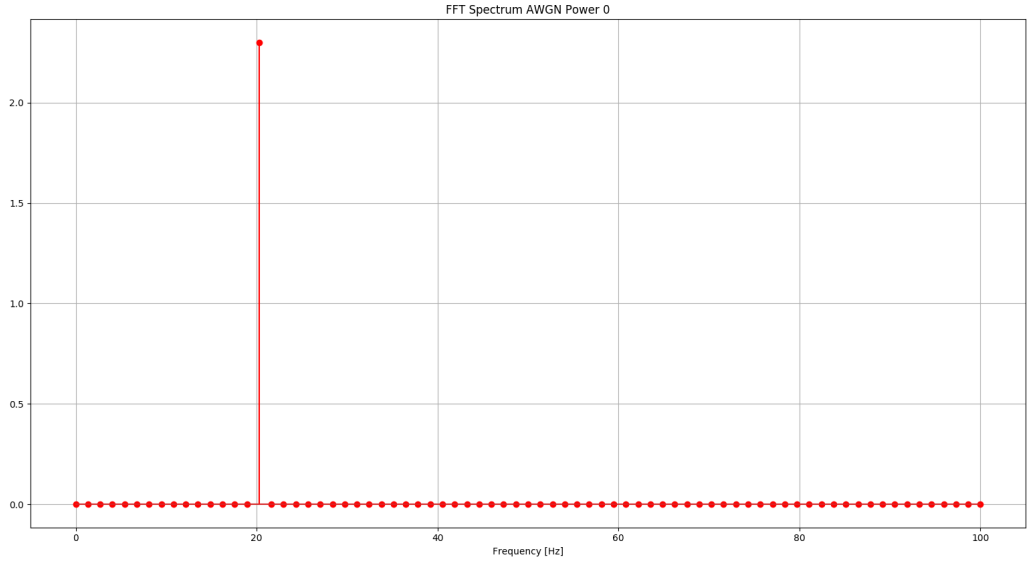

Fig. 16: Amplitudni spektar signala za snagu gausovog šuma 0

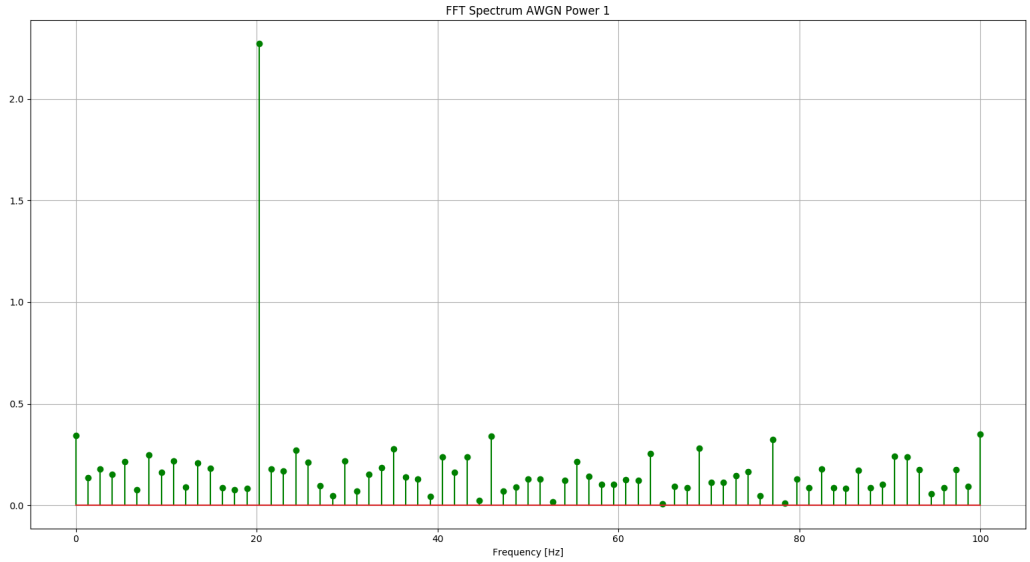

Fig. 17: Amplitudni spektar signala za snagu gausovog šuma 1

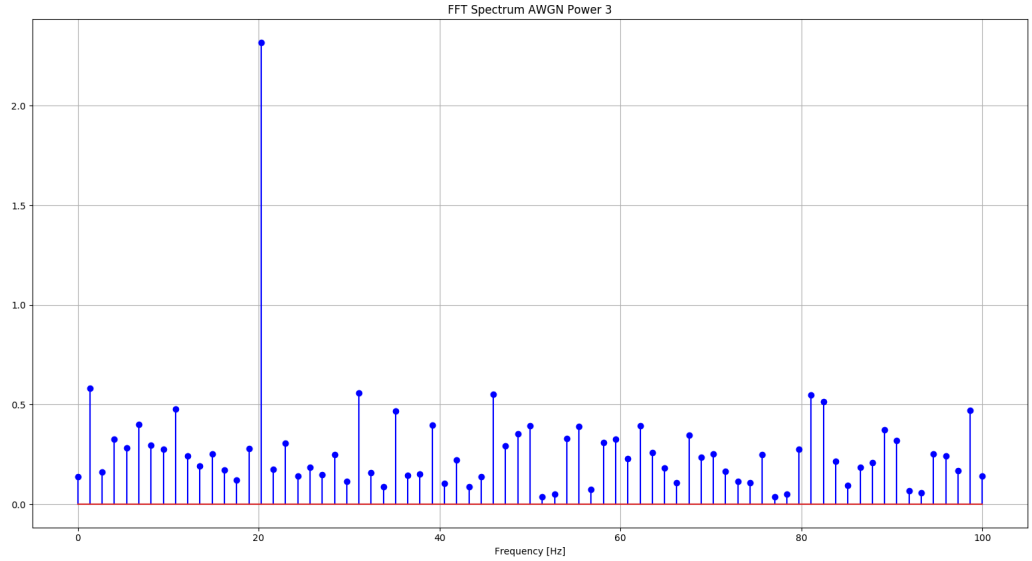

Fig. 18: Amplitudni spektar signala za snagu gausovog šuma 3

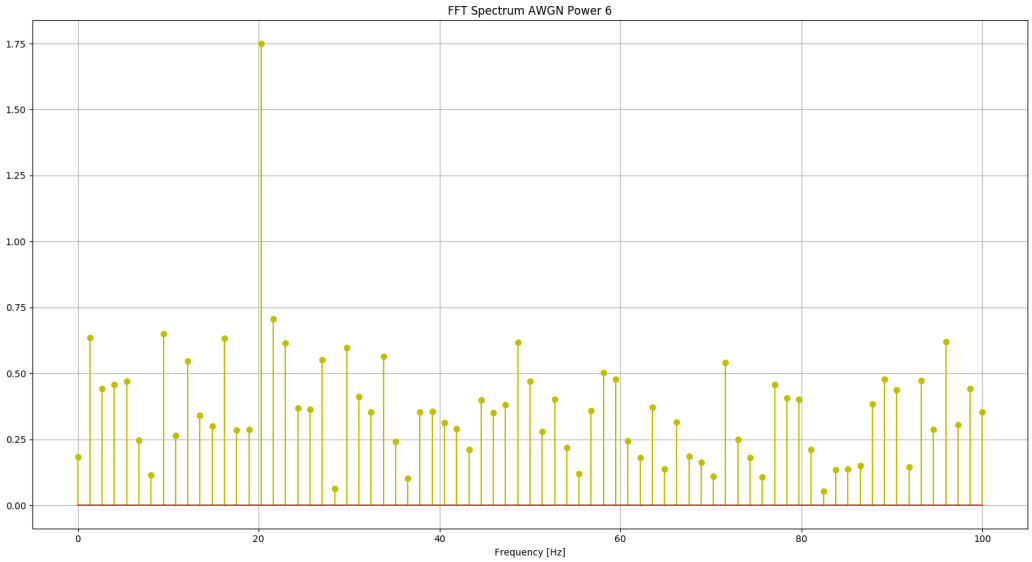

Fig. 19: Amplitudni spektar signala za snagu gausovog šuma 6

Ans: Sa prethodno navedenih grafika možemo uočiti da je glavna komponenta jasno vidljiva u odnosu na komponente šuma.

3. Ponoviti postupak iz tačke 1 za dužinu niza od N=200. Objasniti dobijeni rezultat.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Fs = 200        # Sample frequence
N = 200         # Number of sample points
T = 1.0 / Fs    # Sample spacing
t = np.linspace(T, N * T, N)

A = 2.3
f = 3
x_clear = A * np.sin(f * 2.0 * np.pi * t)

powers = [0, 1, 3, 6]
colors = ['r', 'g', 'b', 'y']
for i, power in enumerate(powers):
    e = np.random.normal(0, 1, N) * np.sqrt(power)
    x = x_clear + e
    xf = fft(x)

    yf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    # Display FFT
    plt.figure(i + 1)
    plt.stem(
        yf,
        2.0 / N * np.abs(xf[0:N // 2]),
        colors[i],
        markerfmt='{}o'.format(colors[i])
    )
    plt.title('FFT Spectrum AWGN Power {}'.format(powers[i]))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 8: Kod za prikazivanje amplitudskog spektra signala sa aditivnim belim gausovim šumom snage 0, 1, 3, 6, napisan u programskom jeziku python

