

Lucrarea 8. Analiza spectrală a semnalelor

Până acum am privit un semnal din perspectiva evoluției amplitudinii acestuia în timp. Totuși, pentru o mare parte a aplicațiilor de procesare a datelor este util și necesar să analizăm semnalul din punctul de vedere al oscilațiilor ce compun semnalul. Astfel, analiza spectrală implică transformarea semnalului din domeniul timp în domeniul frecvențelor.

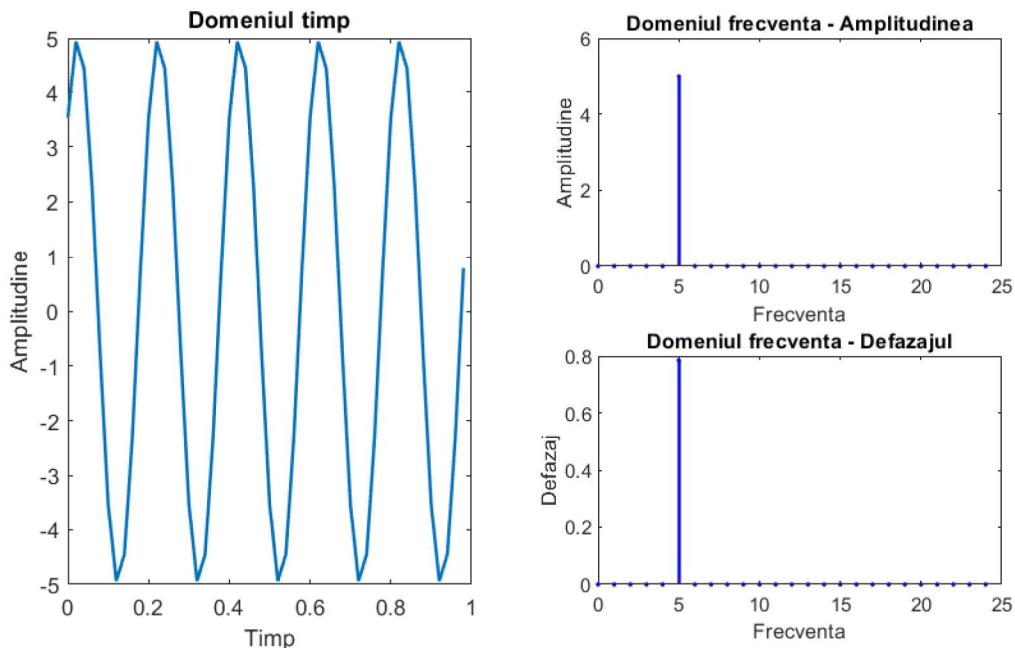


Figura 1 – Semnal sinusoidal în domeniu timp și în domeniu frecvențelor

Analiza spectrală are la bază un concept matematic dezvoltat în secolul al XIX-lea de către Jean Baptiste Fourier: seriile Fourier. Acest concept a fost inițial dezvoltat în timpul invaziei lui Napoleon în Egipt pentru a analiza conduction termică cu scopul de a soluționa problema supraîncălzirii tunurilor. Ulterior, același matematician a generalizat seriile Fourier rezultând Transformata Fourier. Mai târziu, apariția calculatoarelor și nevoia analizei semnalelor digitale au condus la Transformata Fourier Discretă și la Transformata Fourier Rapidă.

În domeniul procesării datelor, analiza spectrală își găsește utilitatea într-o multitudine de procese, precum:

- eliminarea zgomotului din semnale;
- compresie audio / video / imagini;

- identificarea sursei semnalului (exemplu: semnalul de un seismometru a fost cauzat de un cutremur sau de un test nuclear subteran);
- identificarea defectelor în fază incipientă.

8.1. Transformata Fourier

Transformata Fourier a unui semnal continuu [13]:

$$\hat{x}(\omega) = \mathcal{F}(x(t)) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-i \cdot 2\pi \cdot \omega \cdot t} dt$$

transformă semnalul continuu $x(t)$ din domeniul timpului în semnalul continuu $\hat{x}(\omega)$ în domeniul frecvențelor.

