

Lucrarea 3

Filtre digitale – Aplicații în procesarea audio

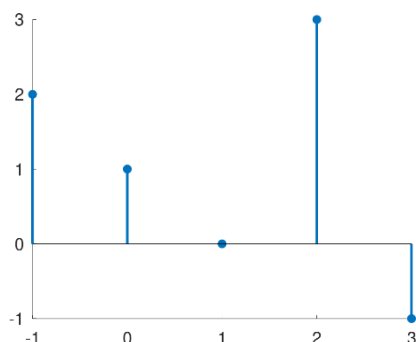
Recapitularea noțiunilor generale despre filtre. Exemplificarea convoluției semnalelor discrete. Familiarizarea cu filtrele digitale de tip FIR, cu modul lor de proiectare și aplicații ale acestora în procesarea de sunet.

1. Noțiuni teoretice

1.1 Transformata Z

Un instrument utilizat des în procesarea digitală a semnalelor este transformata Z. Ea este echivalenta discretă a transformatei Laplace.

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \quad (1)$$



$$X(z) = 2z + 1 + 3z^{-2} - z^{-3}$$

Figura 1. Exemplificare a transformatei Z pentru o secvență discretă

1.2 Convoluția semnalelor

Convoluția reprezintă operația matematică de a combina două semnale pentru a genera un al treilea semnal. Este **operația fundamentală** în domeniul procesării digitale a semnalelor.

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \quad (2)$$

În termeni mai simpli, convoluția celor două semnale prezentate în formula (2) reprezintă oglindirea semnalului $h[n]$ și glisarea acestuia pas cu pas peste eșantioanele semnalului $x[n]$. La fiecare deplasare elementele celor două semnale de pe aceleași poziții sunt înmulțite între ele,

iar apoi produsele rezultate sunt sumate pentru a calcula eșantionul $y[n]$. Procesul continuă până când $h[n]$ a glisat complet peste $x[n]$.

Convoluția a două semnale poate fi calculată în Python prin intermediul funcției *convolve(a,b)* din pachetul *scipy*, sub-pachetul *signal*, unde *a* și *b* reprezintă semnalele care sunt incluse în convoluție.