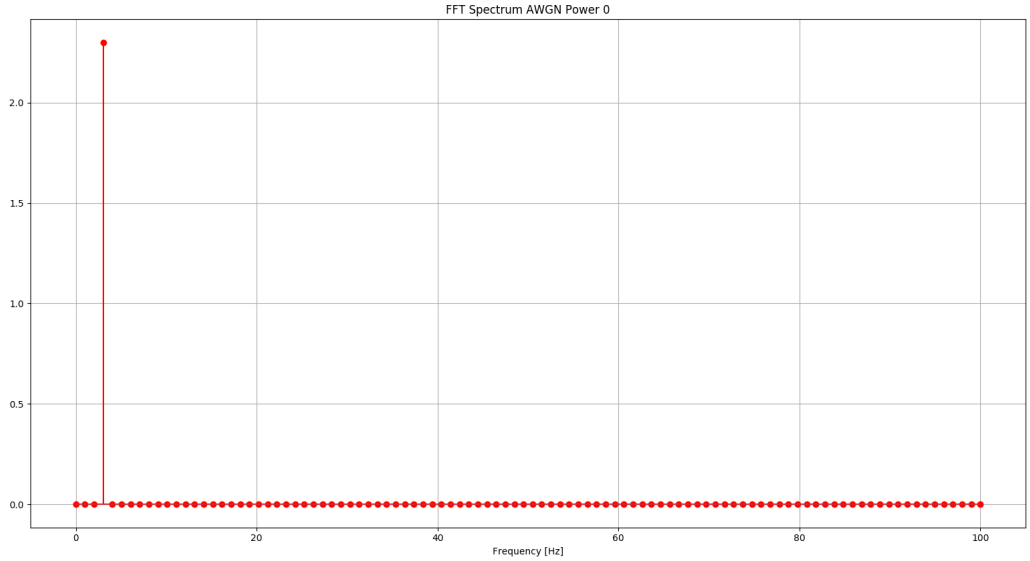

Fig. 20: Amplitudni spektar signala za snagu gausovog šuma 0

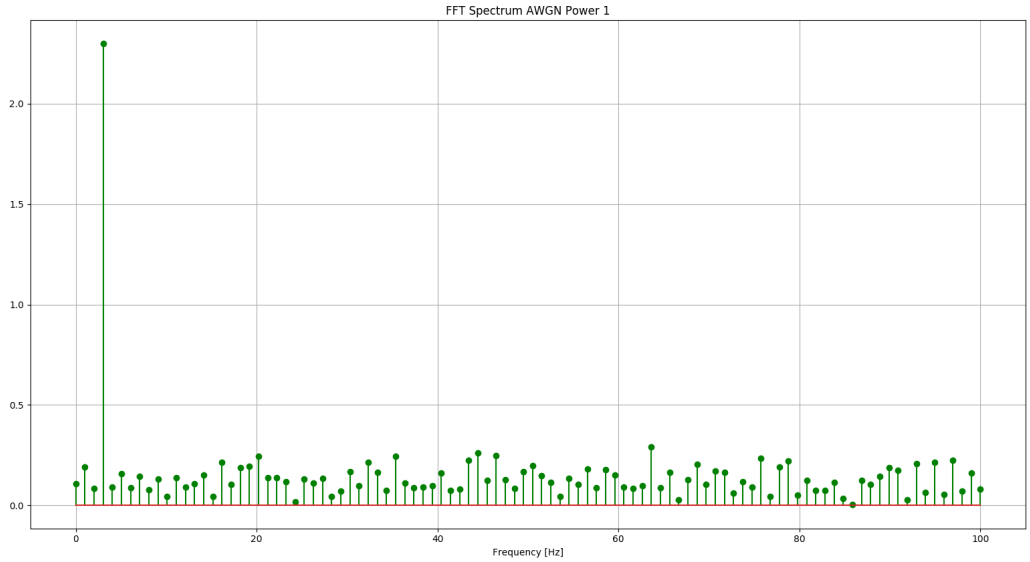

Fig. 21: Amplitudni spektar signala za snagu gausovog šuma 1

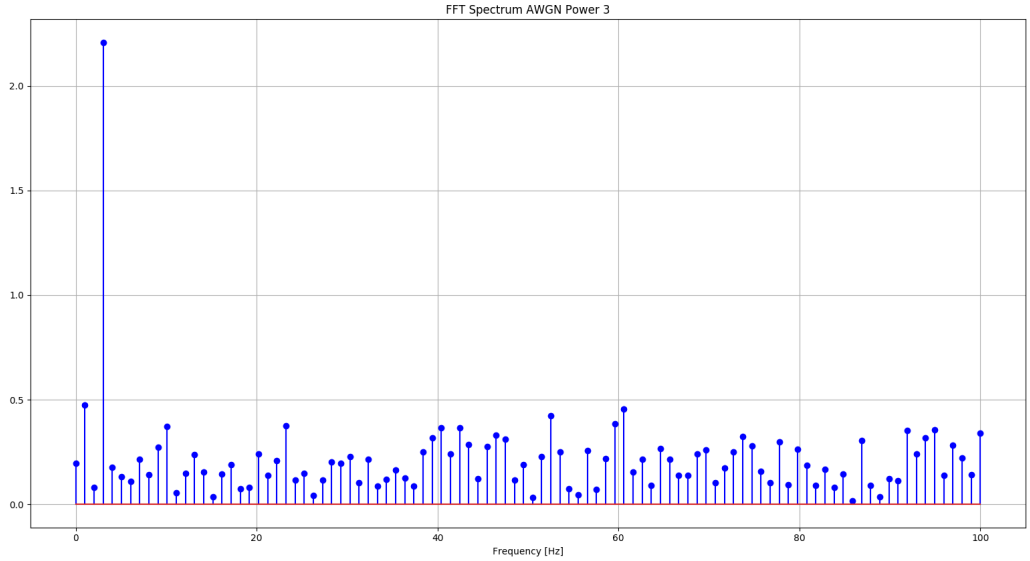

Fig. 22: Amplitudni spektar signala za snagu gausovog šuma 3

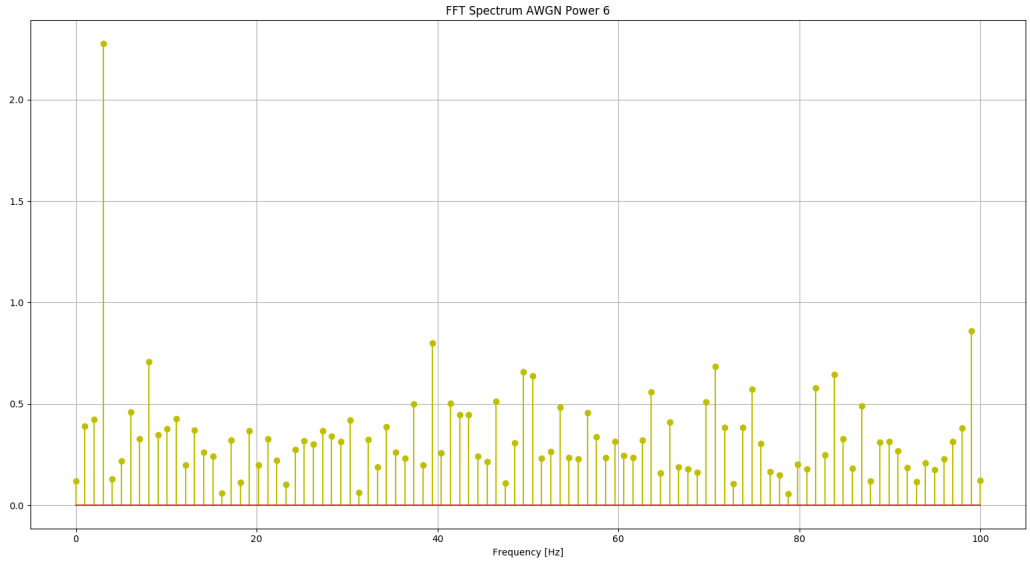

Fig. 23: Amplitudni spektar signala za snagu gausovog šuma 6

Ans: Prilikom ovakvog odabira broja odbiraka ne dolazi do curenja spektra stoga je lakše uočiti komponente korisnog signala.

4. Ponoviti postupak iz tačke 1 za dužinu niza od N=320. Objasniti dobijeni rezultat.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

Fs = 200        # Sample frequence
N = 320         # Number of sample points
T = 1.0 / Fs    # Sample spacing
t = np.linspace(T, N * T, N)

A = 2.3
f = 3
x_clear = A * np.sin(f * 2.0 * np.pi * t)

powers = [0, 1, 3, 6]
colors = ['r', 'g', 'b', 'y']
for i, power in enumerate(powers):
    e = np.random.normal(0, 1, N) * np.sqrt(power)
    x = x_clear + e
    xf = fft(x)

    yf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    # Display FFT
    plt.figure(i + 1)
    plt.stem(
        yf,
        2.0 / N * np.abs(xf[0:N // 2]),
        colors[i],
        markerfmt='{}o'.format(colors[i])
    )
    plt.title('FFT Spectrum AWGN Power {}'.format(powers[i]))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 9: Kod za prikazivanje amplitudskog spektra signala sa aditivnim belim gausovim šumom snage 0, 1, 3, 6, napisan u programskom jeziku python