Transformata Fourier $\hat{x}(\omega)$ reprezintă o funcție complexă, ce poate fi scrisă în funcție de modul și fază: $\hat{x}(\omega) = Re\{\hat{x}(\omega)\} + i \cdot Im\{\hat{x}(\omega)\} = |\hat{x}(\omega)| \cdot e^{i \cdot \varphi(\omega)}$ [13].

Astfel, modulul $|\hat{x}(\omega)|$ reprezintă amplitudinea, iar $\varphi(\omega)$ faza sinusoidei de frecvență ω ce face parte din descompunerea spectrală a semnalului $x(t)$ [13]. Valorile amplitudinilor alcătuiesc **spectrul de amplitudini** al semnalului, iar valorile fazelor **spectrul de faze**.

Relația inversă de transformare care permite recuperarea unui semnal din spectrul lui, relație numită și **Transformata Fourier inversă**, este [13]:

$$x(t) = \mathcal{F}^{-1}(\hat{x}(\omega)) = \frac{1}{2\pi} \cdot \int_{-\infty}^{+\infty} \hat{x}(\omega) \cdot e^{i \cdot 2\pi \cdot \omega \cdot t} d\omega$$

8.1.1. Transformata Fourier Discretă

Transformata Fourier Discretă este varianta eșantionată a transformatei Fourier continue [5]. Astfel, transformata Fourier discretă a unui semnal $x[n]$ format din N eșantioane se poate calcula astfel:

$$\hat{x}[k] = \mathcal{F}(x[n]) = \sum_{n=0}^{N-1} x[n] \cdot e^{-i \cdot \frac{2\pi}{N} k \cdot n}, \quad k = \overline{0, N-1}$$

unde fiecarui element $\hat{x}[k]$ îi corespunde o frecvență $\omega_k = k \cdot \frac{f_s}{N}$ [5].

Relația inversă de transformare care permite recuperarea unui semnal din spectrul lui, relație numită și **Transformata Fourier Discretă Inversă**, este [5]:

$$x[n] = \mathcal{F}^{-1}(\hat{x}[k]) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}[k] \cdot e^{-i \frac{2\pi}{N} k \cdot n}, \quad n = \overline{0, N-1}$$

Cunoscând relația Transformatei Fourier Discrete, se poate observa că o implementare software a acestui algoritm va avea un timp de rulare direct proporțional cu numărul de termeni $x[n] \cdot e^{-i \frac{2\pi}{N} k \cdot n}$ calculați, adică N^2 [5]. Deci timpul de rulare va crește exponențial cu numărul de eșantioane. Putem spune că pentru un semnal cu un număr de eșantioane destul de mare (ex: 1 secundă de semnal audio => 48000 eșantioane => aproximativ $2.3 \cdot 10^9$ termeni de calculat) folosirea relației Transformatei Fourier Discrete pentru a calcula spectrul semnalului nu este fezabilă.

8.1.2. Transformata Fourier Rapidă

Transformata Fourier Rapidă este o tehnică ce ne permite să calculăm mai rapid spectrul unui semnal. Există mai mulți algoritmi pentru a implementa Transformata Fourier Rapidă. Unul din cei mai cunoscuți este **algoritmul cu decimare în frecvență**. Este un algoritm de tip **divide et impera** care împarte Transformata Fourier Discretă a unui semnal într-o secvență de Transformate Fourier Discrete ale unor porțiuni de semnal [5].

Considerăm semnalul $x = \{x[0], x[1], x[2], x[3], \dots, x[N-1]\}$ a căruia reprezentare în domeniul frecvențelor o notăm cu $\hat{x} = \mathcal{F}(x)$. Vom împărți eșantioanele semnalului în două șiruri de eșantioane [5]:

- Eșantioanele pare: $x_1 = \left\{x_1[0], x_1[1], x_1[2] \dots x_1\left[\frac{N}{2} - 1\right]\right\} = \{x[0], x[2], x[4], \dots\}$ cu $\hat{x}_1 = \mathcal{F}(x_1)$
- Eșantioanele impare: $x_2 = \left\{x_2[0], x_2[1], x_2[2] \dots x_2\left[\frac{N}{2} - 1\right]\right\} = \{x[1], x[3], x[5], \dots\}$ cu $\hat{x}_2 = \mathcal{F}(x_2)$