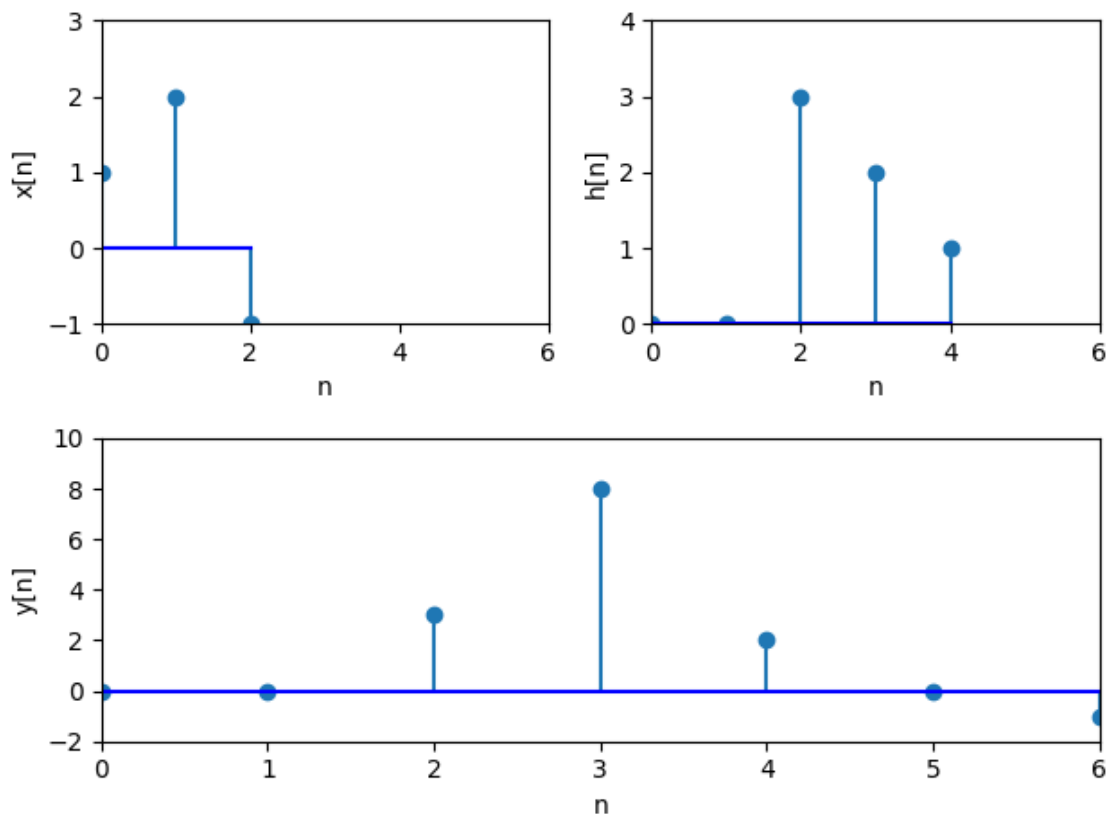


Figura 2. Exemplu de convoluție

Exemplul din Figura 2 arată convoluția a două semnale $x[n]$ și $h[n]$, iar secvența de cod de mai jos a fost folosită pentru a genera această reprezentare.

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

x=np.array([1,2,-1])
h=np.array([0,0,3,2,1])
y=signal.convolve(x,h) #Convolutia semnalelor x si h
n=np.arange(0,7)

#Reprezentare grafica
plt.subplot(2,2,1)
plt.stem(np.arange(0,3),x,baselfmt="b") #schimbare a culorii pentru linia de
baza
plt.xlim([0,6]) #definirea limitelor pentru axa Ox
```

```

plt.ylim([-1,3]) #definirea limitelor pentru axa Oy
plt.xlabel('n')
plt.ylabel('x[n]')

plt.subplot(2,2,2)
plt.stem(np.arange(0,5),h,basefmt="b")
plt.xlim([0,6])
plt.ylim([0,4])
plt.xlabel('n')
plt.ylabel('h[n]')

plt.subplot(2,1,2)
plt.stem(n,y,basefmt="b")
plt.xlim([0,6])
plt.ylim([-2,10])
plt.xlabel('n')
plt.ylabel('y[n]')

plt.tight_layout()
plt.show()


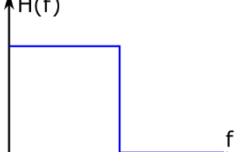
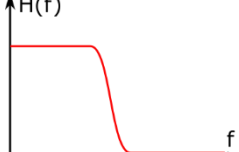