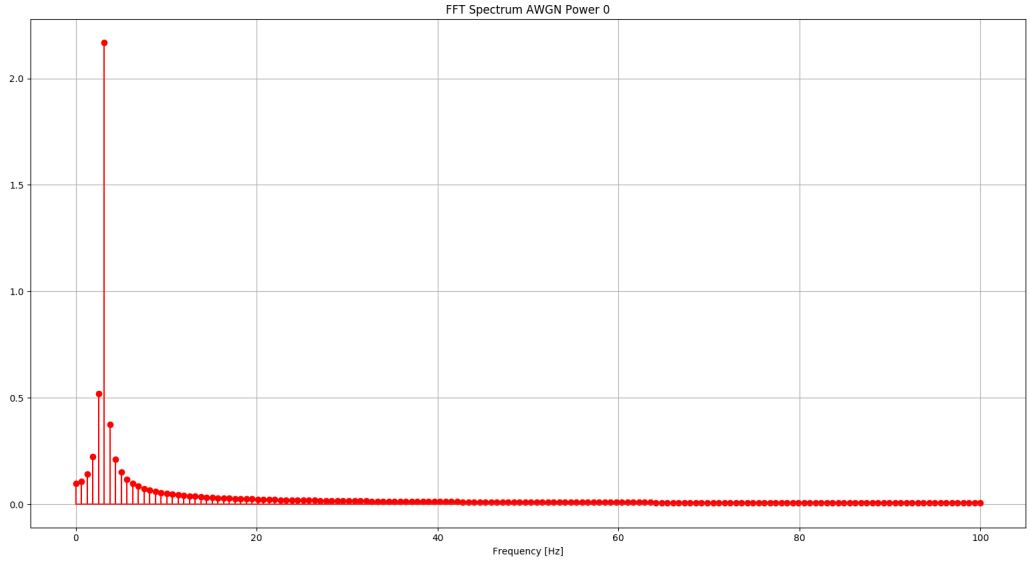

Fig. 24: Amplitudni spektar signala za snagu gausovog šuma 0

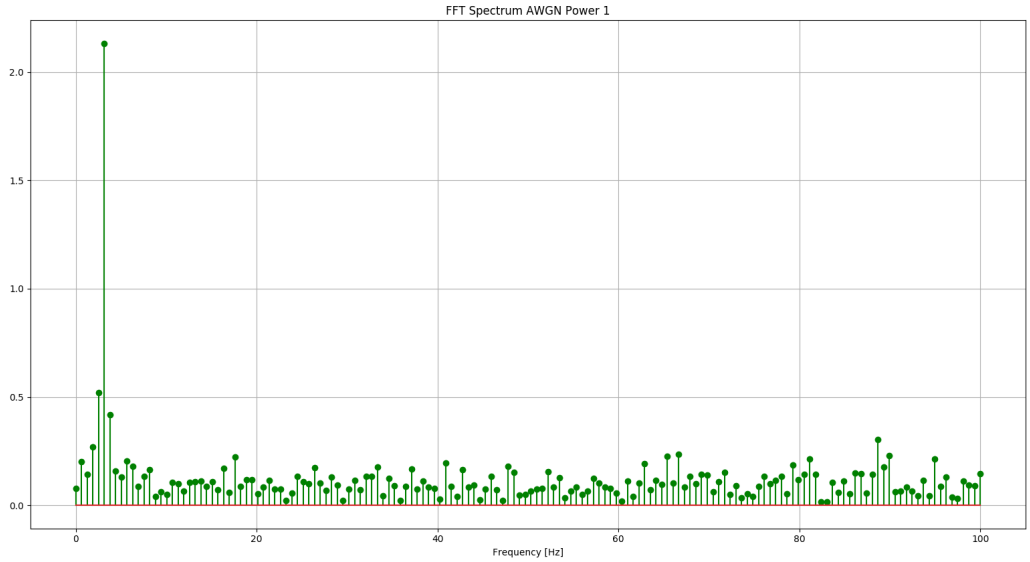

Fig. 25: Amplitudni spektar signala za snagu gausovog šuma 1

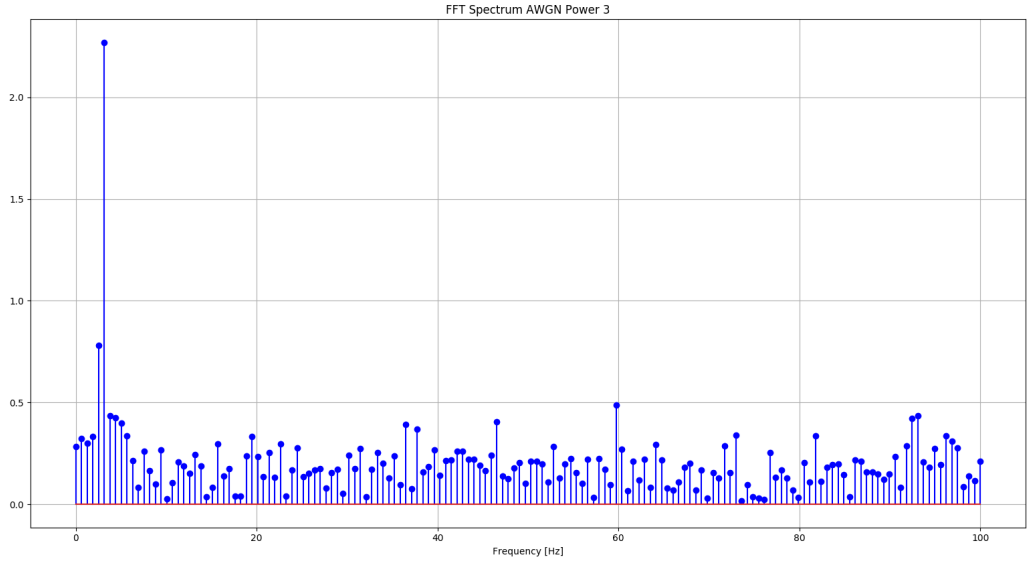

Fig. 26: Amplitudni spektar signala za snagu gausovog šuma 3

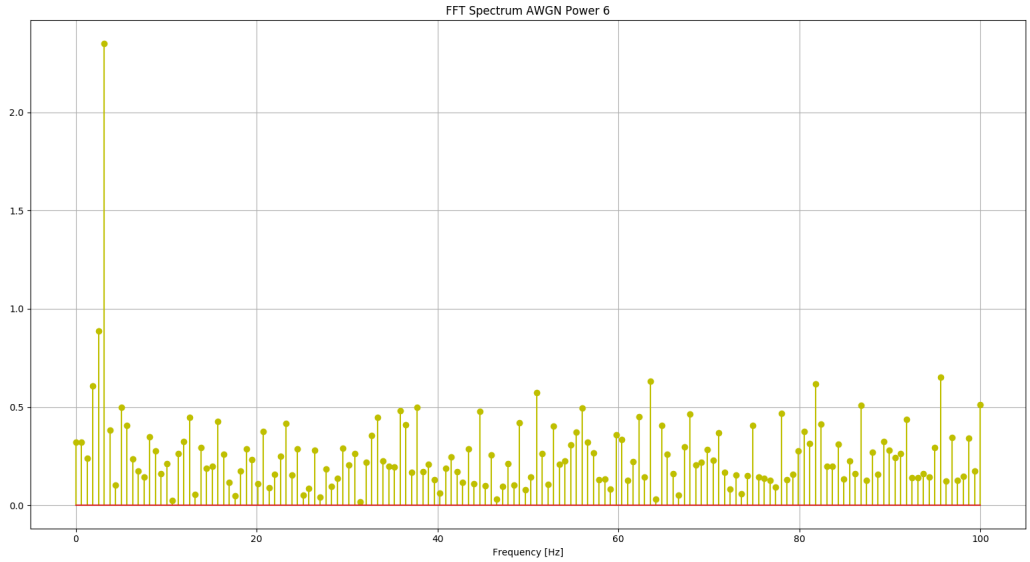

Fig. 27: Amplitudni spektar signala za snagu gausovog šuma 6

Ans: Uočljivo je curenje spektra, međutim, glavna komponenta i dalje ima izrazito veću snagu.

5. Za dužinu niza od N=200 i frekvenciju 3Hz, dopuniti niz nulama do dužine N=300, N=400 i N=500 pri snazi šuma od 3. Objasniti kako utiče dodavanje nula na mogućnost detektovanja frekvencije korisnog signala.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

new_sizes = [200, 300, 400, 500]

Fs = 200
SN = 200    # Starting Number of sample points
T = 1.0 / Fs
t = np.linspace(T, SN * T, SN)

A = 2.3
f = 3
sx = A * np.sin(f * 2.0 * np.pi * t)

power = 3

for i, N in enumerate(new_sizes):
    x = np.pad(sx, (0, N - SN), 'constant')
    e = np.random.normal(0, 1, N) * np.sqrt(power)

    x = x + e
    xf = fft(x)