Plecând de la formula Transformatei Fourier Discrete [5]:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i \frac{2\pi}{N} k \cdot n} = \sum_{m=0}^{N/2-1} x[2m] \cdot e^{-i \frac{2\pi}{N} k \cdot 2m} + \sum_{m=0}^{N/2-1} x[2m+1] \cdot e^{-i \frac{2\pi}{N} k \cdot (2m+1)}$$

Observăm că pentru $m = \overline{0, N/2 - 1}$ $x[2m] = x_1[m]$ și $x[2m+1] = x_2[m]$. Deci:

$$\hat{x}[k] = \sum_{m=0}^{N/2-1} x_1[m] \cdot e^{-i \frac{2\pi}{N/2} k \cdot m} + \sum_{m=0}^{N/2-1} x_2[m] \cdot e^{-i \frac{2\pi}{N/2} k \cdot m - i \frac{2\pi}{N} k} [5]$$

$$\hat{x}[k] = \sum_{m=0}^{N/2-1} x_1[m] \cdot e^{-i \frac{2\pi}{N/2} k \cdot m} + \sum_{m=0}^{N/2-1} x_2[m] \cdot e^{-i \frac{2\pi}{N/2} k \cdot m} \cdot e^{-i \frac{2\pi}{N} k} [5]$$

Cunoscând proprietatea de periodicitate a Transformatei Fourier Discrete se obține [5]:

$$\hat{x}[k] = \hat{x}_1 \left[k \bmod \frac{N}{2} \right] + e^{-i \frac{2\pi}{N} k} \cdot \hat{x}_2 \left[k \bmod \frac{N}{2} \right]$$

Se observă că Transformata Fourier Discretă a unui semnal se poate calcula din Transformatele Fourier Discrete ale eșantioanelor pare și ale eșantioanelor impare, rezultând reducerea numărului de termeni calculați de la N^2 la $2 \cdot \frac{N^2}{4} = \frac{N^2}{2}$, adică la jumătate.

Dacă pentru a calcula Transformata Fourier Discretă a eșantioanelor pare și a eșantioanelor impare folosim aceeași metodă, și tot aşa, până ajungem la un singur eșantion, vom obține o reducere a numărului de termeni calculați de la N^2 la $N \cdot \log_2 N$.

Totuși acest algoritm funcționează doar pentru un număr de eșantioane N putere a lui 2. Dacă N nu este putere a lui 2 putem fie să eliminăm eșantioane până ajungem la o putere a lui 2 sau să adăugăm eșantioane cu valoarea 0 până ajungem la o putere a lui 2.

Aplicând același procedeu și pentru **Transformata Fourier Discretă Inversă**, vom obține algoritmul pentru **Transformata Fourier Rapidă Inversă**.

Pentru a calcula în *Python* **Transformata Fourier Rapidă** directă și inversă vom folosi funcțiile *fft* și *ifft* din sub-modulul *fft* al librăriei **SciPy**. Vom determina valorile spectrului de amplitudini calculând modulul eșantioanelor complexe ale Transformatei Fourier folosind funcția *abs* din librăria **NumPy** și valorile spectrului de faze calculând argumentul eșantioanelor complexe ale Transformatei Fourier folosind funcția *angle* din librăria **NumPy**. Atunci când calculăm Transformata Fourier Inversă, trebuie să știm că eșantioanele rezultante vor fi tot numere complexe (dar cu partea imaginară 0), deci va trebui să păstrăm doar partea reală. Pentru aceasta vom folosi funcția *real* din librăria **NumPy**.

Exemplu:

Dorim să aplicăm **Transformata Fourier Rapidă** semnalului $x(t) = 1 + 3 \cdot \sin(2 \cdot \pi \cdot 1 \cdot t) + \sin\left(2 \cdot \pi \cdot 4 \cdot t + \frac{\pi}{2}\right)$ eșantionat cu o frecvență $f_s = 50 \text{ Hz}$ pe o perioadă de o secundă și să afișăm spectrele de amplitudini și de faze. Vom încerca apoi să refacem semnalul prin folosirea **Transformatei Fourier Rapidă Inversă**.