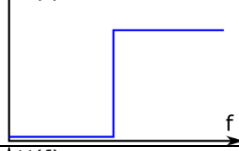
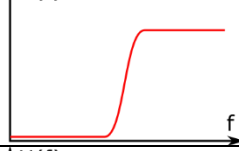
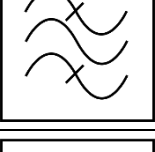
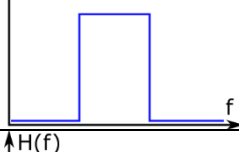
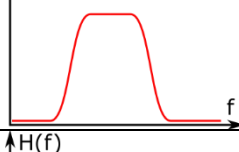
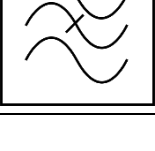
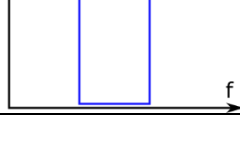

```

1.3 Filtre digitale

Filtrele electronice sunt circuite care prezintă un comportament selectiv în frecvență. Acestea sunt realizate fie cu componente pasive (rezistori, bobine, condensatoare), fie cu componente active (tranzistori, amplificatoare operaționale).

Filtrele introduc, în mod ideal, o atenuare infinită în banda de oprire și o atenuare nulă în banda de oprire. Cele două benzi sunt delimitate de către frecvența de tăiere. Tabelul 2 prezintă cele patru tipuri de filtre ce pot fi realizate, iar Figura 3 ilustrează principalii parametri ai filtrelor.

Tabelul 2. Tipuri de filtre

Filtru	Simbol	Caracteristică ideală	Caracteristică reală
Trece-jos (FTJ)			
Trece-sus (FTS)			
Trece-bandă (FTB)			
Oprește-bandă (FOB)			

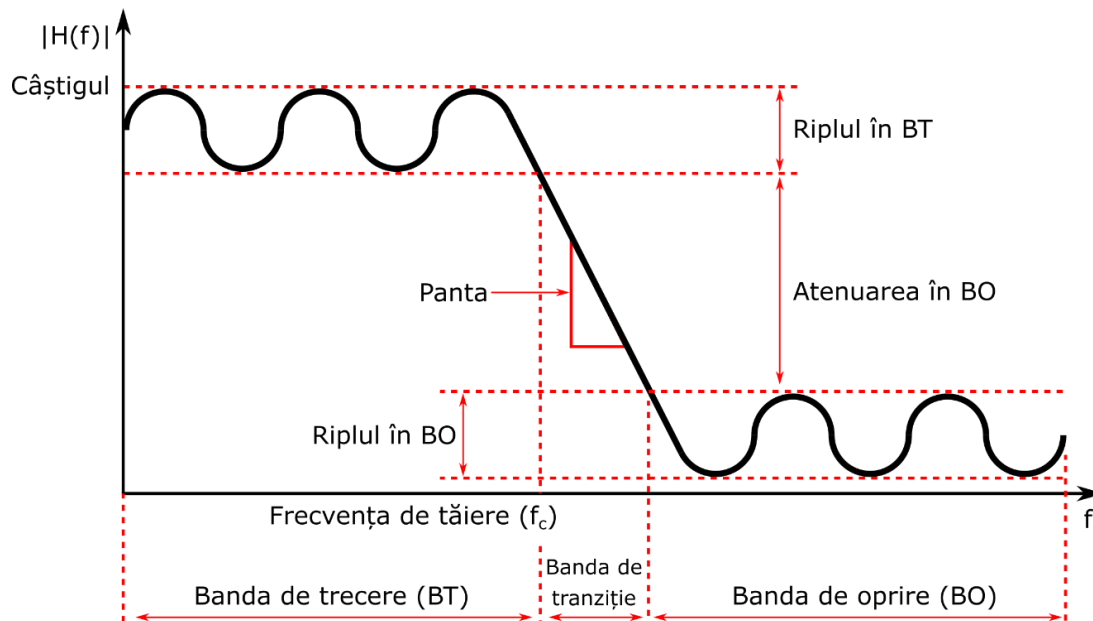


Figura 3. Exemplificarea parametrilor unui FTJ

Filtrele digitale sunt **algoritmi de calcul** aplicați asupra semnalelor discrete. Răspunsul acestora în domeniul timp este dat de relația:

Se pot proiecta algoritmi de calcul care reproduc caracteristicile celor patru tipuri de filtre elementare. Printre avantajele filtrelor digitale se numără:

- Proiectarea unor caracteristici imposibil de realizat cu filtre analogice;
- Performanță acestora nu este afectată de mediu;
- Filtrarea mai multor semnale prin multiplexare;
- Atenuare mare în banda de oprire;
- Performanțe ce pot fi reproduse de la procesor la procesor fără reglare.

Dezavantajele acestor filtre sunt legate de:

- Frecvența de procesare limitată de procesor și de timpul de conversie al convertoarelor analog-numerice și digital-analogice;
- Prelucrarea semnalelor se realizează pe o lățime de bandă mai mică decât cea a filtrelor analogice.

O diferență notabilă față de filtrele analogice o reprezintă modul în care este exprimată frecvența filtrelor digitale. Frecvența în funcție de care se reprezintă caracteristica filtrelor digitale se numește *frecvență normală*. Practic, axa de frecvențe clasică este împărțită la frecvența de eșantionare, rezultând astfel o nouă axă de frecvențe normate cu valori de la 0 la 0.5 (valorile mai mari nu sunt de folos).

Filtrele digitale prezintă o frecvență maximă de lucru, mai precis jumătate din frecvența de eșantionare. Aceasta poartă denumirea de frecvența Nyquist și este definită într-un similar frecvenței de eșantionare din lucrarea trecută.

$$f_{Nyquist} = \frac{f_e}{2} \quad (3)$$

1.4 Filtre cu răspuns finit la impuls (FIR)

Sunt filtre digitale a căror funcție de transfer este de forma:

$$H(z) = \sum_{k=0}^{N-1} a_k z^{-k} \quad (4)$$

unde N reprezintă lungimea filtrului sau numărul de eșantioane necesare calculării unui nou eșantion filtrat.

Filtrarea cu filtre FIR se realizează aplicând asupra eșantionului curent și a celor precedente operații de adunare sau înmulțire. O diagramă logică a modului de funcționare a unui astfel de filtru este ilustrată în Figura 4. Blocurile notate cu Z^{-1} poartă denumirea de blocuri de întârziere și au rolul de a memora un eșantion.

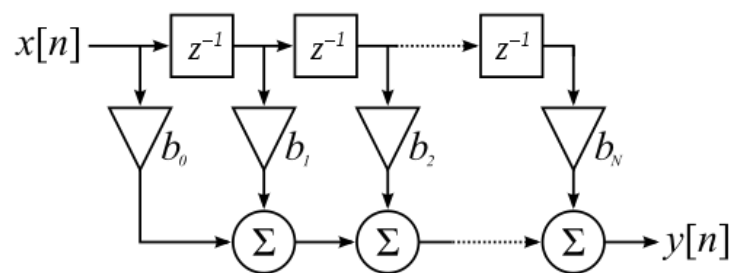


Figura 4. Structura logică a unui filtru FIR

Filtrele de tip FIR pot fi definite în Python prin intermediul funcției *firwin(n,w,pass_zero='t')* din pachetul *scipy*, unde n reprezintă ordinul filtrului, w frecvența de tăiere, iar t tipul filtrului. Pentru un FTJ sau FTS w este un scalar, iar pentru FTB și FTP un vector cu 2 elemente (frecvențele de tăiere). Frecvența ia valori de la 0 la 1, unde 1 reprezintă frecvența Nyquist (jumătate din frecvența de eșantionare). Argumentele pentru tipurile de filtre sunt: "lowpass", "highpass", "bandstop", "bandpass".

Pentru a afișa caracteristicile de amplitudine și fază ale unui filtru digital folosim comanda *freqz()*, tot din pachetul *scipy*.