    yf = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

    # Display FFT
    plt.figure(i + 1)
    plt.stem(yf, 2.0 / N * np.abs(xf[0:N // 2]))
    plt.title('FFT Spectrum AWGN Power 3, Number of sample points'.format(N))
    plt.xlabel('Frequency [Hz]')
    plt.grid()
    plt.show()
```

Listing 10: Kod za prikazivanje amplitudskog spektra signala sa aditivnim belim gausovim šumom snage 0, 1, 3, 6, napisan u programskom jeziku python

Ans: Povećanjem niza nulama, povećava se rezolucija. Međutim, u svim ovim slučajevima glavna komponenta je uočljiva zbog izraženije snage.

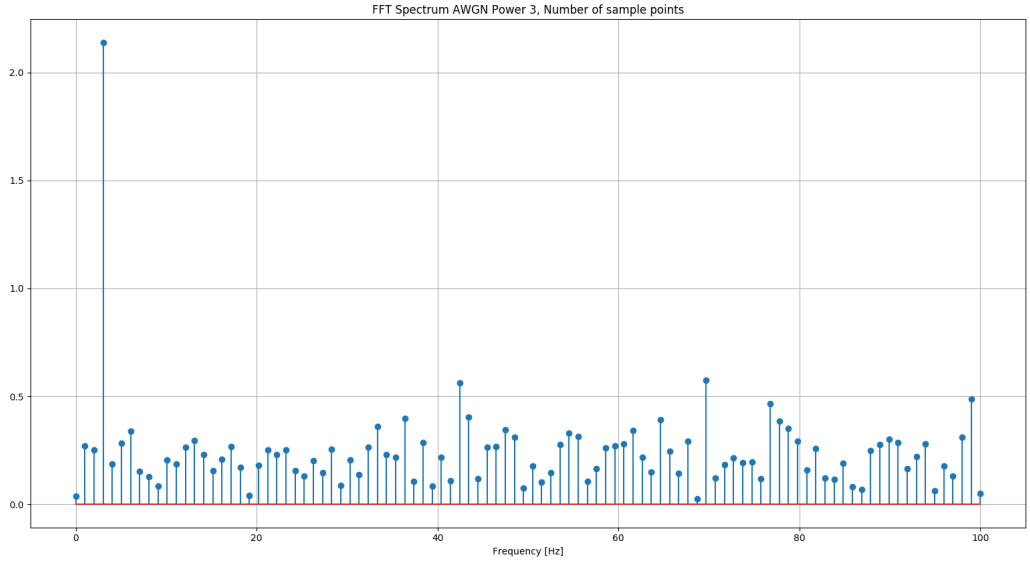

Fig. 28: Amplitudni spektar signala za broj odbiraka 200

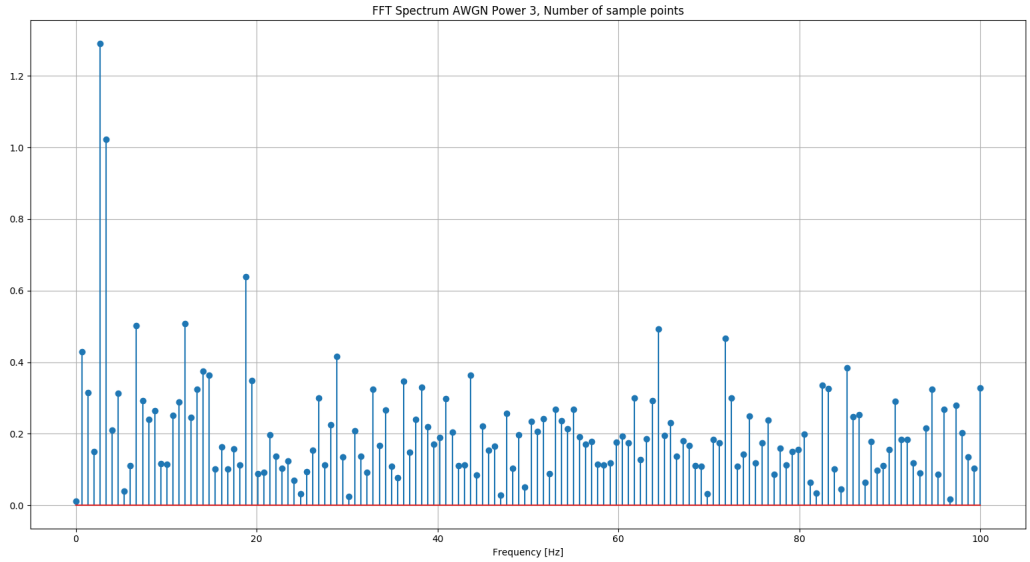

Fig. 29: Amplitudni spektar signala za broj odbiraka 300

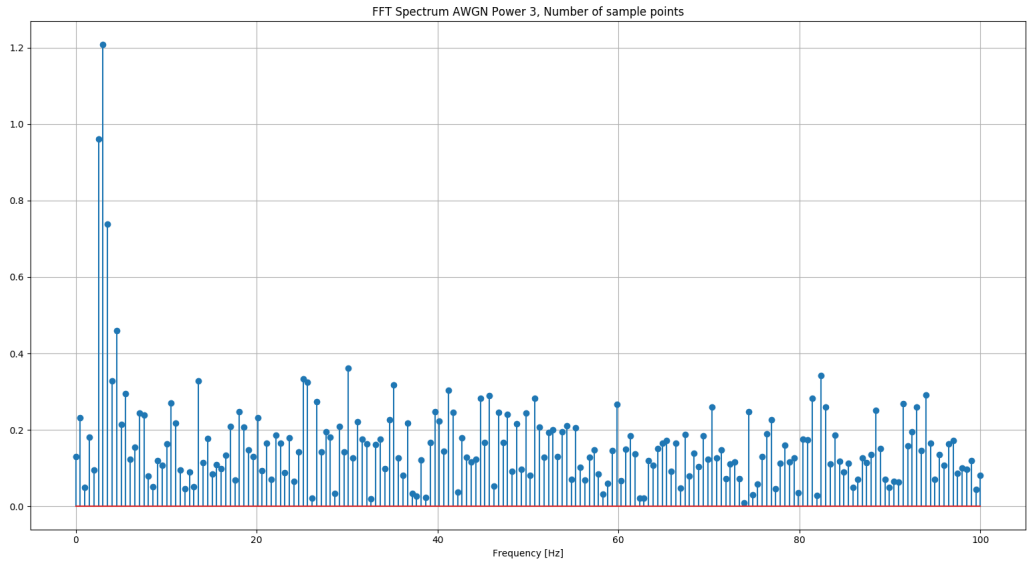

Fig. 30: Amplitudni spektar signala za broj odbiraka 400

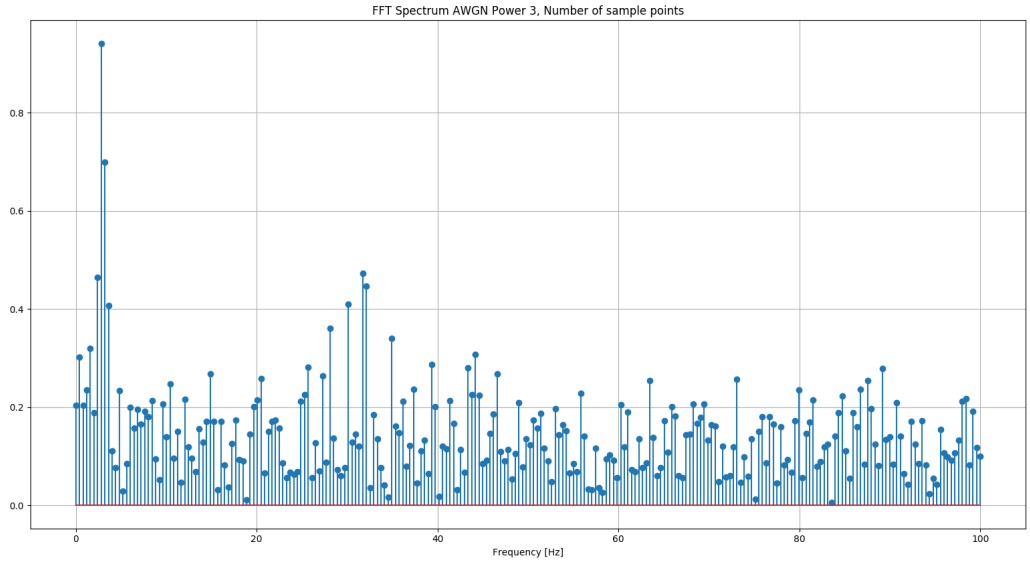

Fig. 31: Amplitudni spektar signala za broj odbiraka 500

Zadatak III

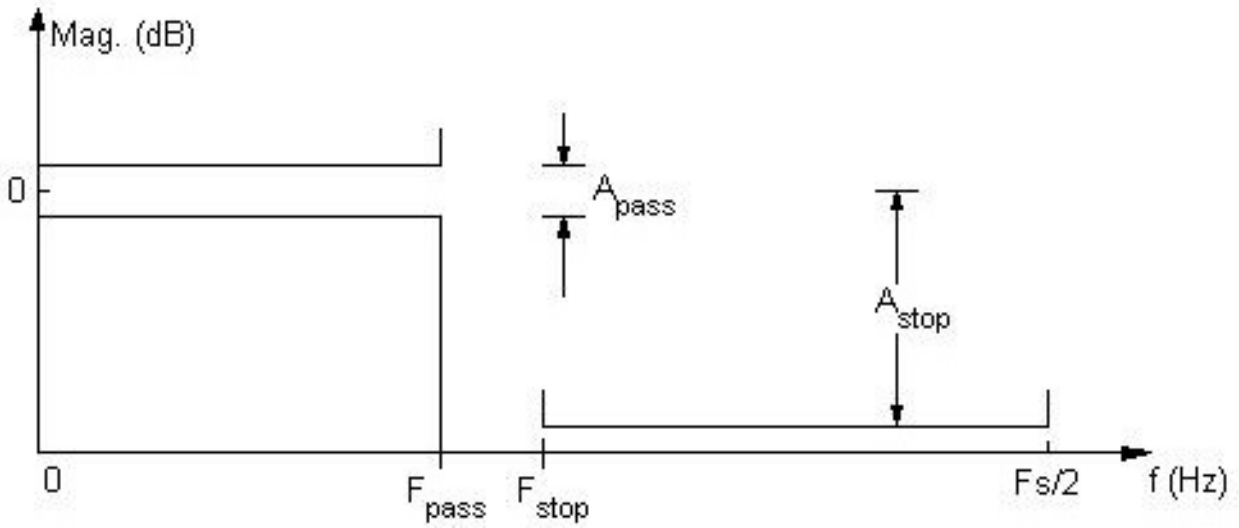

Fig. 32: Specifikacija filtra

Filtar je propusnik niskih frekvencija, granica propusnog opsega je F_{pass} = 200Hz, granica nepropusnog opsega je F_{stop} = 400Hz, učestanost odabiranja je F_s = 1200Hz, maksimalno slabljenje u propusnom opsegu je A_{pass} = 1dB, minimalno slabljenje u nepropusnom opsegu je A_{stop} = 70dB

1. Za zadatu filtersku specifikaciju odrediti FIR filtar najmanjeg reda (koristiti prozorsku funkciju Hamming, Hann, Blackman, Kaiser)

   Ans: Korišćenjem funkcije kaiserord, za određivanje minimalnog reda i parametra beta tako da budu zadovoljeni svi parametri, dobijemo uređenu dvojku M, beta = 27, 6.75526. Prilikom plotovanja karakteristike vidimo da predloženi filter ne zadovoljava karakteristike Fig: 33. Malim podešavanjem koeficijenata u python fdatoolu dolazimo do specifikacije koja zadovoljava sve uslove Fig: 34. Posmatranjem ostalih karakteristika, za prozorsku funkciju Hann Fig: 