main.py

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

```

tf, fs = 1, 50
t = np.arange(0, tf, 1/fs)
N = len(t)
x = 1 + 3*np.sin(2*np.pi*1*t) + np.sin(2*np.pi*4*t+np.pi/2)

plt.subplot(2,2,1), plt.plot(t, x, 'r')
plt.xlabel('Time [s]'), plt.ylabel('Amplitude')
plt.title('Semnalul Original')

xhat = scipy.fft.fft(x)
w = np.arange(0, N) * fs / N

plt.subplot(2,2,3), plt.stem(w, np.abs(xhat), 'b')
plt.xlabel('Freq [Hz]'), plt.ylabel('|X(w)|')
plt.title('Spectrul de amplitudini')

plt.subplot(2,2,4), plt.stem(w, np.angle(xhat), 'b')
plt.xlabel('Freq [Hz]'), plt.ylabel('phi(w)')
plt.title('Spectrul de faze')

plt.subplot(2,2,2),
plt.plot(t,np.real(scipy.fft.ifft(xhat)), 'g')
plt.xlabel('Freq [Hz]'), plt.ylabel('phi(w)')
plt.title('Semnalul refacut')

plt.tight_layout()
plt.show()

```

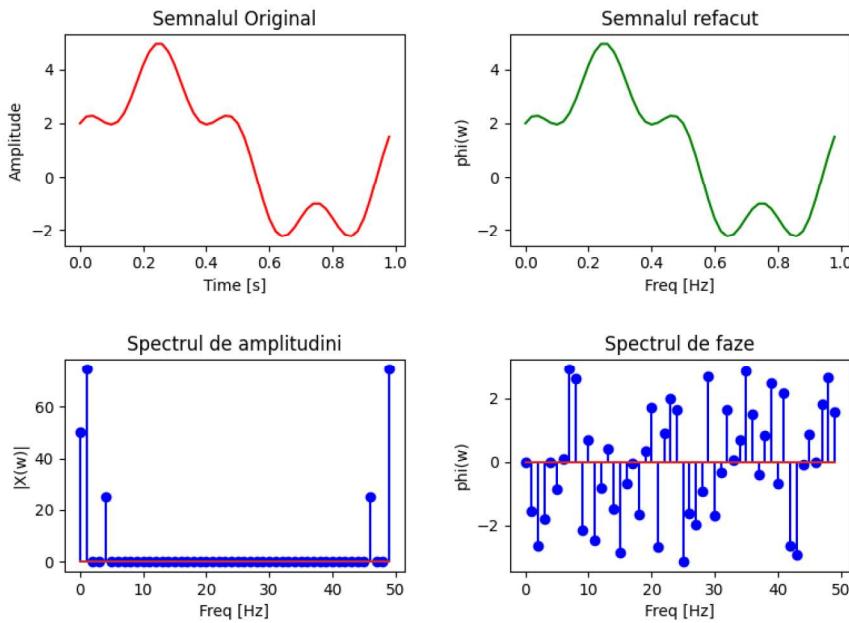


Figura 2 – Exemplu de grafic afișat

Observăm:

- spectrul de amplitudini, exceptând componenta continuă de 0 Hz este simetric față de frecvența Nyquist $\frac{f_s}{2}$. Din această cauză, strict pentru afișarea grafică, putem ascunde această jumătate;
- amplitudinea componentei continue (0 Hz) are o valoare de N ori mai mare decât amplitudinea componentei continue a semnalului în timp, în timp ce amplitudinile celorlalte componente sunt de $N/2$ ori mai mari decât amplitudinile componentelor sinusoidale din semnalul în domeniul timp. Astfel, dacă dorim să reprezentăm grafic valorile exacte ale componentelor, se impune o normalizare a amplitudinilor;
- se observă că defazajul arată foarte zgomotos rezultate din cauza erorilor de calcul care apar pentru componente cu amplitudine 0, pentru aceasta va trebui să triem defazajele și să le stabilim 0 pentru frecvențele ale căror amplitudini sunt mai mici de un anumit prag (spre exemplu mai mici de 0.0001);
- de asemenea, pentru a afișa corect defazajele sinusoidelor la defazajele obținute cu transformata Fourier trebuie să adăugăm $\pi/2$ pentru $k > 0$.

