Univerzitet u Beogradu - Elektrotehnički fakultet
Katedra za Signale i sisteme

13E053DOS - Digitalna obrada signala

Domaći zadatak

STUDENT

Mladen Bašić

BROJ INDEKSA

2018/0111

PARAMETRI

P = 3   Q = 3   R = 1   S = 1

Decembar 2020.

Zadatak 1

- *Parametar P = 3*

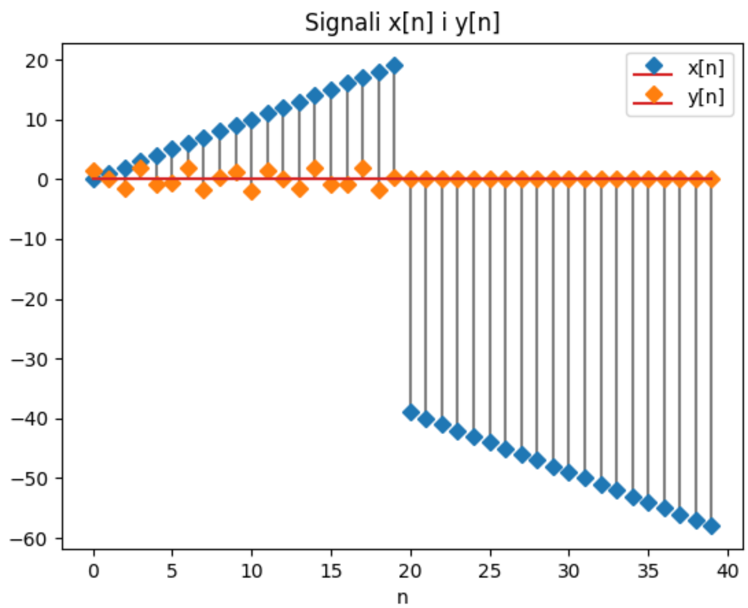
- *Analitički oblik signala x[n] i y[n] i broj tačaka N*

$$x[n] = \begin{cases} n & 0 \leq n \leq 19 \\ n-19 & inace \end{cases}$$

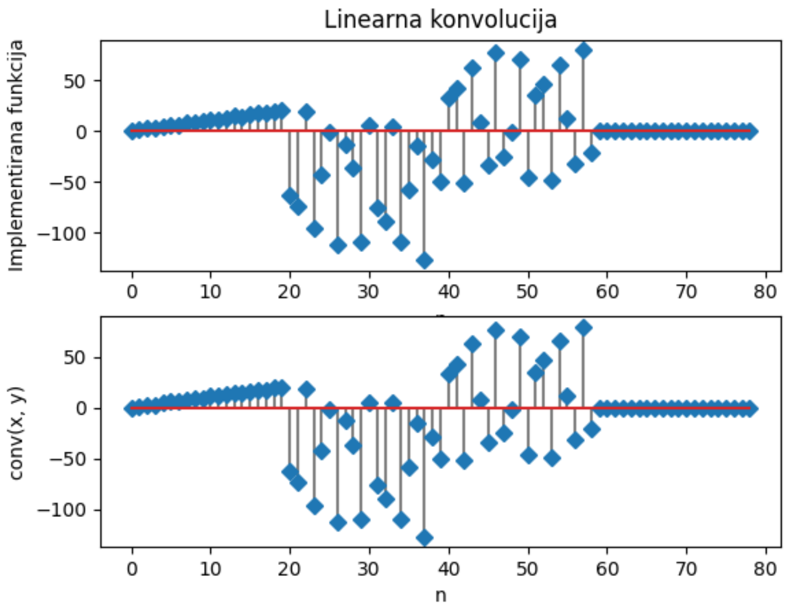
$$y[n] = \begin{cases} 2\cos(4n + \frac{\pi}{4}) & 0 \leq n \leq 19 \\ 0 & inace \end{cases}$$