35, odnosno, za prozorsku funkciju Hamming Fig: 39, vidimo da ne možemo postići traženu specifikaciju. Koristeći prozorsku funkciju Blackman možemo postići željenu karakteristiku Fig: 38, međutim ovo rešenje koristi filtar većeg reda. Ukoliko postavimo red na veličinu 28, dobijamo neodgovarajuću karakteristiku Fig: 40.

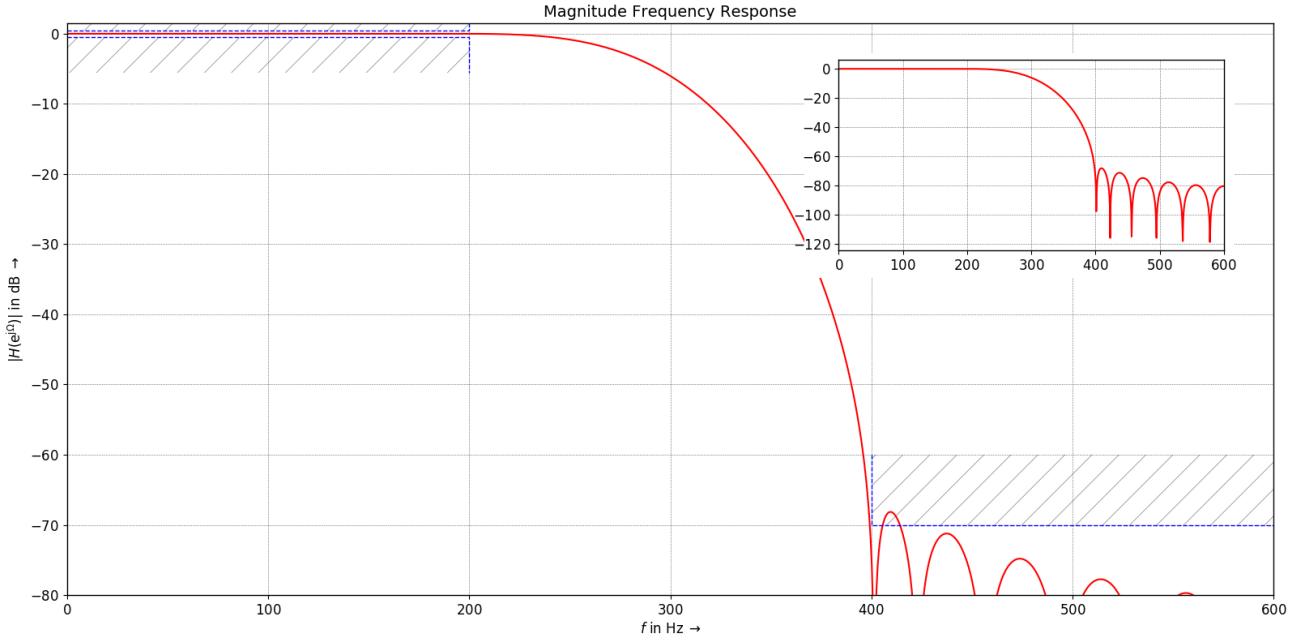

Fig. 33: Frekvencijski odziv filtra, Kaiser, M = 27, beta = 6.75526

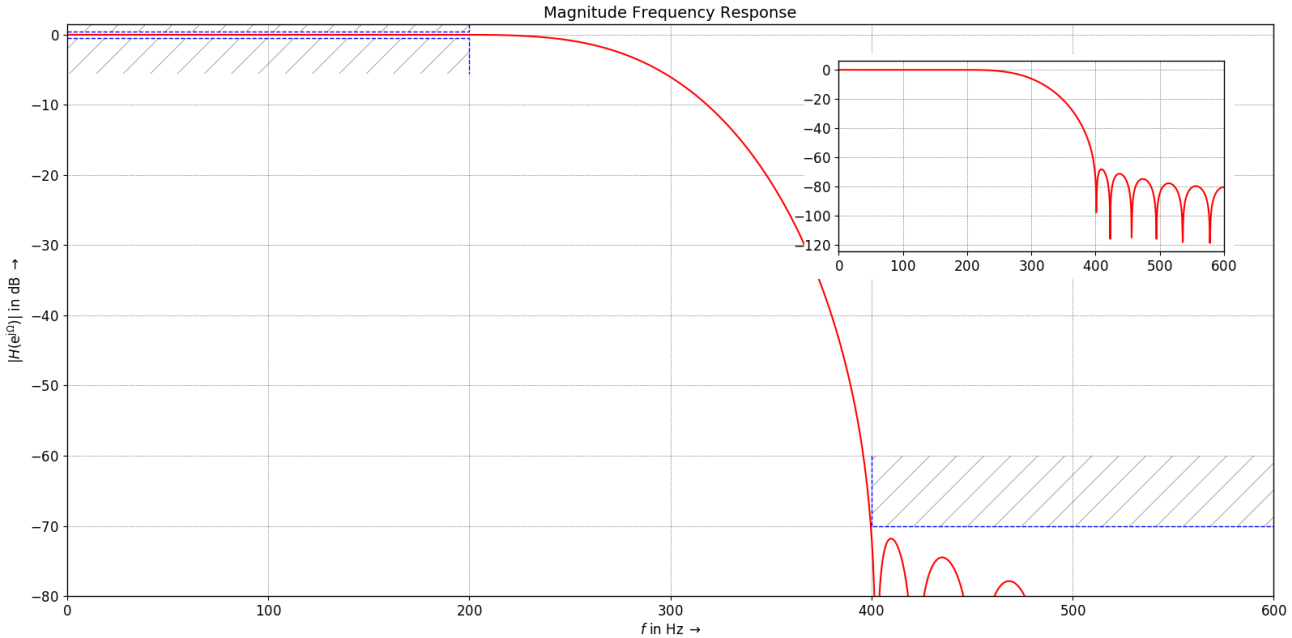

Fig. 34: Frekvencijski odziv filtra, Kaiser, M = 28, beta = 7.1

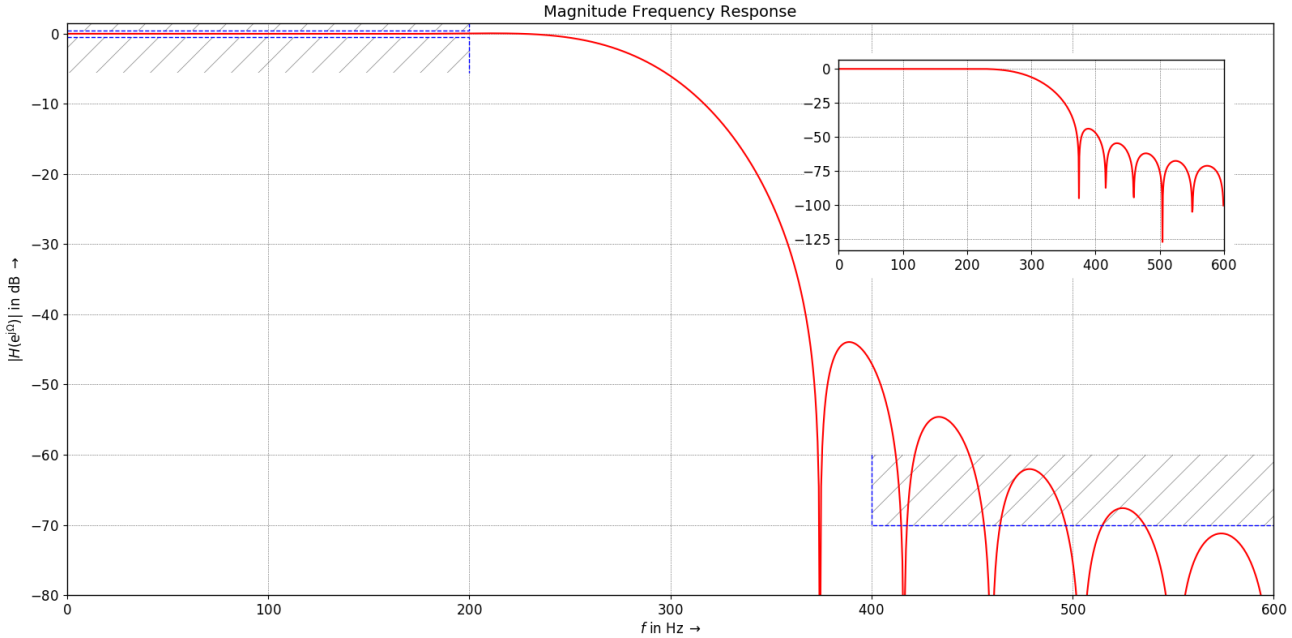

Fig. 35: Frekvencijski odziv filtra, Hann, M = 28

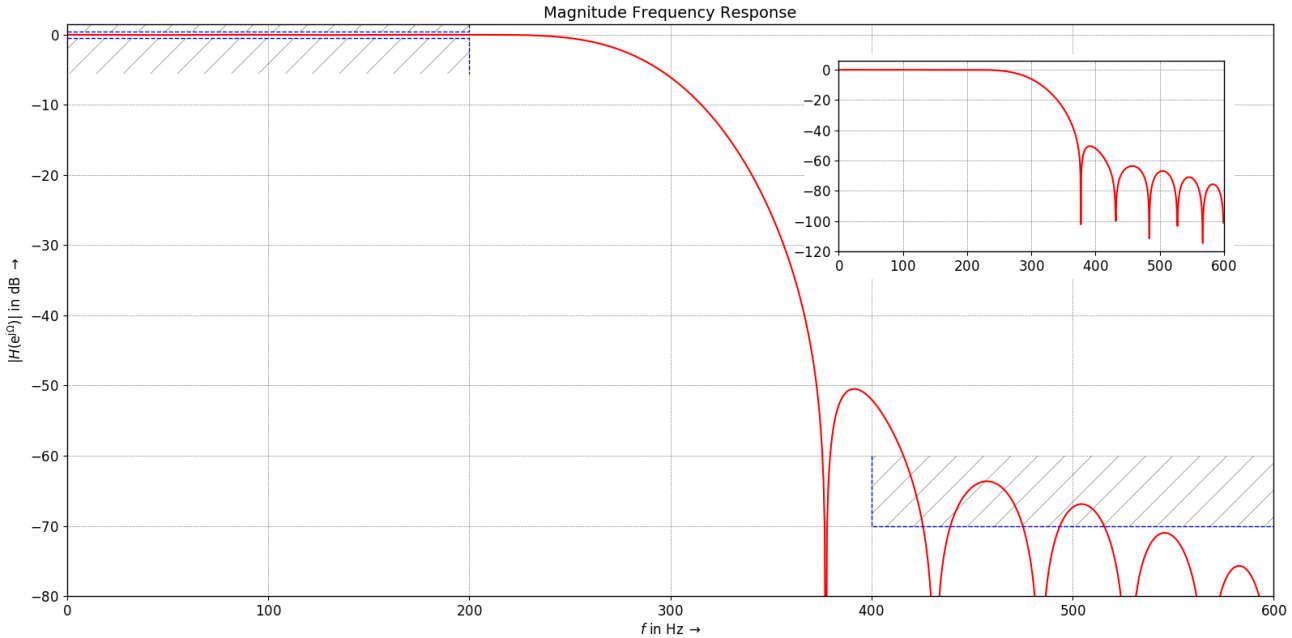

Fig. 36: Frekvencijski odziv filtra, Hamming, M = 28

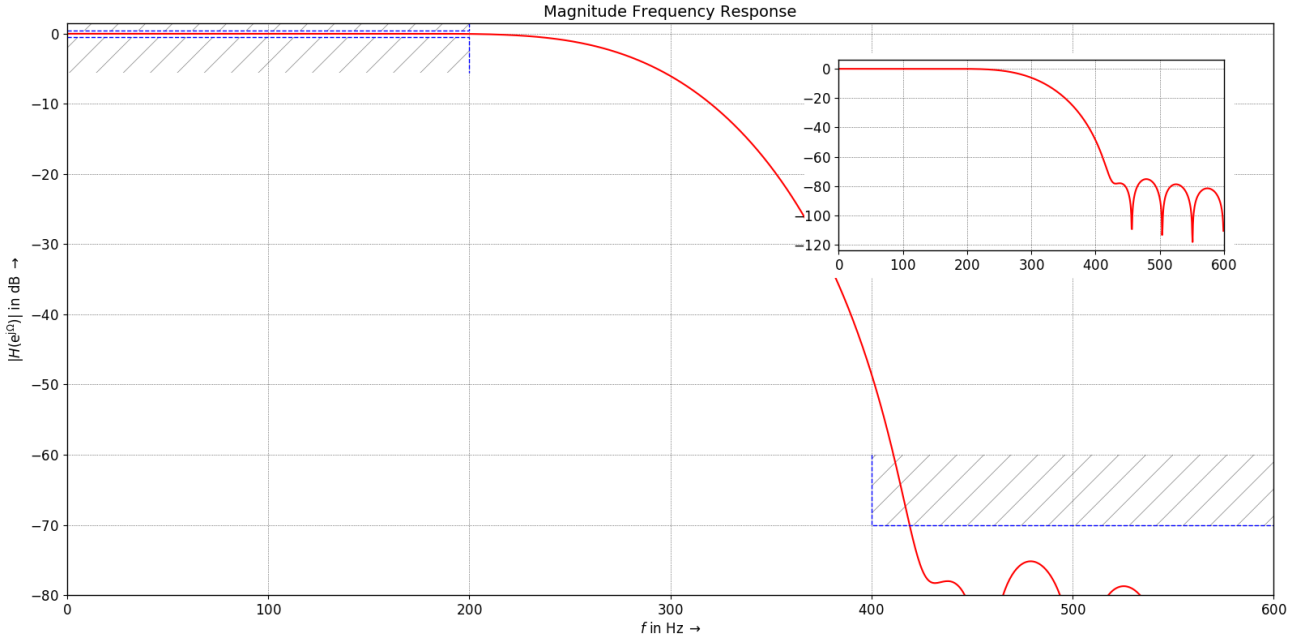

Fig. 37: Frekvencijski odziv filtra, Blackman, M = 28

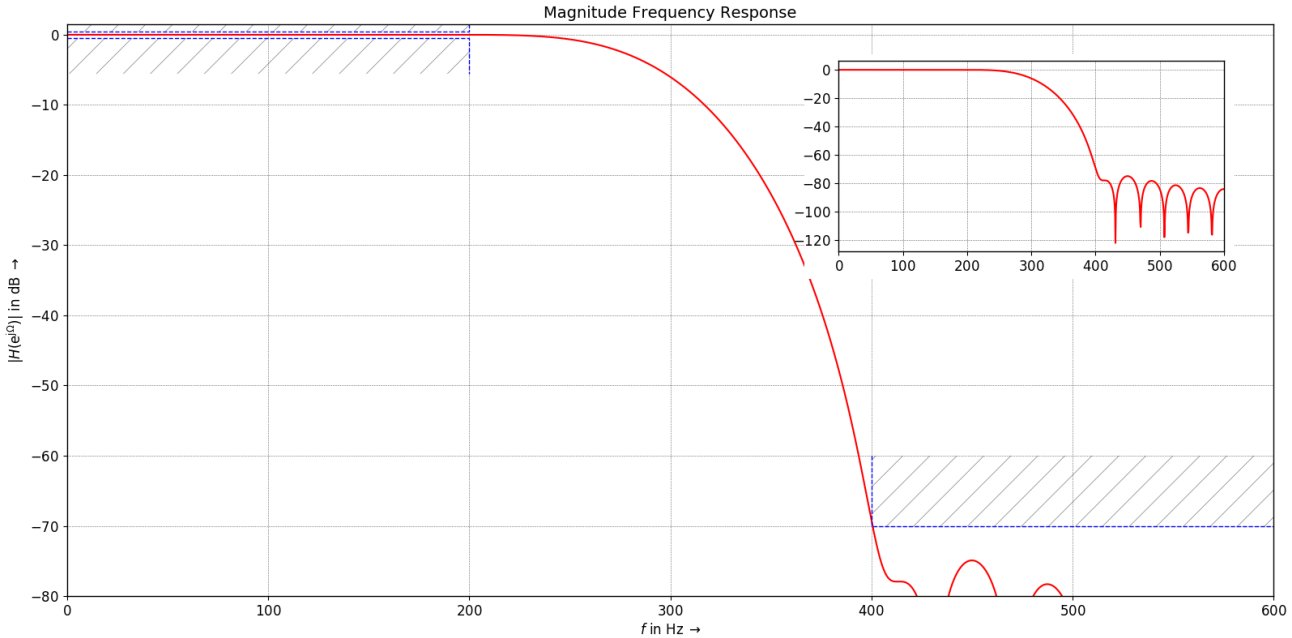

Fig. 38: Frekvencijski odziv filtra, Blackman, M = 33

2. Napisati kod za prikazivanje karakteristika slabljenja, faze, grupnog kasnjenja, položaja polova i nula projektovanog filtra.

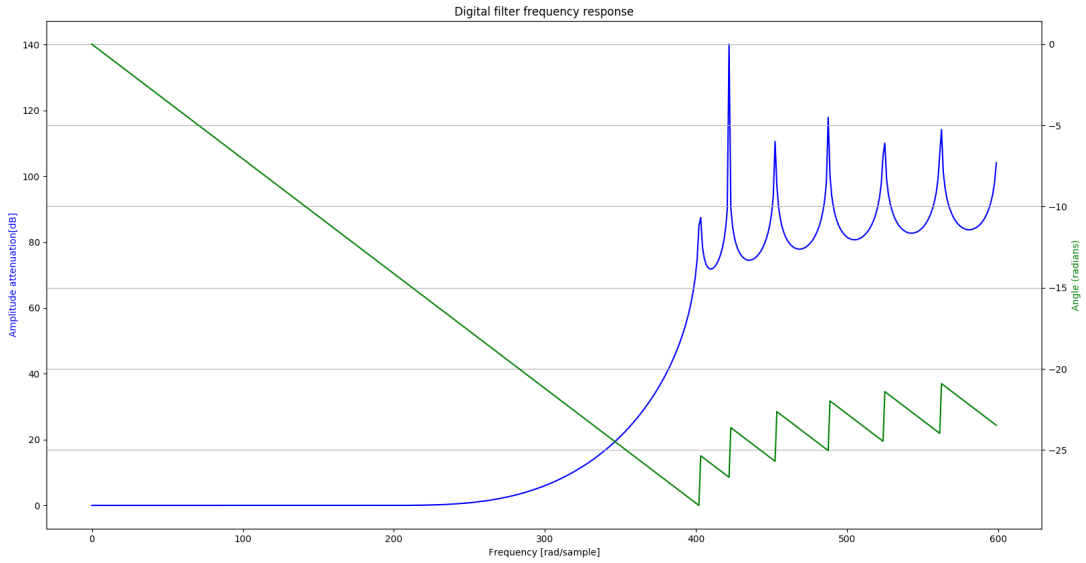

Fig. 39: Karakteristike slabljenja i faze

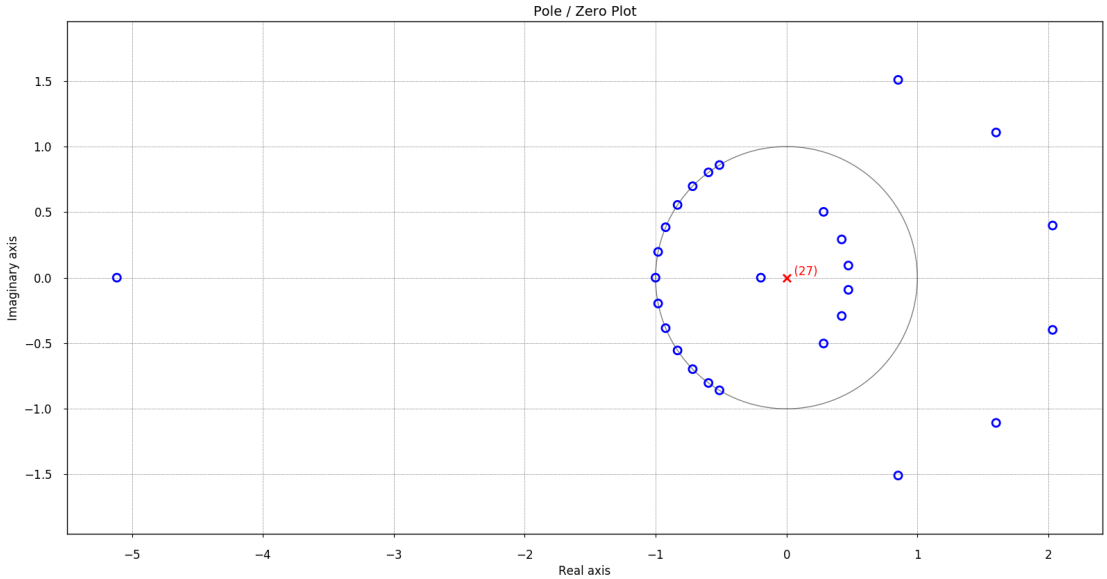

Fig. 40: Položaj nula i polova filtra

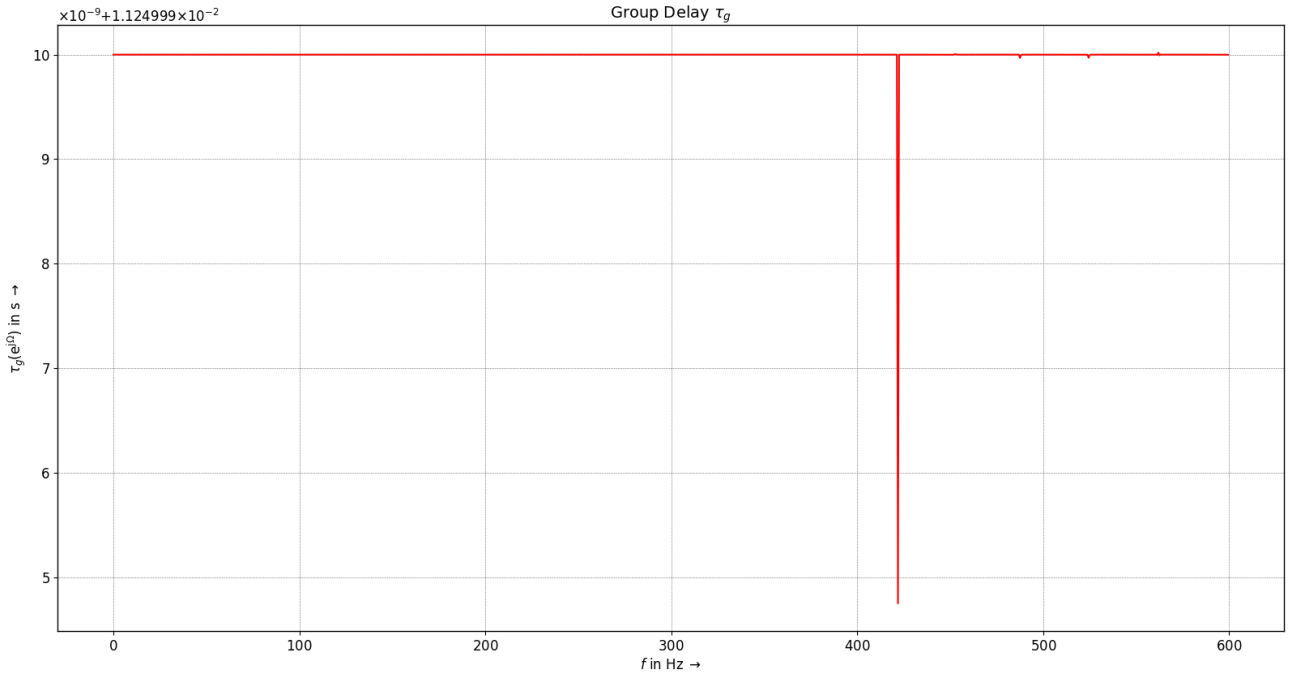

Fig. 41: Karakteristika grupnog kašnjenje filtra

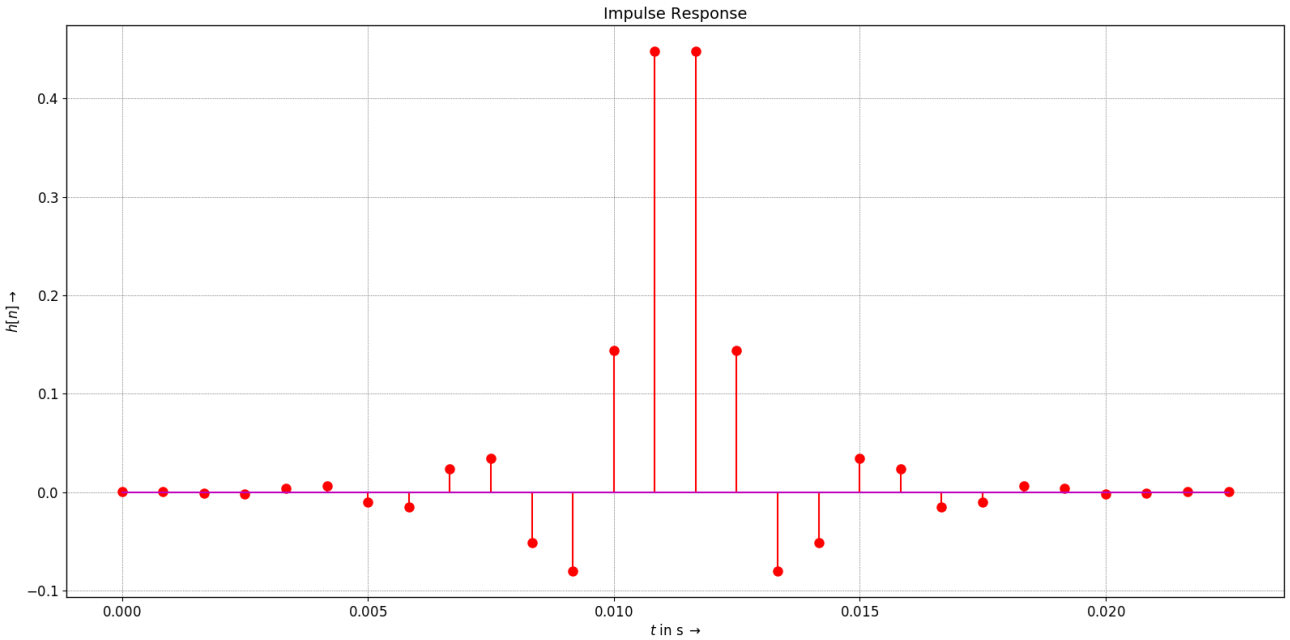

Fig. 42: Impulsni odziv

3. Generisati signal koji se sastoji iz prostoperiodničnih komponenti čije su učestanosti jednake graničnim frekvencijama specifikacije filtra, filtrirati taj signal i korišćenjem DFT pokazati da filter ispunjava postavljene zahteve.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import kaiserord, firwin, freqz, lfilter, tf2zpk, group_delay
from scipy.fftpack import fft
from zplane import zplane

Fs = 1200
N = 512

T = 1.0 / Fs
t = np.linspace(T, N * T, N)

x = 2 * np.sin(2 * np.pi * 200 * t) + 2.5 * np.sin(2 * np.pi * 400 * t)

Fpass = 200.0
Fstop = 400.0
Apass = 1.0
Astop = 70.0