Astfel, dacă dorim o afișare mai ușor de interpretat a spectrelor de amplitudini și de fază pentru semnalul din exemplul anterior, vom proceda astfel:

main.py

```

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

tf, fs = 1, 50
t = np.arange(0, tf, 1/fs)
N = len(t)
x = 1 + 3*np.sin(2*np.pi*1*t) + np.sin(2*np.pi*4*t+np.pi/2)

xhat = scipy.fft.fft(x)

# folosim prima jumătate a spectrului
w=np.arange(0,int(N/2))*fs/N
xhat_amp = np.abs(xhat[:int(N/2)])
xhat_phi = np.angle(xhat[:int(N/2)])

xhat_amp *= 2 / N; xhat_amp[0] /= 2 # normalizăm valorile
plt.subplot(2, 1, 1), plt.stem(w, xhat_amp, 'b')
plt.xlabel('Freq [Hz]', color='blue'), plt.ylabel('|X(w)|', color='blue')
plt.title('Spectrul de amplitudini')

# adaugam pi/2 pentru w > 0
xhat_phi[1:] += np.pi / 2
# triem defazajele (doar pt comp. cu amplitudinea > 0.0001)

```

```
xhat_phi[xhat_amp < 0.0001] = 0

plt.subplot(2, 1, 2), plt.stem(w, xhat_phi, 'b')
plt.xlabel('Freq [Hz]', color='blue'), plt.ylabel('phi(w)', color='blue')
plt.title('Spectrul de faze')

plt.tight_layout(), plt.show()
```

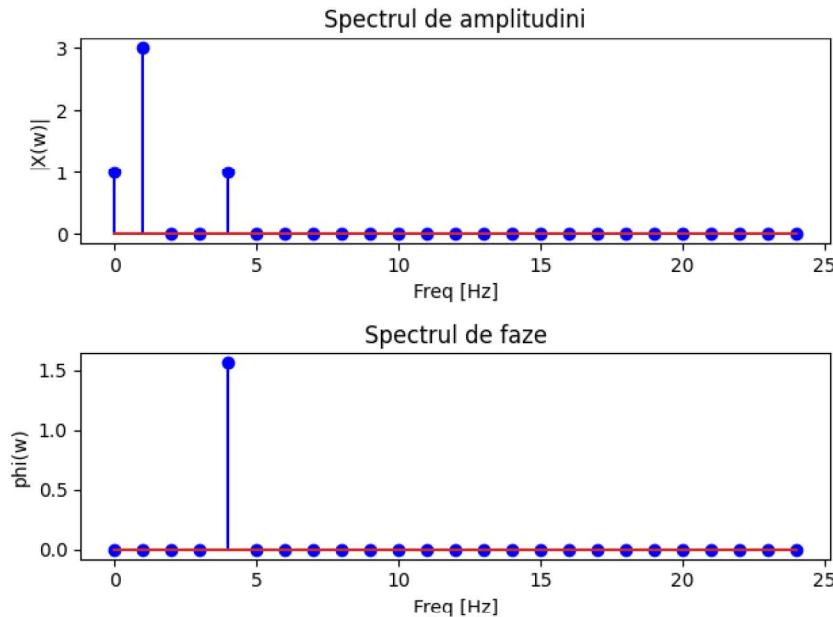


Figura 3 – Exemplu de grafic afișat

Exemplu de utilizare a Transformatei Fourier pentru eliminarea zgomotului alb dintr-un semnal:

main.py

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

tf, fs = 1, 1000
t = np.arange(0, tf, 1 / fs)
N = len(t)
x =
np.sin(2*np.pi*5*t)+np.sin(2*np.pi*8*t)+2.5*np.random.randn(N)

plt.subplot(2, 2, 1), plt.plot(t, x, 'r')
plt.xlabel('Time [s]', color='blue'), plt.ylabel('Amplitude')
plt.title('Semnalul Original')

xhat = scipy.fft.fft(x)
```

```
w = np.arange(0, N) * fs / N

plt.subplot(2, 2, 3), plt.stem(w, np.abs(xhat), 'b')
plt.xlabel('Freq [Hz]'), plt.ylabel('|X(w)|')
plt.title('Spectrul de amplitudini')

plt.subplot(2, 2, 4), plt.stem(w, np.angle(xhat), 'b')
plt.xlabel('Freq [Hz]'), plt.ylabel('phi(w)')
plt.title('Spectrul de faze')