$$N = 40$$

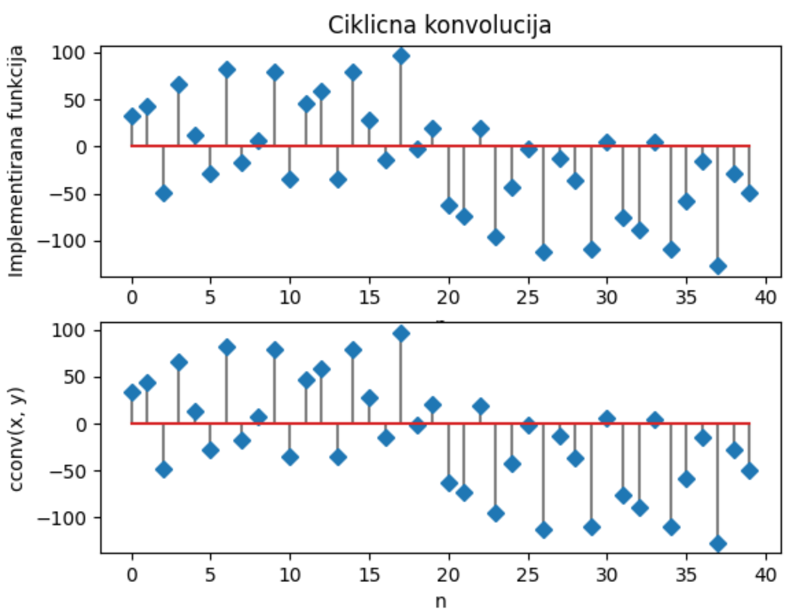
- *Prikaz signala x[n] i y[n]*



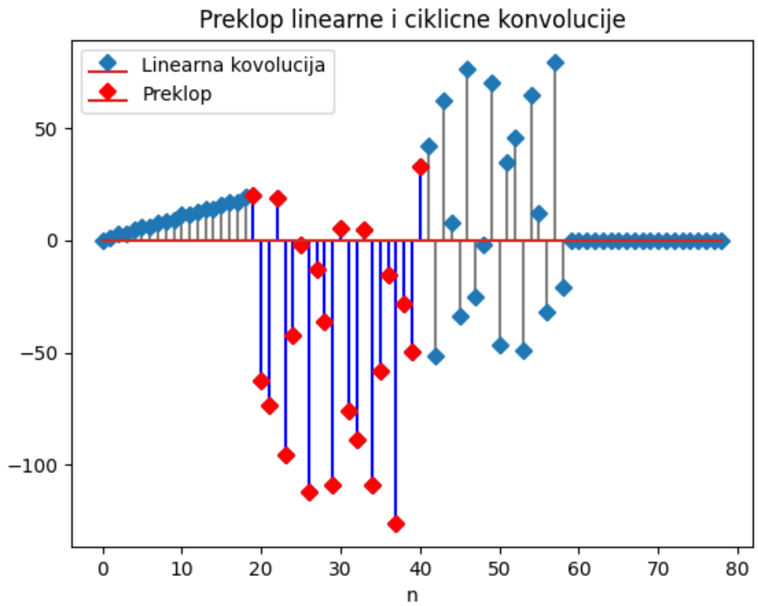
- *Prikaz rezultata linearne konvolucije i rezulata funkcije conv*



- *Prikaz rezultata ciklične konvolucije i rezulata funkcije cconv*



- *Linearna i ciklična konvolucija imaju iste vrednosti u sledećim odbircima*



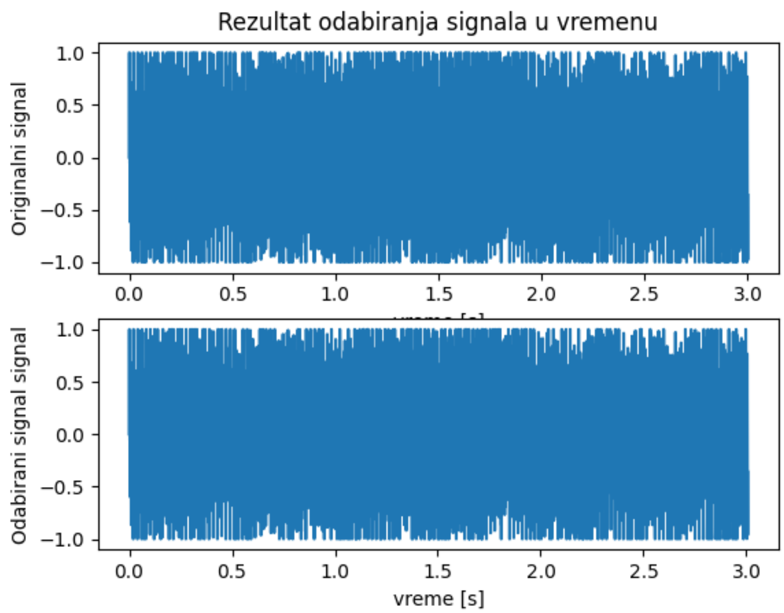
- *Programski kôd*

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

# Postavke zadatka
P = 3
N = 10 * (P + 1)

def lin_conv(x, y):
    """
        Returns linear convolution of two descrete signals
    """

    N = len(x)
    x = np.concatenate((x, np.zeros(N)))
    y = np.concatenate((y, np.zeros(N)))

    lin = np.zeros(2 * N - 1)
    for n in range(2 * N - 1):
        lin[n] = sum([x[k] * y[n - k] for k in range(n + 1)])
    return lin

def circ_conv(x, y):
    """
        Returns circular convolution of two descrete signals
    """

    N = len(x)
    circ = np.zeros(N)
    for n in range(N):
        circ[n] = sum([x[k] * y[(n - k) % N] for k in range(N)])
    return circ

def cconv(x, y):
    """ Imitation of MATLAB's cconv function """
    return np.real(np.fft.ifft( np.fft.fft(x)*np.fft.fft(y) ))

if __name__ == "__main__":

    # Definicija x[n] signala
```

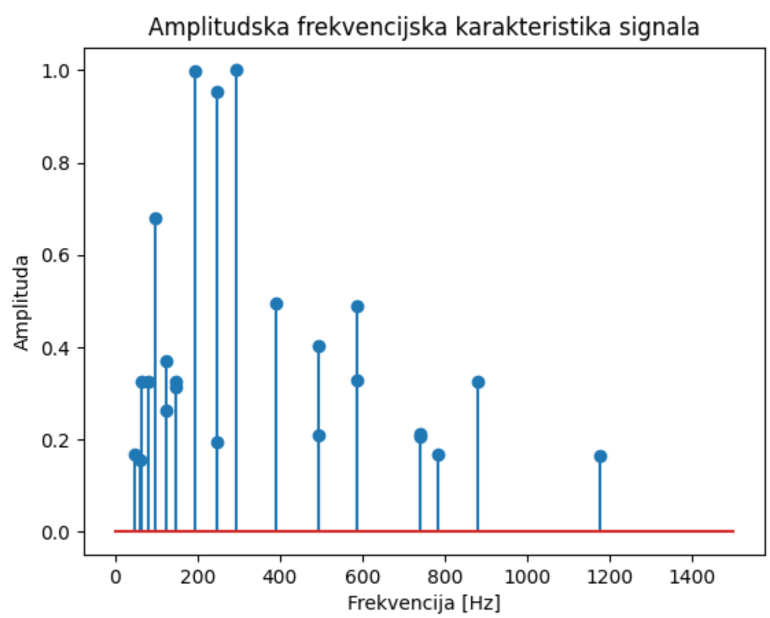
```
    x = np.arange(N)
    x[N//2:] = 1 - x[N//2:] - N/2