```

import numpy as np
from scipy import signal
import matplotlib.pyplot as plt

f=signal.firwin(23,0.3,pass_zero='highpass') #FIR FTS cu frecventa de taiere
egala cu 0.3 din Frecventa Nyquist

#Afisare
w,h=signal.freqz(f) #Generare componente amplitudine si faza pentru FIR
anterior
w=w/np.pi

plt.subplot(2,1,1)
plt.plot(w,20*np.log10(abs(h)), 'b')
plt.xlabel('Frecventa normata [x$\pi$ rad/esantion]')
plt.ylabel('Amplitudine [dB]')
plt.xlim([0,1])
plt.grid()

plt.subplot(2,1,2)
angles=np.unwrap(np.angle(h))
plt.plot(w,angles, 'g')
plt.xlabel('Frecventa normata [x$\pi$ rad/esantion]')
plt.ylabel('Faza [grade]')
plt.xlim([0,1])
plt.grid()

plt.tight_layout()
plt.show()

```

Graficele rezultate sunt reproduse în Figura 5

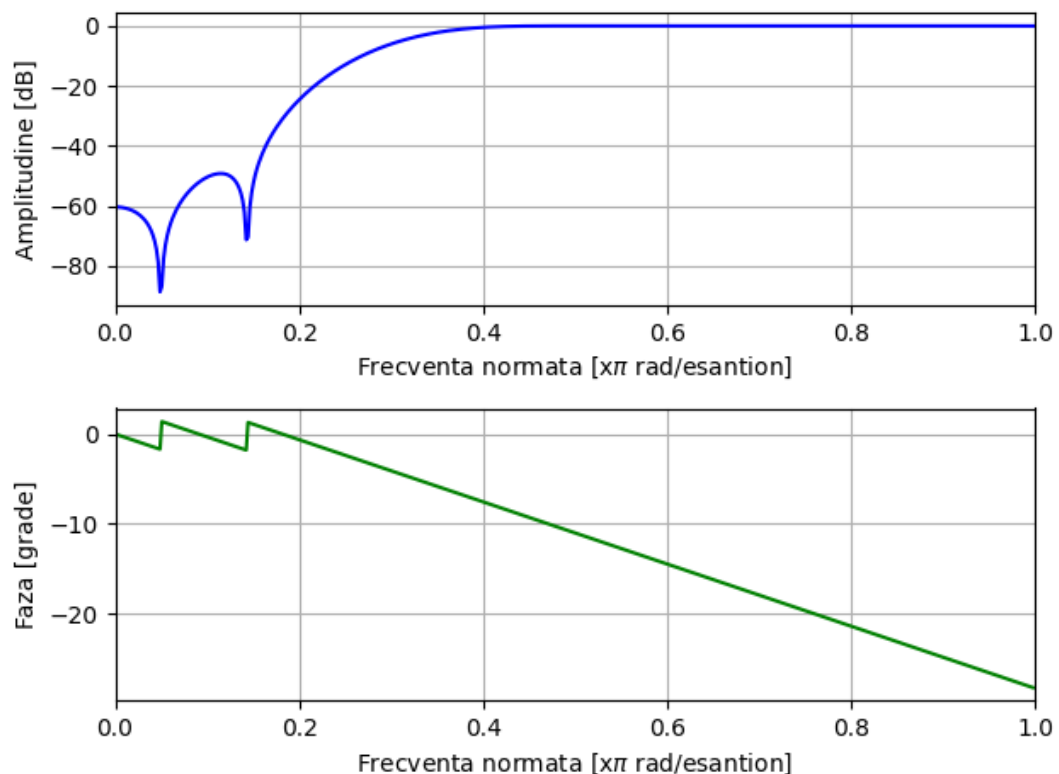


Figura 5. Exemplu de caracteristici de amplitudine și fază pentru un FIR de tip FTS

1.5 Prelucrarea semnalelor audio

În Python este posibilă înregistrarea, prelucrare și redarea semnalelor audio. Un pachet foarte util pentru aceste aplicații este *sounddevice*.

Folosind funcția *rec(t,samplerate=f, channels=n)* se pot înregistra secvențe audio la microfonul calculatorului. Variabila *t* reprezintă durata înregistrării în secunde înmulțită cu frecvența de eșantionare, *f* frecvența de eșantionare, iar *n* numărul de canale pe care se face înregistrarea.

Redarea semnalelor audio se face prin funcția *play(r,f)*, unde *r* este înregistrarea audio, iar *f* este frecvența de eșantionare.

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

fs = 12000 #Frecventa de esantionare
sec = 3 # Durata inregistrarii

rec = sd.rec(int(sec * fs), samplerate=fs, channels=1)
sd.wait() #Asteapta pana la finalizarea inregistrarii
sd.play(rec,fs) #Reda inregistrarea
sd.wait()

#Reprezentare grafica
plt.plot(np.linspace(0,3,fs*sec),rec)
plt.xlim([0,3])
plt.xlabel('Timp [s]')
plt.show()
```