# eliminare frecvențe de amplitudine mica - eliminare zgomot alb
xhat[np.abs(xhat) < 300] = 0
plt.subplot(2, 2, 2), plt.plot(t, np.real(scipy.fft.ifft(xhat)), 'g')
plt.xlabel('Freq [Hz]'), plt.ylabel('phi(w)')
plt.title('Semnalul refacut')

plt.tight_layout(), plt.show()
```

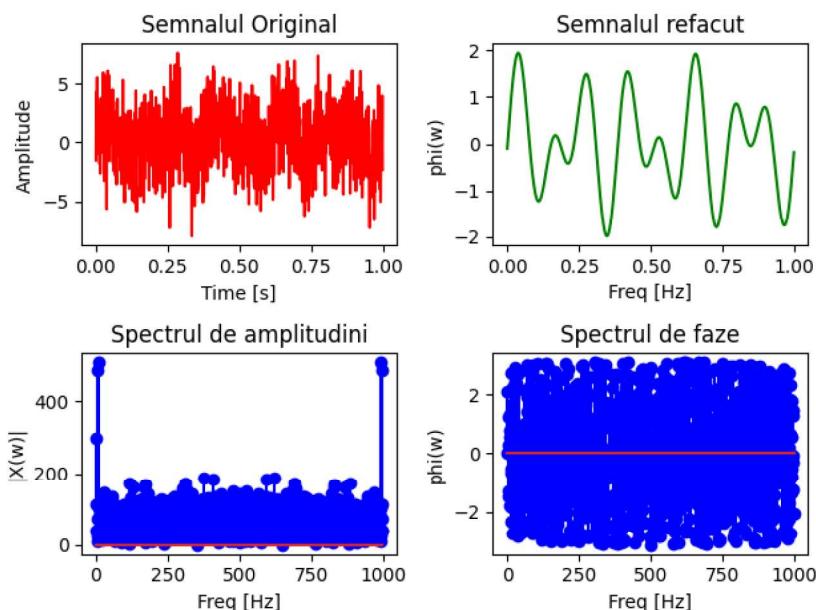


Figura 4 – Exemplu de grafic afișat

8.2. Teorema lui Parceval

Definiție: Energia unui semnal $x(t)$ se definește ca aria de sub pătratul magnitudinii semnalului: $E_s = \int_{-\infty}^{+\infty} |x(t)|^2 dt$ [14]. Evident, pentru că lucrăm cu semnale discrete în timp cu un număr finit de eșantioane, formula matematică a energiei semnalului devine o sumă finită: $E_s = \sum_{n=0}^{N-1} |x[n]|^2$.

Teorema lui Parceval: Trecerea din domeniul timp în domeniul frecvențelor utilizând Transformata Fourier nu modifică energia semnalului: $\sum_{n=0}^{N-1}|x[n]|^2 = \sum_{k=0}^{N-1}|\hat{x}[k]|^2$ [14].

Această teoremă are implicații majore în compresia semnalelor (audio, imagine, video). Astfel, știind că energia semnalului în domeniul timp este egală cu energia semnalului în domeniul frecvențelor deducem că, dacă aducem o modificare minoră energiei în domeniul frecvențelor, ea se va manifesta ca o modificare minoră în domeniul timp.

Exemplu: un semnal audio de 10 secunde eșantionat cu o frecvență de eșantionare de 48 kHz necesită pentru stocare un spațiu de aproximativ 400 KB. Dacă aplicăm Transformata Fourier semnalului nostru, vom observa că nu toate componentele spectrale au o amplitudine considerabilă, deci nu toate contribuie în mod egal la energia semnalului. Astfel, putem renunța la 80% din componente spectrale fără a pierde prea mult din energia semnalului, cu alte cuvinte fără a pierde din calitatea semnalului audio. Astfel, pentru stocarea fișierului audio putem stoca semnalul în domeniul frecvențelor (doar cele mai considerabile 20% componente), reducând astfel considerabil utilizarea spațiului de stocare. Acesta este algoritmul care stă la baza formatului MP3.