    # Definicija y[n] signala
    y = np.arange(N, dtype=float)
    y[:N//2] = 2.0 * np.cos((P + 1)*y[:N//2] + np.pi/4)
    y[N//2:] = 0

    # Plot ulaznih signala
    plt.stem(x, linefmt='gray', markerfmt='D', label='x[n]')
    plt.stem(y, linefmt='gray', markerfmt='D', label='y[n]')
    plt.xlabel('n')
    plt.title('Signali x[n] i y[n]')
    plt.legend()
    plt.savefig('figures/zad1_signali.png')

    # Linearna konvolucija
    lin = lin_conv(x, y)

    # Plot linearne konvolucije
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.title('Linearna konvolucija')
    plt.stem(lin, linefmt='gray', markerfmt='D')
    plt.xlabel('n')
    plt.ylabel('Implementirana funkcija')

    plt.subplot(2, 1, 2)
    plt.stem(np.convolve(x, y), linefmt='gray', markerfmt='D')
    plt.xlabel('n')
    plt.ylabel('conv(x, y)')
    plt.savefig('figures/zad1_linearna_konvolucija.png')

    # Ciklicna konvoluciju
    circ = circ_conv(x, y)

    # Ciklicna konvolucija
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.title('Ciklicna konvolucija')
    plt.stem(circ, linefmt='gray', markerfmt='D')
    plt.xlabel('n')
    plt.ylabel('Implementirana funkcija')

    plt.subplot(2, 1, 2)
    plt.stem(cconv(x, y), linefmt='gray', markerfmt='D')
    plt.xlabel('n')
    plt.ylabel('cconv(x, y)')
    plt.savefig('figures/zad1_ciklicna_konvolucija_provera.png')

    # Overlap dve funkcije
    overlap = lin * np.isin(lin, circ)
    overlap[overlap == 0] = None

    plt.figure()
    plt.title('Preklop linearne i ciklicne konvolucije')
    plt.stem(lin, linefmt='gray', markerfmt='D', label='Linearna kovolucija')
    plt.stem(overlap, linefmt='blue', markerfmt='rD', label='Preklop')
    plt.xlabel('n')
    plt.legend()
    plt.savefig('figures/zad1_overlap.png')
```

Zadatak 2

- *Parametar Q = 3*
- *Prikaz originalnog signala i signala dobijenog odabiranjem sa f_S iz tabele*



- *Amplitudska frekvencijska karakteristika originalnog signala*



- *Signal se sastoji od sledećih tonova*

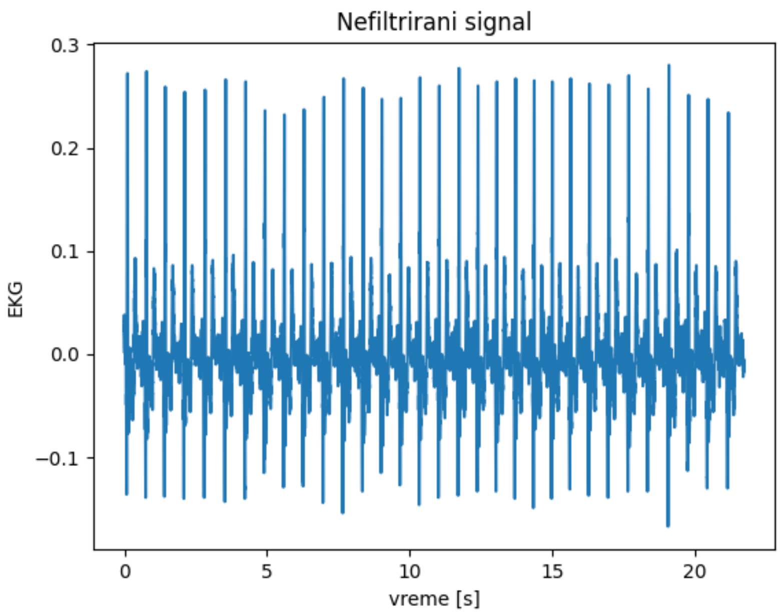
G3, C4, E4

- *Programski kôd*

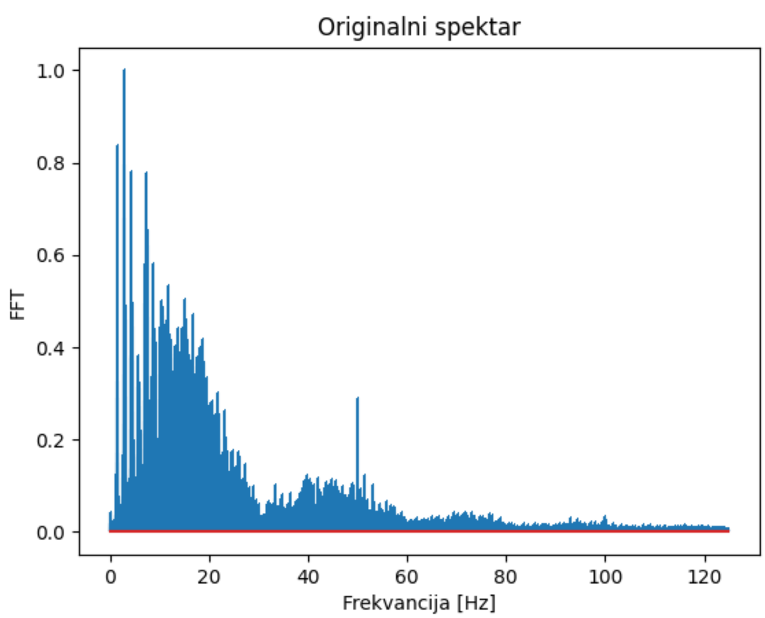
```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.io.wavfile import read