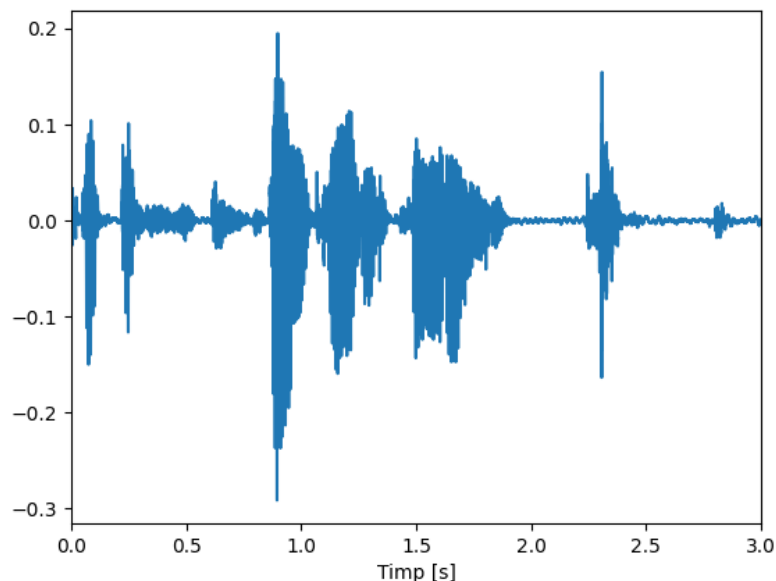


Figura 6. Secvență audio înregistrată

Se pot, de asemenea încărca fișiere audio deja salvate pe calculator cu ajutorul pachetului `scipy`. Același pachet conține și funcția de scriere a fișierelor audio.

```
import sounddevice as sd
from scipy.io.wavfile import write
from scipy.io.wavfile import read

fs, rec = read('fisier_audio.wav') #Citeste fisier_audio existent, fs este
frecventa de esantionare
sd.play(rec, fs) #Reda fisierul
sd.wait()
write('output.wav', fs, rec) # Salveaza fisierul audio
```

Pentru a analiza semnalele audio din punct de vedere al frecvențelor putem realiza o reprezentare de tip spectrogramă prin funcția *specgram(y)*, unde *y* este variabila ce se dorește a fi reprezentată.

```
import sounddevice as sd
from scipy.io.wavfile import write
import matplotlib.pyplot as plt

fs = 12000 #Frecventa de esantionare
sec = 3 # Durata inregistrarii

#Inregistrare audio
rec = sd.rec(int(sec * fs), samplerate=fs, channels=1)
sd.wait()

rec=rec.flatten() #redimensionarea inregistrarii intr-un vector
write('output.wav', fs, rec) #salvare inregistrare

plt.specgram(rec, Fs=fs)
plt.xlabel('Timp [s]')
plt.ylabel('Frecventa')

plt.colorbar()#Afisaza legenda culorilor

plt.show()
```