8.3. Spectrograma

Fie un semnal sinusoidal cu amplitudinea 1 eșantionat cu frecvență de $f_s = 1 \text{ kHz}$ pe o perioadă de 1s a cărui frecvență este de 20 Hz pentru prima jumătate de secundă și 120 Hz. Reprezentarea grafică a semnalului în domeniul timp și în domeniul frecvențelor:

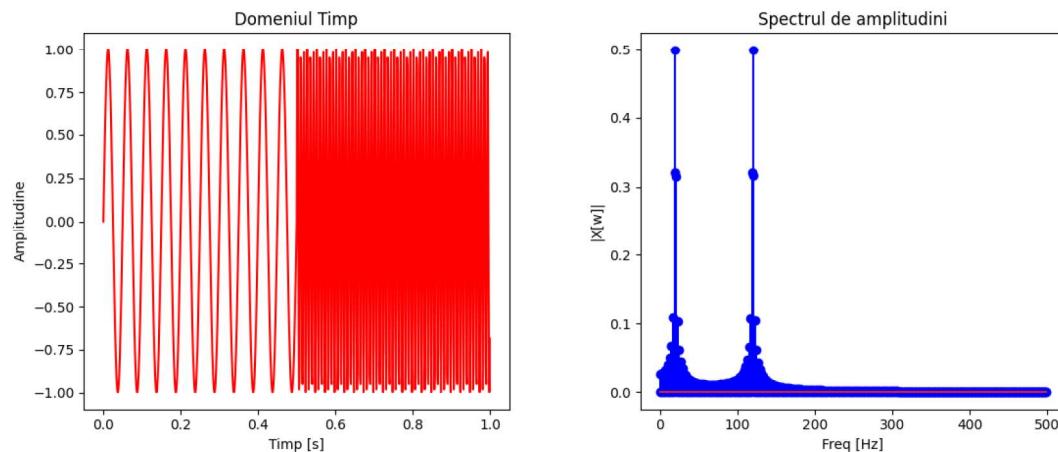


Figura 5 – Semnalul dat în domeniul timp și în domeniul frecvențelor

Dacă analizăm spectrul de amplitudini se pot observa cele două componente spectrale (totuși cu amplitudinea înjumătățită datorită faptului că fiecare din ele se manifestă doar pe jumătate din perioadă). Totuși domeniul frecvențelor nu ne sugerează nimic în privința evoluției componentelor spectrale ale semnalului în timp.

Spectrogramele unui semnal sunt o reprezentare a spectrului de frecvențe al semnalului pe măsură ce semnalul evoluează în timp. Astfel, pentru a realiza spectrograma unui semnal, se folosește o fereastră de NFFT eșantioane care este plimbată în timp peste semnalul nostru. La fiecare moment de timp se calculează FFT pentru porțiunea de semnal de sub fereastra. În final rezultă o serie de spectre care vor fi apoi afișate într-o manieră grafică [15].

Pentru reprezentarea spectrogramei în Python vom folosi funcția `specgram` din sub-modulul `pyplot` al librăriei **Matplotlib**, dând ca parametri: semnalul `x`, frecvența de eșantionare, numărul de eșantioane al fiecărei ferestre și încă doi parametri pentru afișarea amplitudinii: `mode='magnitude'`, `scale='linear'`. Astfel, pentru semnalul dat, vom avea:

main.py

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

tf, fs = 1, 1000
t = np.arange(0, tf, 1/fs)
N = len(t)
x1 = np.sin(2*np.pi*20*t[:int(N/2)])
x2 = np.sin(2*np.pi*120*t[int(N/2):])
x = np.hstack((x1, x2))

# NFFT numărul de eșantioane al fiecărei ferestre
plt.specgram(x, Fs=fs, NFFT=int(fs/5), mode='magnitude', scale='linear')
plt.xlabel('Timp [s]', plt.ylabel('Freq [Hz]')
plt.tight_layout(), plt.show()
```