# Postavka zadatka
Q = 3
fs = 4400

if __name__ == "__main__":
    # Ucitavanje originalnog signala
    fs_original, x = read('sounds/audio' + str(Q) + '.wav')
    time_original = np.arange(0, len(x) / fs_original, 1 / fs_original)

    # Odabiranje signala zadatom frekvencijom
    x_sampled = x[::fs_original // fs]
    time_sampled  = np.arange(0, len(x_sampled) / fs, 1 / fs)

    # Prikaz vremenskih oblika odabranog i ne odabranog signala
    plt.subplot(2, 1, 1)
    plt.title('Rezultat odabiranja signala u vremenu')
    plt.xlabel('vreme [s]')
    plt.ylabel('Originalni signal')
    plt.plot(time_original, x / max(x))
    plt.subplot(2, 1, 2)
    plt.xlabel('vreme [s]')
    plt.ylabel('Odabirani signal signal')
    plt.plot(time_sampled, x_sampled / max(x_sampled))
    plt.savefig('figures/zad2_odabiranje.png')

    # Furijeova transformacija originalnog signala
    X = np.fft.fft(x) / len(x)
    X = X[range(len(x) // 2)]
    X = abs(X) / max(abs(X))

    # Niz frekvencija za Furijeovu transformaciju
    freq = np.arange(len(x) // 2) * fs_original / len(x)

    # Poslednji prikaz u bitnom delu spektra
    last_idx = np.where(freq <= 1500)[0][-1]

    # Uklanjanje manje bitnih delova spektra
    # radi lepseg plotovanja
    X[X < 0.15] = None

    # Prikaz spektra signala
    plt.figure()
    plt.title('Amplitudska frekvencijska karakteristika signala')
    plt.xlabel('Frekvencija [Hz]')
    plt.ylabel('Amplituda')
    plt.stem(freq[:last_idx], X[:last_idx])
    plt.savefig('figures/zad2_spektar.png')

    # Trazeni tonovi

    tones = ['D3', 'E3', 'F3', 'G3', 'A3', 'B3', 'C4', 'D4', 'E4', 'F4', 'G4', 'A4', 'B4
', 'C5', 'D5']
    freqs = np.array([146.83, 164.81, 174.61, 196.00, 220.00, 220.00, 246.94, 261.63,
293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25, 587.33])
    main_freq = [freq[idx] for idx, f in enumerate(X) if f > 0.8]

    print([tones[np.argmin(abs(freqs - f))] for f in main_freq])
```

Zadatak 3

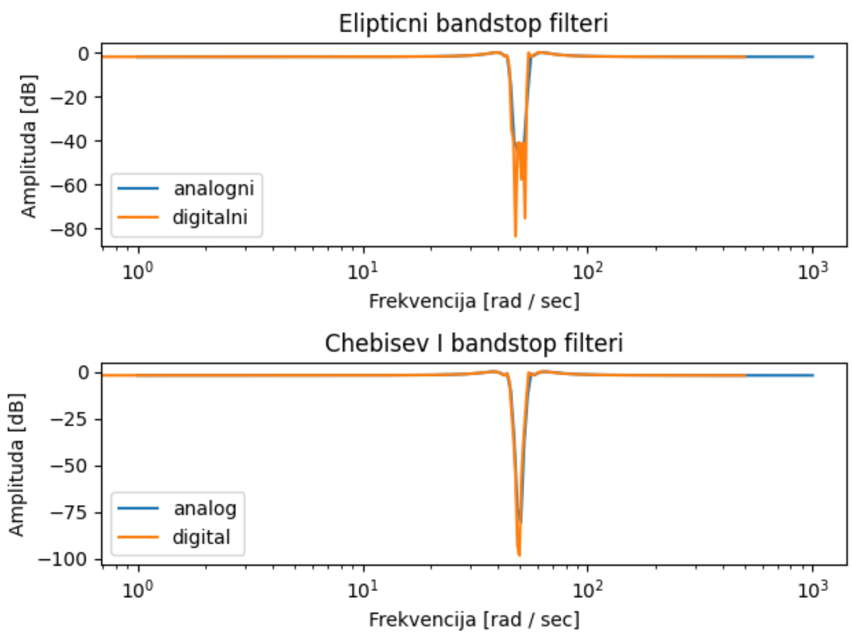
- *Parametar R = 1*
- *EKG signal u vremenskom domenu*



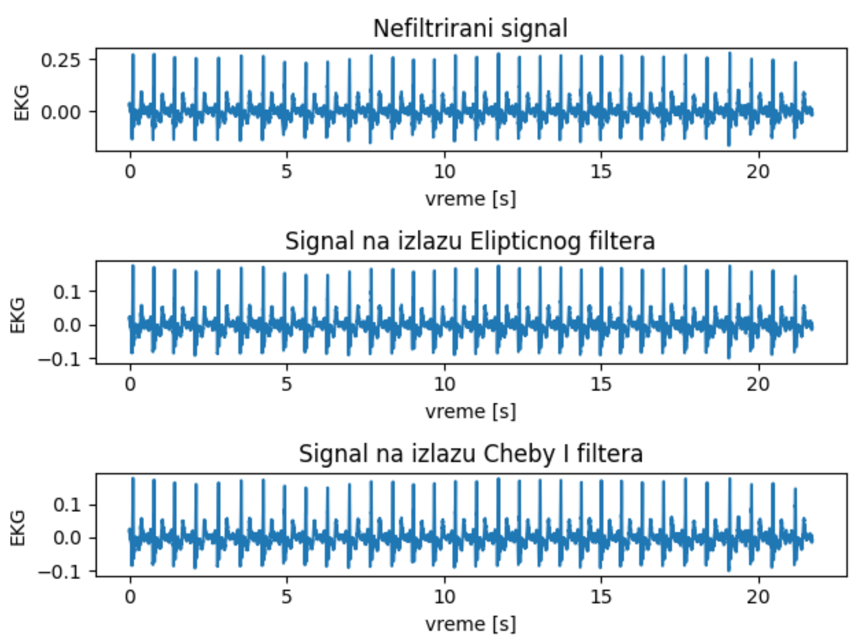
- *Amplitudska frekvencijska karakteristika EKG signala*



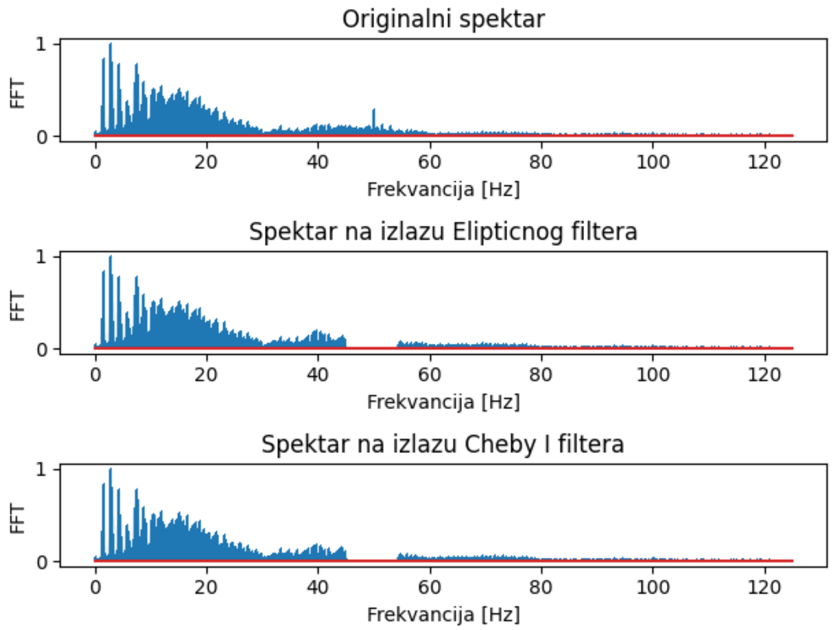
- *Amplitudske frekvencijske karakteristika analognih i digitalnih filtara*



- *Originalni i filtrirani signali u vremenskom domenu*



- *Amplitudske frekvencijske karakteristika originalnog i filtriranih signala*



- *Programski kôd*