flag = 'scale'


nyq_rate = Fs / 2.0
M, beta = kaiserord(Astop, (Fstop - Fpass) / nyq_rate)
M, beta = 28, 7.1

cutoff_hz = (Fpass + Fstop) / 2
taps = firwin(M, cutoff_hz/nyq_rate, window=('kaiser', beta))
w, h = freqz(taps)

# Amplitude and Phase response
fig = plt.figure()
plt.title('Digital filter frequency response')

# Aplitude
ax1 = fig.add_subplot(111)
plt.plot((w/np.pi/2) * Fs, -20 * np.log10(abs(h)), 'b')
plt.ylabel('Amplitude attenuation[dB]', color='b')
plt.xlabel('Frequency [rad/sample]')
```

```
# Phase
ax2 = ax1.twinx()
angles = np.unwrap(np.angle(h))
plt.plot((w/np.pi/2) * Fs, angles, 'g')
plt.ylabel('Angle (radians)', color='g')
plt.grid()
plt.axis('tight'), plt.show()

# Poles and zeros
z, p, k = tf2zpk(taps, 1.0)

plt.figure()
zplane(z, p)
plt.grid(True, color='0.9', linestyle='-', which='both', axis='both')
plt.title('Poles and zeros')

# Group delay
w1, gd = group_delay((taps, 1.0))
plt.figure()
plt.title('Digital filter group delay')
plt.plot((w/np.pi/2) * Fs, gd)
plt.ylabel('Group delay [samples]'), plt.xlabel('Frequency [Hz]')
plt.grid(True, color='0.9', linestyle='-', which='both', axis='both')
plt.show()

# Filter Signal
filtered_x = lfilter(taps, 1.0, x)

# Display original and filtered signal
plt.figure()
plt.plot(t, x, 'g', label='UnFiltered')
plt.plot(t, filtered_x, 'r', label='Filtered')
plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
plt.legend(loc='upper right')
plt.grid(), plt.show()

X = fft(x)  # Calculate FFT
filtered_X = fft(filtered_x)

y = np.linspace(0.0, 1.0 / (2.0 * T), N // 2)

# Display FFT
plt.figure()
unfiltered = plt.stem(y, 2.0 / N * np.abs(X[0:N//2]),
                      'g', markerfmt='go', label='UnFiltered')
filtered = plt.stem(y, 2.0 / N * np.abs(filtered_X[0:N//2]),
                    'r', markerfmt='ro', label='Filtered')
plt.legend(handles=[filtered, unfiltered], prop={'size': 16})
plt.title('FFT Spectrum')
plt.xlabel('Frequency [Hz]')
plt.grid(), plt.show()
```