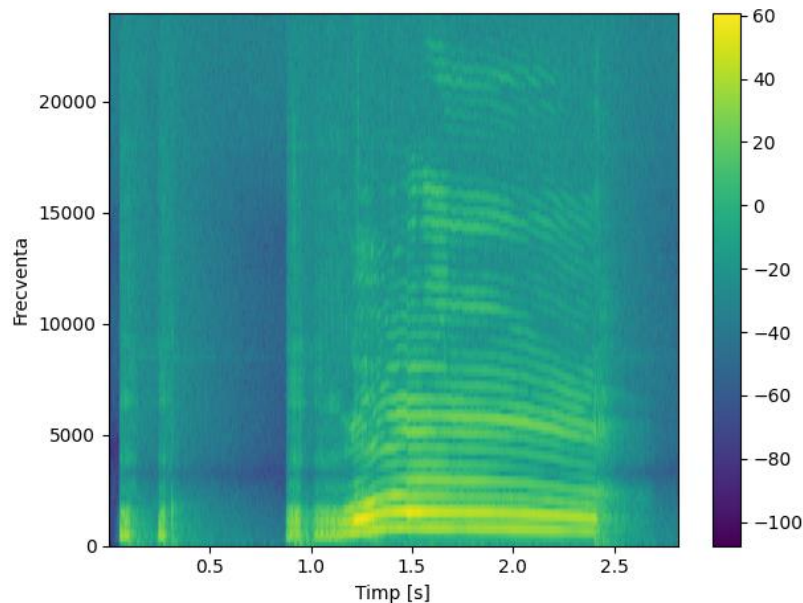



Figura 7. Spectrogramă a unui clip audio realizată cu funcția specgram()

Figura 7 este un exemplu de spectrogramă. Pe axa Ox este reprezentat timpul, pe Oy frecvența normală, iar culorile indică amplitudinea semnalului. Se observă ca semnalul vocal este concentrat în partea de jos a spectrogramei.

De asemenea se observă prezența unor frecvențe mai înalte parazite. Acestea pot filtrate cu ajutorul unui filtru FIR FTJ și a operației de convoluție.

```
import sounddevice as sd
from scipy import signal
from scipy.io.wavfile import read
import matplotlib.pyplot as plt

fs,rec=read('output.wav') #Citire inregistrare
rec=rec.flatten()#redimensionarea inregistrarii intr-un vector

b=signal.firwin(50,0.3,pass_zero='lowpass')#Definire filtru FIR FTJ de ordin 50
y=signal.convolve(rec,b) #Convolutia inregistrarii cu filtrul

#Spectrograma
plt.specgram(y,Fs=fs)
plt.xlabel('Timp [s]')
plt.ylabel('Frecventa')
plt.colorbar()#Afisaza legenda culorilor

plt.show()
```