```
import numpy as np
from scipy import signal
from scipy.io import loadmat
from matplotlib import pyplot as plt

P = 3
R = 1 # Eliptic i Cheb I


if __name__ == "__main__":

    ## Ucitavanje podataka

    # Vremenski signal
    data = loadmat('input/ekg' + str(R) + '.mat')
    x    = data['x'].flatten()
    fs   = 2 * data['fs']
    t    = (np.arange(len(x)) / fs).flatten()

    # Spektar
    X = np.fft.fft(x)
    X = X[range(len(X) // 8)] / max(X)
    freq = (np.arange(len(X)) * fs / len(x)).flatten()

    ## Filtriranje

    # Elliptic filter
    be, ae = signal.ellip(4, 2, 40, [45, 55], btype='bandstop', analog=True)
    bze, aze = signal.bilinear(be, ae, fs / (2 * np.pi))

    # Filtriranje
    x_ellip = signal.filtfilt(bze, aze, x)

    # Racunanje spektra
    X_ellip = np.fft.fft(x_ellip)
    X_ellip = X_ellip[range(len(X_ellip) // 8)] / max(X_ellip)
    freq = (np.arange(len(X_ellip)) * fs / len(x_ellip)).flatten()

    # Cheby I filter
```

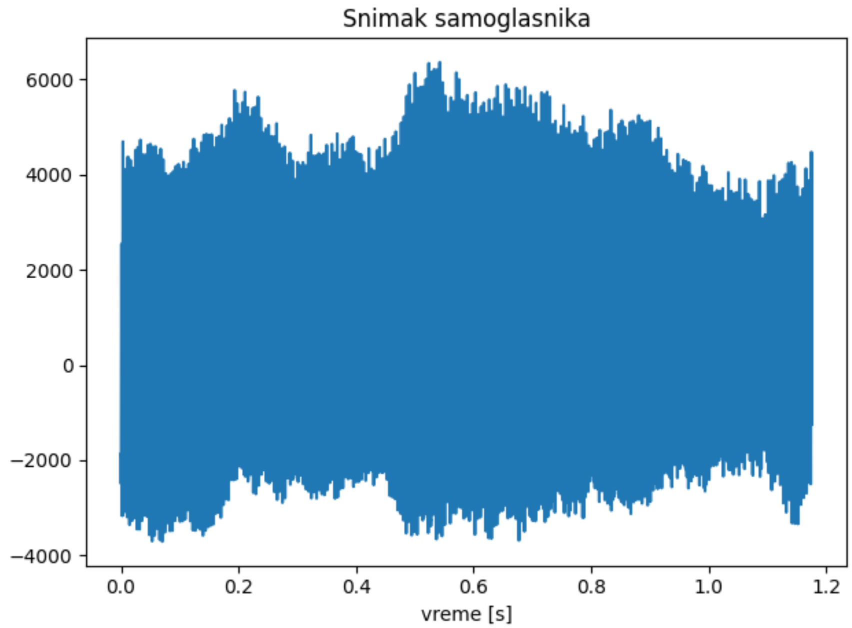
```
    bc, ac = signal.cheby1(4, 2, [45, 55], btype='bandstop', analog=True)
    bzc, azc = signal.bilinear(bc, ac, fs / (2 * np.pi))