Listing 11: Kod za prikazivanje karakteristika, primenjivanje i testiranje filtra

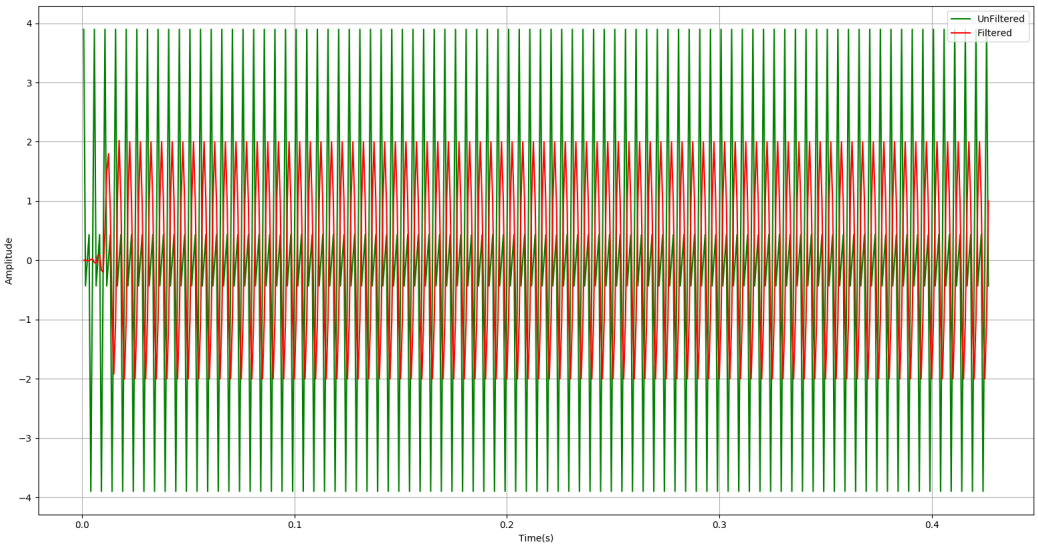

Fig. 43: Prikaz izgenerisanog signala

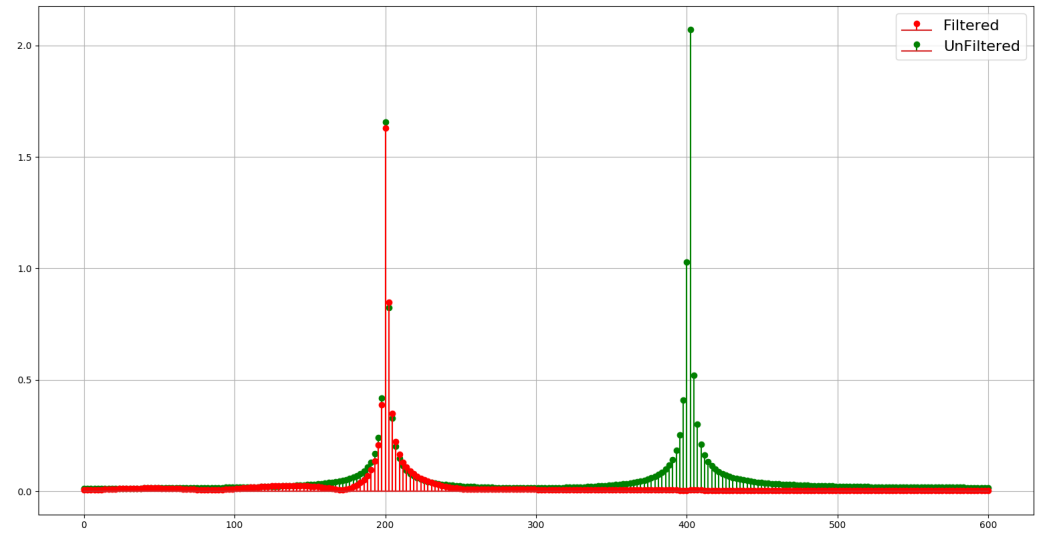

Fig. 44: Amplitudski spektar

Zadatak IV

Zadat je signal "signal.wav" koji je amplitudski modulisan koji osim korisnog signala sadrži i uskopojasni šum. Potrebno je isprojektovati filtar koji potiskuje šum, a zatim demodulisati signal. Napisati kod koji bi trebalo da sadži deo koji omogućuje slušanje obrađenog signala

1. Izbor filtra

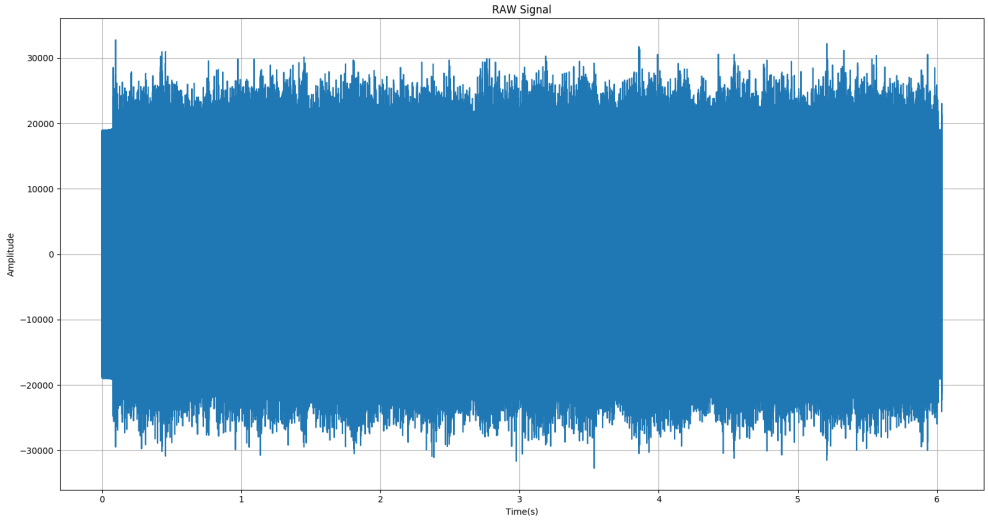

Fig. 45: Prikaz učitanog signala

Ans: Potrebno je uraditi FFT učitanog signala da bi smo detektovali šum. Sa grafika Fig: 46 vidimo sukopojasni šum na frekvenciji 20000 Hz. Dakle potrebno je isprojektovati filtar nepropusnik signala. U ovom rešenju predlžena su dva filtra, jedan malo savršeniji notch filtar Fig: 47, drugi stopband butter filter Fig: 48. U nastavku možete videti dobijene rezultate primenom navedenih filtera.