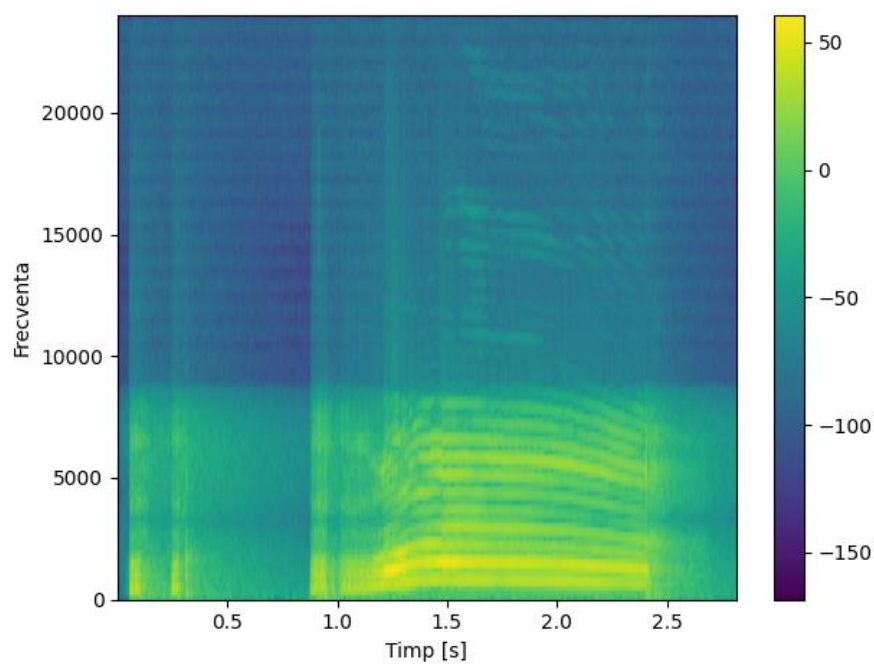


Figura 8. Spectrograma semnalului filtrat

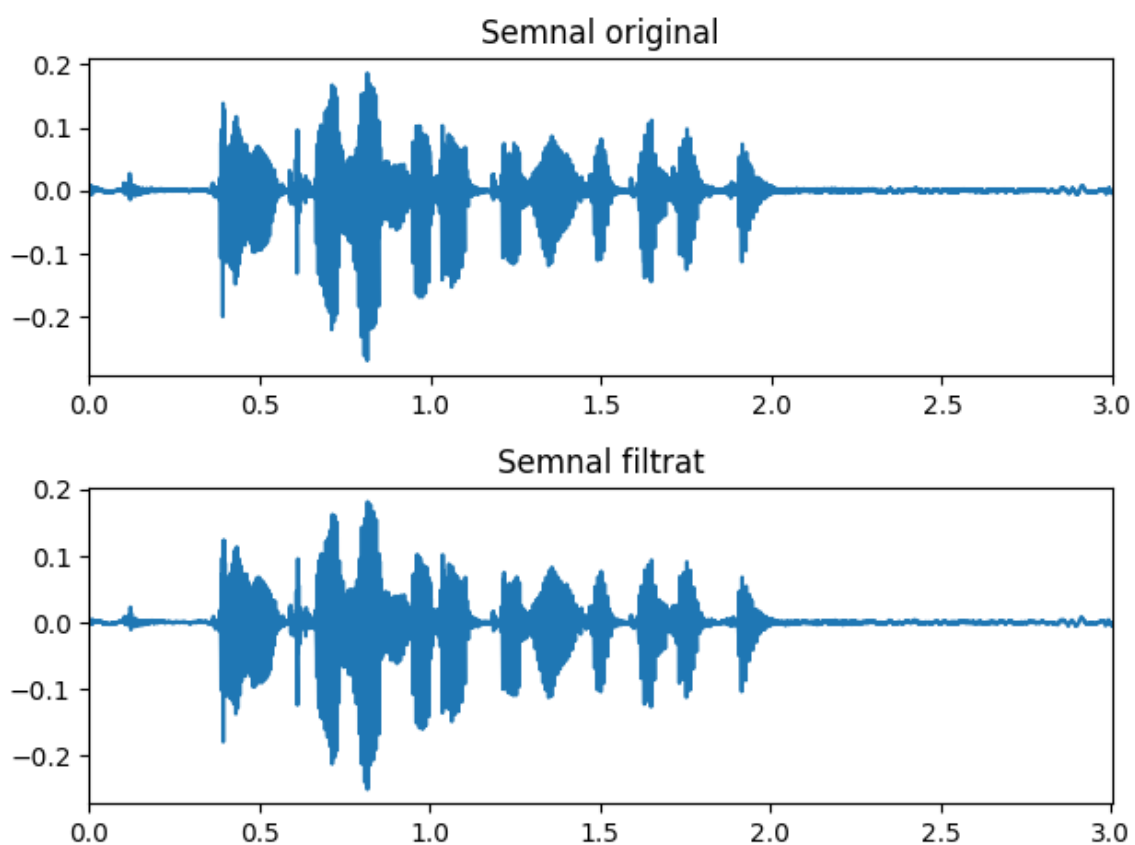


Figura 9. Comparație între semnalul înregistrat și cel filtrat

2. Modul de lucru

Se parcurg următoarele cerințe, notându-se comanda executată și rezultatul acesteia.

- 1) Definiți vectorul z ca fiind convoluția a doi vectori x și y ;
- 2) Reprezentați caracteristicile de amplitudine și fază pentru un filtru FIR cu 32 de termeni de tip FTJ și frecvență critică 0.6;
- 3) Reprezentați caracteristicile de amplitudine și fază pentru un filtru FIR cu 53 de termeni de tip FTS și frecvență critică 0.45;
- 4) Înregistrați un clip audio de 7 secunde și reprezentați spectrograma acestuia;
- 4) – alt. Citiți unul din fișierele de tip .wav puse la dispoziție pentru această lucrare (*alternativă la punctul anterior pentru lucrul cu lucrul cu Raspberry Pi*)
- 5) Filtrați semnalul audio înregistrat anterior folosind un FTJ cu orice număr de termeni și cu frecvență de tăiere aleasă astfel încât să se filtreze cât mai bine frecvențele parazite înalte. Reprezentați spectrograma semnalului filtrat.