    # Filtriranje
    x_cheby1 = signal.filtfilt(bzc, azc, x)

    # Racunanje spektra
    X_cheby1 = np.fft.fft(x_cheby1)
    X_cheby1 = X_cheby1[range(len(X_cheby1) // 8)] / max(X_cheby1)

    ## Racunanje broja otkucaja u minuti

    # Tresholdovanje signala
    trashhold = 0.8 * max(x_cheby1)
    thr = x_cheby1 > trashhold

    # Brojanje pikova
    cnt = sum([1 for idx, _ in enumerate(thr[1:]) if thr[idx - 1] and not thr[idx]])

    # Rezultat
    print(f'Broj otkucaja u minuti je {int(cnt * (60 / t[-1]))}')


    ## Plotovanje razultata

    # a) signal vreme
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Nefiltrirani signal')
    plt.plot(t, x)

    plt.savefig('figures/zad3_signal_vreme.png')

    # b) spektar signal
    plt.figure()
    plt.stem(freq, abs(), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Originalni spektar')

    plt.savefig('figures/zad3_signal_spektar.png')

    # c) AFK filtera
    plt.figure()

    plt.subplot(2, 1, 1)
    w, h = signal.freqs(be, ae)
    wz, hz = signal.freqz(bze, aze)
    plt.semilogx(w, 20 * np.log10(abs(h)), label='analogni')
    plt.semilogx((wz * fs / (2 * np.pi)).T, 20 * np.log10(abs(hz)), label='digitalni')
    plt.legend()
    plt.title('Elipticni bandstop filteri')
    plt.xlabel('Frekvencija [rad / sec]')
    plt.ylabel('Amplituda [dB]')

    plt.subplot(2, 1, 2)
    w, h = signal.freqs(bc, ac)
    wz, hz = signal.freqz(bzc, azc)
    plt.semilogx(w, 20 * np.log10(abs(h)), label='analog')
    plt.semilogx((wz * fs / (2 * np.pi)).T, 20 * np.log10(abs(hz)), label='digital')
    plt.legend()
    plt.title('Chebisev I bandstop filteri')
    plt.xlabel('Frekvencija [rad / sec]')
    plt.ylabel('Amplituda [dB]')

    plt.tight_layout()
    plt.savefig('figures/zad3_filteri.png')
```

```
    # d) svi signali u vremenu
    plt.figure()

    plt.subplot(3, 1, 1)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Nefiltrirani signal')
    plt.plot(t, x)

    plt.subplot(3, 1, 2)
    plt.plot(t, x_ellip)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Signal na izlazu Elipticnog filtera')

    plt.subplot(3, 1, 3)
    plt.plot(t, x_cheby1)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Signal na izlazu Cheby I filtera')

    plt.tight_layout()
    plt.savefig('figures/zad3_filtrirani_vreme.png')

    # e) AFK svih signala
    plt.figure()
    plt.tight_layout()

    plt.subplot(3, 1, 1)
    plt.stem(freq, abs(X), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Originalni spektar')

    plt.subplot(3, 1, 2)
    plt.stem(freq, abs(X_ellip), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Spektar na izlazu Elipticnog filtera')

    plt.subplot(3, 1, 3)
    plt.stem(freq, abs(X_cheby1), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Spektar na izlazu Cheby I filtera')