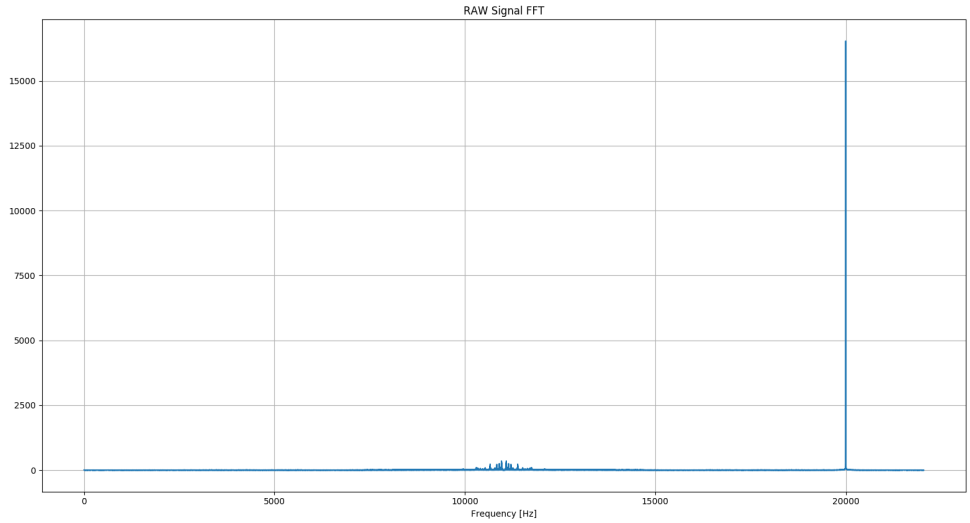

Fig. 46: Amplitudni spektar učitanog signala

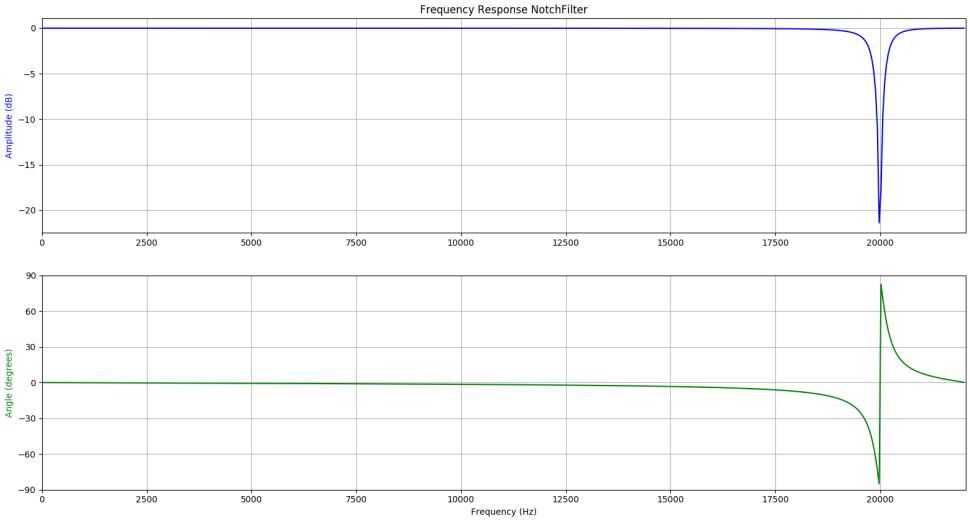

Fig. 47: Predlog Notch filtra

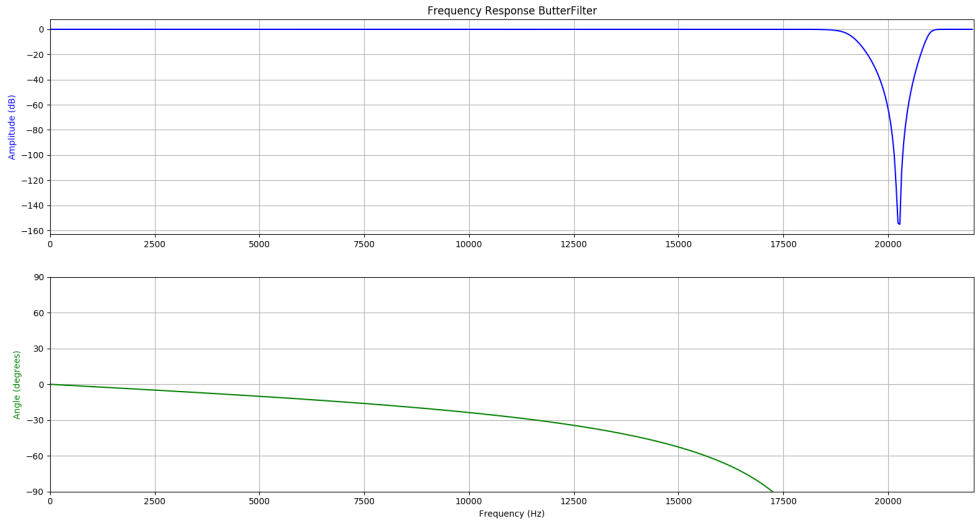

Fig. 48: Predlog Butter StopBand filtra

2. Otklanjanje šuma Notch

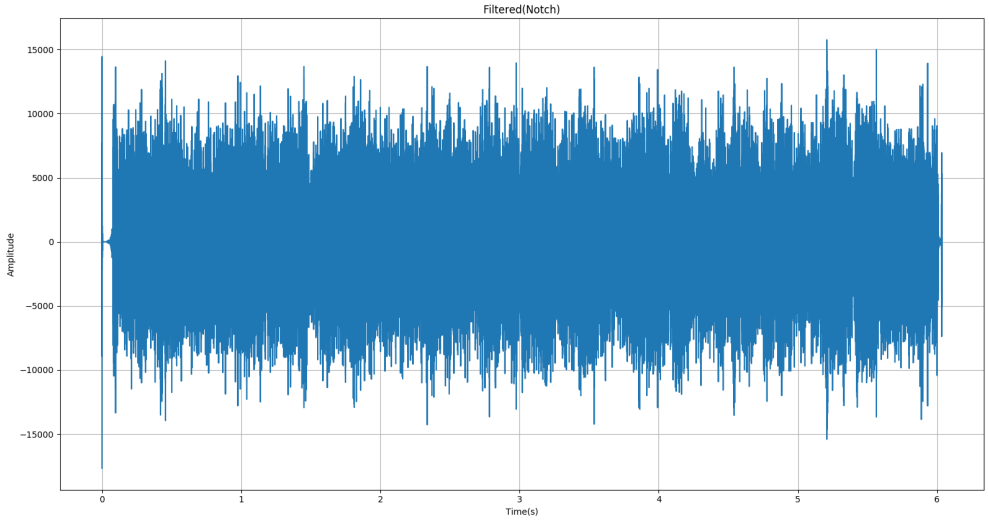

Fig. 49: Grafik filtriranog signala

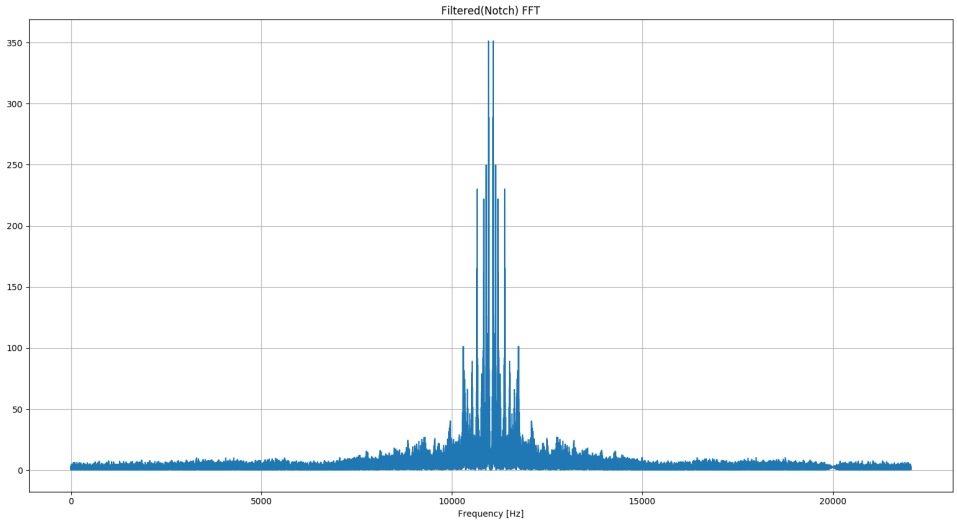

Fig. 50: Amplitudski spektar filtriranog signala

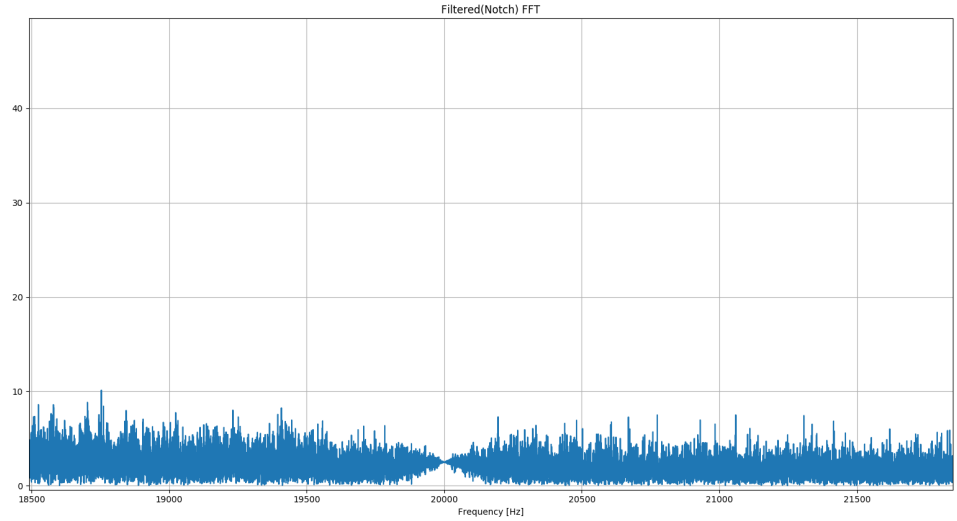

Fig. 51: Amplitudski spetar filtriranog signala u oblasti delovanja filtra

3. Otklanjanje šuma Notch

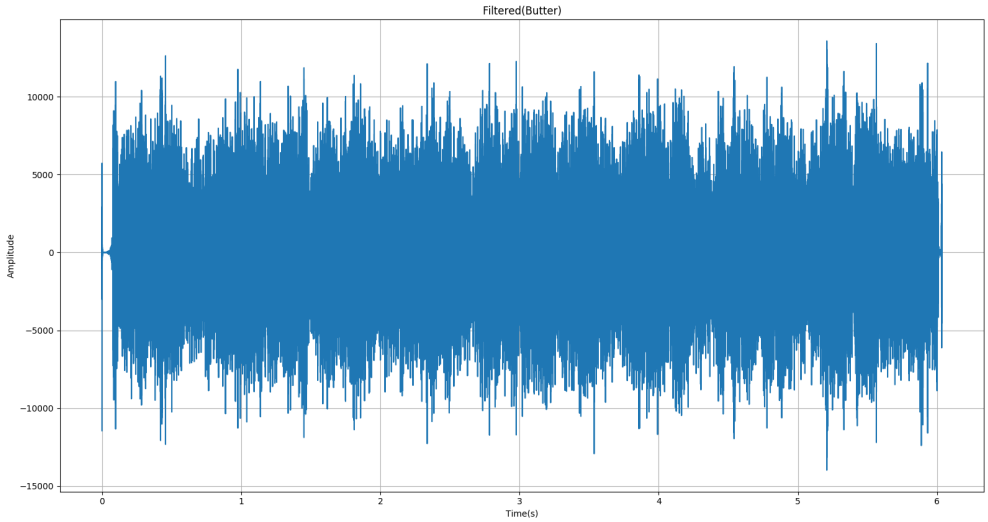

Fig. 52: Grafik filtriranog signala

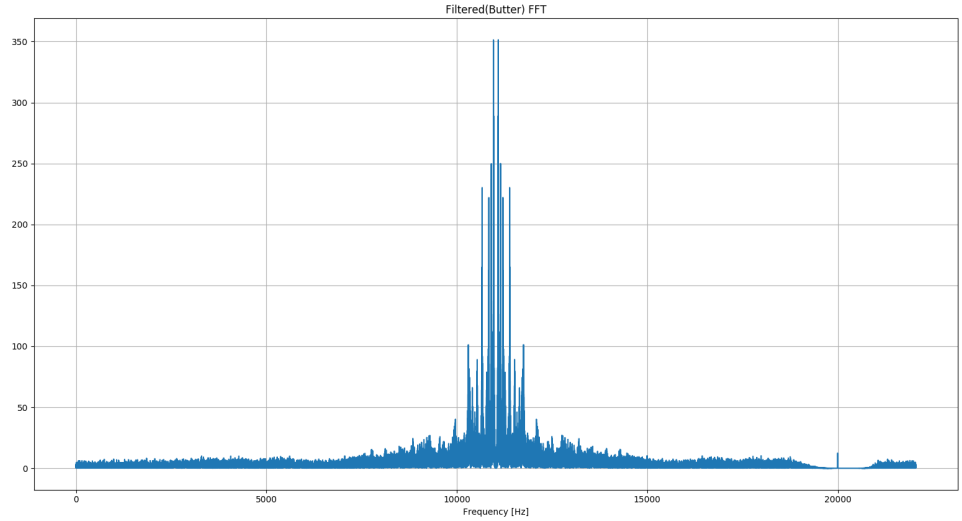

Fig. 53: Amplitudski spektar filtriranog signala

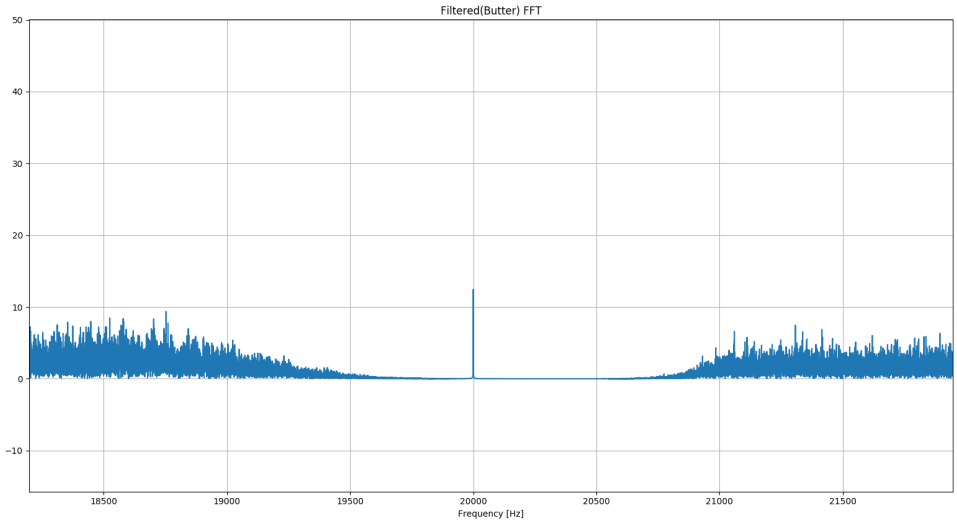

Fig. 54: Amplitudski spetar filtriranog signala u oblasti delovanja filtra

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft
from scipy.io import wavfile

class Signal(object):
    def __init__(self, x):
        self.Fs, self.x = x
        self.T, self.N = 1.0 / self.Fs, len(self.x)

    @staticmethod
    def read(filename):
        return wavfile.read(filename)

    def store(self, filename):
        wavfile.write(filename, self.Fs, self.x)

    def _display(self, y, x, title):
        plt.figure()
        plt.title(title)
        plt.plot(y, x)
        plt.grid()
        plt.show()

    def display(self, title=''):
        t = np.linspace(self.T, self.N * self.T, self.N)
        self._display(
            y=t,
            x=self.x,
            title=title
        )

    def display_fft(self, title=''):
        fftX = fft(self.x)
        yf = np.linspace(0.0, 1.0 / (2.0 * self.T), self.N // 2)
        self._display(
            y=yf,
            x=2.0 / self.N * np.abs(fftX[0:self.N // 2]),
            title=title + " FFT"
        )
```

Listing 12: Klasa za reprezentovanje signala ssignal.py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import iirnotch, freqz, butter, lfilter

class BandStopFilter(object):
    def __init__(self, fs, f0):
        self.Fs, self.F0 = fs, f0
        self.a = self.b = None

    def filter(self, signal):
        return signal.Fs, lfilter(self.b, self.a, signal.x)

    def display(self):
        # Frequency response
        w, h = freqz(self.b, self.a)
        # Generate frequency axis
        freq = w * self.Fs / (2 * np.pi)
        # Plot
        fig, ax = plt.subplots(2, 1, figsize=(8, 6))
        ax[0].plot(freq, 20 * np.log10(abs(h)), color='blue')
        ax[0].set_title("Frequency Response " + type(self).__name__)
        ax[0].set_ylabel("Amplitude (dB)", color='blue')
        ax[0].set_xlim([0, self.Fs // 2])
        ax[0].grid()

        ax[1].plot(freq, np.unwrap(np.angle(h)) * 180 / np.pi, color='green')
        ax[1].set_ylabel("Angle (degrees)", color='green')
        ax[1].set_xlabel("Frequency (Hz)")
        ax[1].set_xlim([0, self.Fs // 2])
        ax[1].set_yticks([-90, -60, -30, 0, 30, 60, 90])
        ax[1].set_ylim([-90, 90])
        ax[1].grid(), plt.show()


class NotchFilter(BandStopFilter):
    def __init__(self, fs, f0, qfactor=50):
        super().__init__(fs, f0)
        w0 = f0 / (fs / 2)
        self.b, self.a = iirnotch(w0, qfactor)

class ButterFilter(BandStopFilter):
    def __init__(self, fs, f0, order=5):
        super().__init__(fs, f0)
        nyq = 0.5 * fs
        low, high = (f0 - 1000) / nyq, (f0 + 1000) / nyq
        self.b, self.a = butter(order, [low, high], btype='stop')
```

Listing 13: Klase za reprezentovanje filtara filter.py

```
from ssignal import Signal
from filter import NotchFilter, ButterFilter
import sounddevice as sd

if __name__ == '__main__':
    signal = Signal(Signal.read('signal.wav'))
    signal.display('RAW Signal')
    signal.display_fft('RAW Signal')

    notch = NotchFilter(fs=signal.Fs, f0=20000.0)
    notch.display()

    butt = ButterFilter(fs=signal.Fs, f0=20000.0)
    butt.display()

    filtered_signal = Signal(notch.filter(signal))
    filtered_signal.display('Filtered(Notch)')
    filtered_signal.display_fft('Filtered(Notch)')

    filtered_signal.store('notch.wav')

    filtered_signal = Signal(butt.filter(signal))
    filtered_signal.display('Filtered(Butter)')
    filtered_signal.display_fft('Filtered(Butter)')

    filtered_signal.store('butter.wav')

    # play sound
    Fs, data = Signal.read('signal.wav')
    sd.play(data, Fs)
```

Listing 14: Klase za pokretanje primera __main__.py