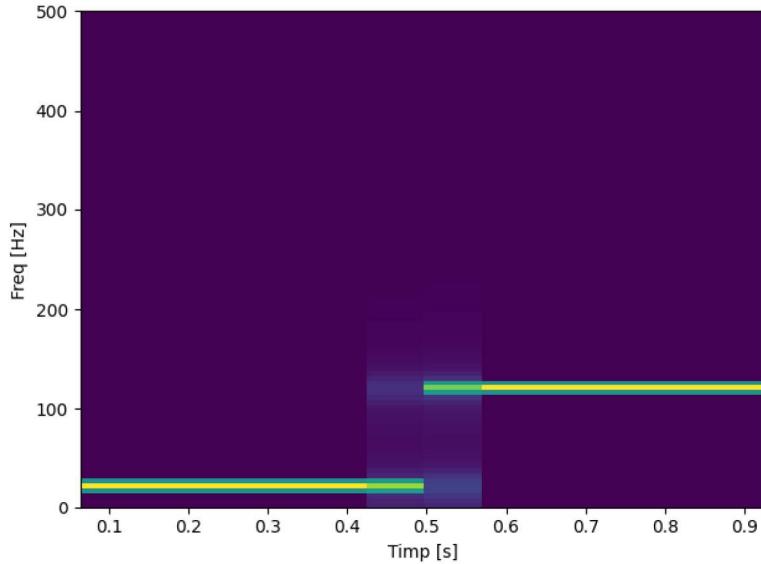


Figura 6 – Exemplu de grafic afișat (2)

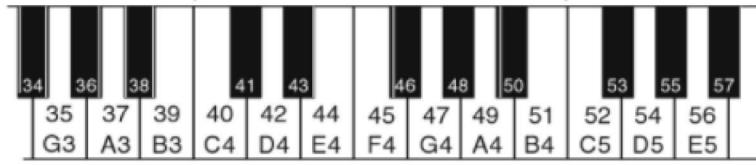
În graficul rezultat, se observă clar cele două componente spectrale și trecerea de la o frecvență la alta la momentul $t = 0.5\text{ s}$. Ar trebui menționat că în cazul spectrogramelor se aplică principiul incertitudinii: cu cât dorim să aflăm mai multe informații în domeniul timp, cu atât vom afla mai puține informații în domeniul frecvențelor, și viceversa. Adică, scăderea dimensiunii ferestrei (o granularitate mai mare, și implicit mai bună în timp) duce la o scădere a granularității în domeniul frecvențelor.

8.4. Exerciții

1. Reprezentați grafic în domeniul timp și în domeniul frecvențelor un semnal $x(t) = 25 + 10 \cdot \sin(2 \cdot \pi \cdot t \cdot 5) + 2 \cdot \sin(2 \cdot \pi \cdot t \cdot 20 + \pi / 4)$ eșantionat cu frecvența de eșantionare $f_s = 100\text{ Hz}$ pe o perioadă de 50s .

2. Generați eșantioanele unui semnal $x(t) = \frac{\sin(2 \cdot \pi \cdot t \cdot 20)}{1-t}$ eșantionat cu $f_s = 1\text{ kHz}$ pe o perioadă de 1s . Reprezentați grafic semnalul, atât în domeniul timp cât și în domeniul frecvențelor. Afisați grafic spectrograma semnalului.

3. Generați una dintre notele de pian din imagine pe o perioadă de 1s cu o frecvență de eșantionare de $f_s = 48 \text{ kHz}$. Afipați grafic nota generată în domeniul frecvențelor [13].



Notele se vor genera ca un semnal sinusoidal de amplitudine 1 și frecvență $f = 440 \cdot 2^{\frac{n-49}{12}} \text{ Hz}$ unde n este numărul notei (exemplu, pentru nota F4, n = 45).

Salvați semnalul într-un fișier „ex4.wav” și ascultați semnalul audio generat.
Salvare se va face cu:

```
scipy.io.wavfile.write('./ex4.wav', fs, x.astype('float32'))
```

4. Generați o secvență de note C4 – D4 – E4 – F4 – G4 – A4 – B4, fiecare notă având o durată de 1s, cu o frecvență de eșantionare de $f_s = 48 \text{ kHz}$. Afipați grafic secvența generată în domeniul frecvențelor și spectrograma acestuia folosind o fereastră de 0.5 s.
5. Achiziționați de la „Temperature Control Lab” valorile de temperatură de la primul senzor pentru 90 de secunde, timp în care încălzitorul va fi setat la 100 %:
 a. reprezentați grafic în domeniul timp semnalul achiziționat;
 b. reprezentați grafic în domeniul frecvențelor semnalul achiziționat.