    plt.tight_layout()
    plt.savefig('figures/zad3_filtrirani_spektar.png')
```

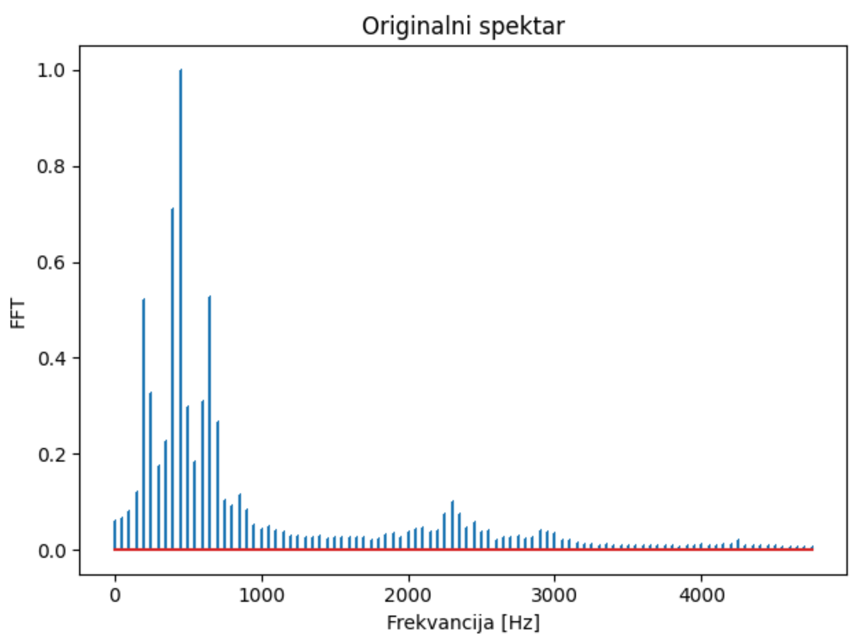
Zadatak 4

- *Parametar S = 1*

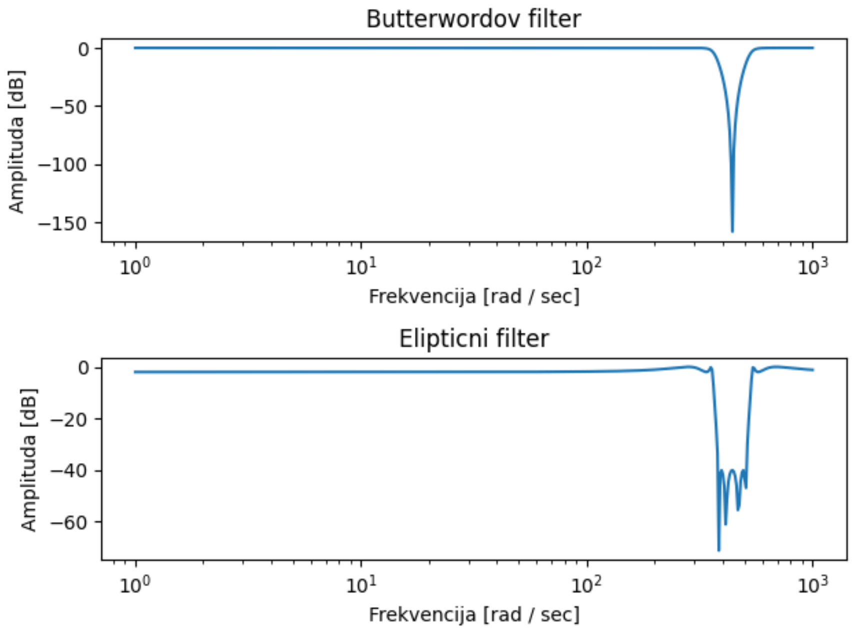
- *Snimak samoglasnika u vremenskom domenu*



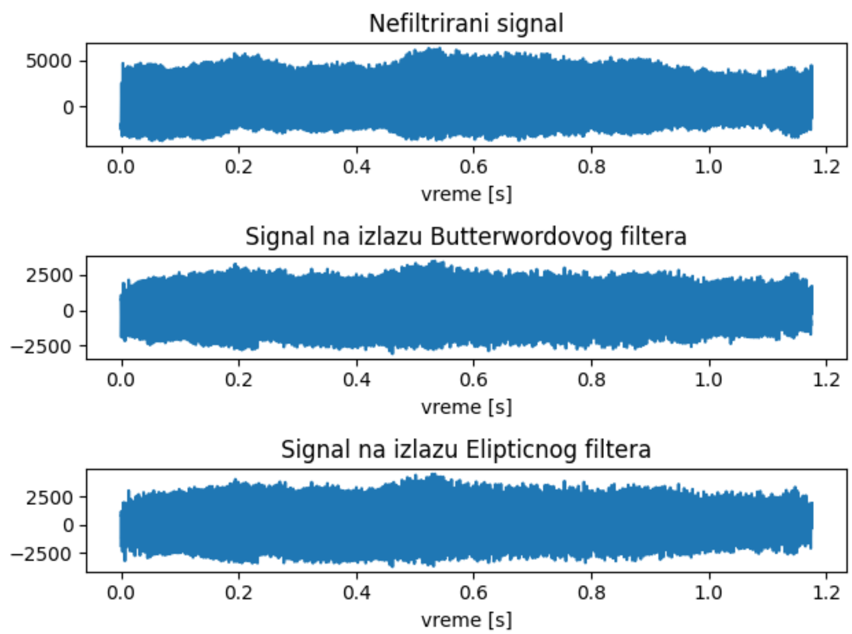
- *Usrednjena amplitudska frekvencijska karakteristika snimljenog signala*



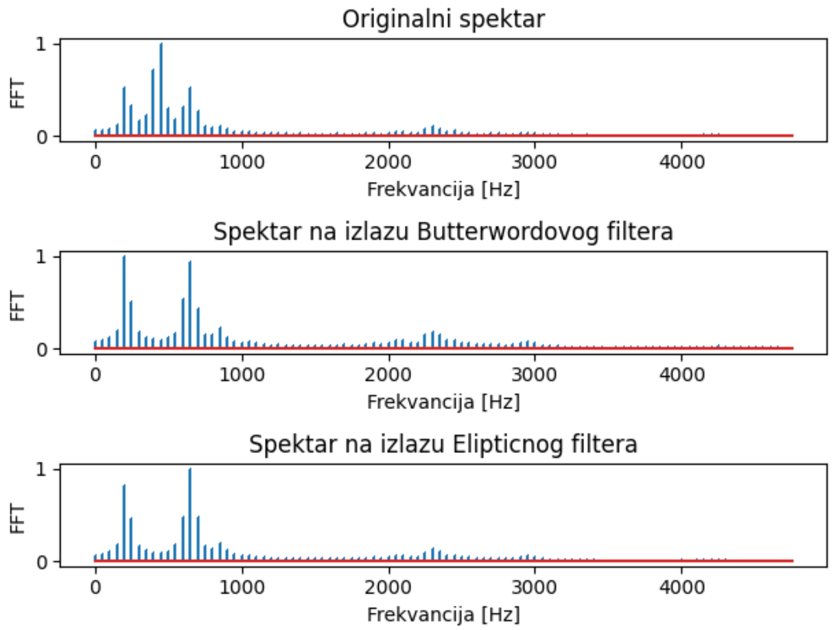
- *Amplitudske frekvencijske karakteristike projektovanih filtara*



- *Originalni i filtrirani signali u vremenskom domenu*



- *Amplitudske frekvencijske karakteristike originalnog i filtriranih signala*



- *Programski kôd*

```python
import numpy as np
from scipy import signal
from scipy.io import loadmat
from matplotlib import pyplot as plt

P = 3
R = 1 # Eliptic i Cheb I


if __name__ == "__main__":

    ## Ucitavanje podataka

    # Vremenski signal
    data = loadmat('input/ekg' + str(R) + '.mat')
    x    = data['x'].flatten()
    fs   = 2 * data['fs']
    t    = (np.arange(len(x)) / fs).flatten()