2.1 Indicații Raspberry Pi

Înainte de rularea programelor care folosesc semnale audio este necesară instalarea pachetului *libportaudio2* pentru sistemul de operare Raspberry Pi OS. Acest lucru se poate face rulând în terminal următoarea comandă:

```
sudo apt-get install libportaudio2
```

Pentru a putea utiliza pe deplin pachetele *numpy* și *scipy* este posibil să fie nevoie de rularea următoarelor comenzi în terminal:

```
sudo apt-get install python-dev libatlas-base-dev  
sudo apt-get update -fix-missing
```

3. Lucrarea 3 în MATLAB/Octave

Această secțiune cuprinde exemplele de cod prezentate anterior, dar adaptate pentru a fi utilizate în MATLAB/Octave

3.1 Convoluție

```
x=[1 2 -1];  
h=[0 0 3 2 1];  
y=conv(x,h); %Convolutia semnalelor x si h  
n=[0:6];
```

```
%Reprezentarea grafica  
subplot(2,2,1)  
stem(0:2,x)  
xlim([0 6])  
xlabel('n')  
ylabel('x[n]')  
subplot(2,2,2)  
stem(0:4,h)  
xlim([0 6])  
xlabel('n')  
ylabel('h[n]')  
subplot(2,1,2)  
stem(0:6,y)  
xlabel('n')  
ylabel('y[n]')
```

3.2 Filtre FIR

Filtrele de tip FIR pot fi definite în Octave și MATLAB prin intermediul funcției *fir1(n,w,t)*, unde *n* reprezintă ordinul filtrului, *w* frecvența de tăiere, iar *t* tipul filtrului. Pentru un FTJ sau FTS *w* este un scalar, iar pentru FTB și FTP un vector cu 2 elemente (frecvențele de tăiere). Frecvența ia valori de la 0 la 1, unde 1 reprezintă frecvența Nyquist (jumătate din frecvența de eșantionare). Argumentele pentru tipurile de filtre sunt: "low", "high", "stop", "bandpass". **Atenție:** funcția *fir1()* face parte din pachetul **signal**. Pentru a o utiliza trebuie încărcat mai întâi pachetul respectiv.

Pentru a afișa caracteristicile de amplitudine și fază ale unui filtru digital folosim comanda *freqz()*.

```
freqz(fir1(24,0.3,"high")) %FIR FTS cu frecventa de taiere egala cu 0.3 din  
frecventa Nyquist
```

3.3 Prelucrarea semnalelor audio

Octave și MATLAB permit înregistrarea, prelucrare și redarea semnalelor audio. Folosind funcția *record(t,f)* se pot înregistra secvențe audio la microfonul calculatorului. Variabila *t*

reprezintă lungimea înregistrării în secunde, iar f frecvența de eșantionare (valoarea implicită este de 8000 Hz).

Redarea semnalelor audio se face prin funcțiile *sound(s,f)* sau *soundsc(s,f)*. Variabila s reprezintă semnalul redat, iar f frecvența de eșantionare folosită la înregistrarea secvenței. Cele două funcții se diferențiază prin faptul că *soundfc()* redimensionează semnalul audio pentru a fi mai bine auzit.

```
R=record(5); %Inregistreaza o secventa audio de 5 secunde
sound(R) %Reda clipul audio
soundsc(R) %Reda clipul audio
```

Se pot, de asemenea, încărca fișiere audio deja salvate pe calculator cu funcția *wavread(nume_fișierului)*. Salvarea în format de fișier audio se face cu funcția *wavwrite(y, nume_fișier)*, unde y este variabila ce se dorește a fi salvată.

Pentru a analiza semnalele audio din punct de vedere al frecvențelor putem realiza o reprezentare de tip spectrogramă prin funcția *specgram(y)*, unde y este variabila ce se dorește a fi reprezentată.

3.4 Filtrare audio

```
F=fir1(50,0.4); %Definire filtru FIR de ordin 50 (implicit FTJ)
F=F'; %Transpunerea elementelor filtrului
S=conv(R,F) %Convolutia semnalului audio R cu filtrul F
soundsc(S) %Redare audio a semnalului filtrat
specgram(S) %Reprezentare spectrala
```