    # Spektar
    X = np.fft.fft(x)
    X = X[range(len(X) // 8)] / max(X)
    freq = (np.arange(len(X)) * fs / len(x)).flatten()

    ## Filtriranje

    # Elliptic filter
    be, ae = signal.ellip(4, 2, 40, [45, 55], btype='bandstop', analog=True)
    bze, aze = signal.bilinear(be, ae, fs / (2 * np.pi))

    # Filtriranje
    x_ellip = signal.filtfilt(bze, aze, x)

    # Racunanje spektra
    X_ellip = np.fft.fft(x_ellip)
    X_ellip = X_ellip[range(len(X_ellip) // 8)] / max(X_ellip)
    freq = (np.arange(len(X_ellip)) * fs / len(x_ellip)).flatten()

    # Cheby I filter
```

```
    bc, ac = signal.cheby1(4, 2, [45, 55], btype='bandstop', analog=True)
    bzc, azc = signal.bilinear(bc, ac, fs / (2 * np.pi))

    # Filtriranje
    x_cheby1 = signal.filtfilt(bzc, azc, x)

    # Racunanje spektra
    X_cheby1 = np.fft.fft(x_cheby1)
    X_cheby1 = X_cheby1[range(len(X_cheby1) // 8)] / max(X_cheby1)

    ## Racunanje broja otkucaja u minuti

    # Tresholdovanje signala
    trashhold = 0.8 * max(x_cheby1)
    thr = x_cheby1 > trashhold

    # Brojanje pikova
    cnt = sum([1 for idx, _ in enumerate(thr[1:]) if thr[idx - 1] and not thr[idx]])

    # Rezultat
    print(f'Broj otkucaja u minuti je {int(cnt * (60 / t[-1]))}')


    ## Plotovanje razultata

    # a) signal vreme
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Nefiltrirani signal')
    plt.plot(t, x)

    plt.savefig('figures/zad3_signal_vreme.png')

    # b) spektar signal
    plt.figure()
    plt.stem(freq, abs(), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Originalni spektar')

    plt.savefig('figures/zad3_signal_spektar.png')

    # c) AFK filtera
    plt.figure()

    plt.subplot(2, 1, 1)
    w, h = signal.freqs(be, ae)
    wz, hz = signal.freqz(bze, aze)
    plt.semilogx(w, 20 * np.log10(abs(h)), label='analogni')
    plt.semilogx((wz * fs / (2 * np.pi)).T, 20 * np.log10(abs(hz)), label='digitalni')
    plt.legend()
    plt.title('Elipticni bandstop filteri')
    plt.xlabel('Frekvencija [rad / sec]')
    plt.ylabel('Amplituda [dB]')

    plt.subplot(2, 1, 2)
    w, h = signal.freqs(bc, ac)
    wz, hz = signal.freqz(bzc, azc)
    plt.semilogx(w, 20 * np.log10(abs(h)), label='analog')
    plt.semilogx((wz * fs / (2 * np.pi)).T, 20 * np.log10(abs(hz)), label='digital')
    plt.legend()
    plt.title('Chebisev I bandstop filteri')
    plt.xlabel('Frekvencija [rad / sec]')
    plt.ylabel('Amplituda [dB]')

    plt.tight_layout()
    plt.savefig('figures/zad3_filteri.png')

```

```
    # d) svi signali u vremenu
    plt.figure()

    plt.subplot(3, 1, 1)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Nefiltrirani signal')
    plt.plot(t, x)

    plt.subplot(3, 1, 2)
    plt.plot(t, x_ellip)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Signal na izlazu Elipticnog filtera')

    plt.subplot(3, 1, 3)
    plt.plot(t, x_cheby1)
    plt.xlabel('vreme [s]')
    plt.ylabel('EKG')
    plt.title('Signal na izlazu Cheby I filtera')

    plt.tight_layout()
    plt.savefig('figures/zad3_filtrirani_vreme.png')

    # e) AFK svih signala
    plt.figure()
    plt.tight_layout()

    plt.subplot(3, 1, 1)
    plt.stem(freq, abs(X), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Originalni spektar')

    plt.subplot(3, 1, 2)
    plt.stem(freq, abs(X_ellip), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Spektar na izlazu Elipticnog filtera')

    plt.subplot(3, 1, 3)
    plt.stem(freq, abs(X_cheby1), markerfmt=',')
    plt.xlabel('Frekvancija [Hz]')
    plt.ylabel('FFT')
    plt.title('Spektar na izlazu Cheby I filtera')

    plt.tight_layout()
    plt.savefig('figures/zad3_filtrirani_spektar.png')
```

- *Bonus - kôd i rezultati*

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal
from scipy.io.wavfile import read, write


def estimate_pitch(x, fs):
    """ Returns fft of 20ms window """

    step = int(fs * 20 * 1e-3)

    spek = []

    for idx in range(0, step * (len(x) // step), step):
        X = np.fft.fft(x[idx:idx + step])
        X = X[:len(X) // 2] / max(X)
        X = abs(X)
```

```
        if len(spek) == 0:
            spek = X
        else:
            spek += X

    spek /= len(x) / step
    spek /= max(spek)
    freq = (np.arange(len(spek)) * fs / step).flatten()

    lowbound   = np.nonzero(freq < 165)[-1][-1]
    highbound  = np.nonzero(freq < 255)[-1][-1]

    return freq[lowbound + np.argmax(spek[lowbound:highbound+1])]



if __name__ == "__main__":
    # Ucitavanje originalnog signala
    fs, x = read('sounds/in.wav')

    print(f'Pitch frekvencija je {estimate_pitch(x, fs)}')
```

Pitch frekvencija je 200Hz