Lucrarea 2

Semnale continue și discrete

Refamiliarizarea cu diferitele tipuri de semnale continue și discrete, a metodelor de analiză a acestora (serii și transformate Fourier) în Python. Exemplificarea procesului de eșantionare a semnalelor continue. Definirea transformatei Fourier discrete și a algoritmilor de calcul tip transformată Fourier rapidă.

1. Noțiuni teoretice

1.1 Semnale periodice și continue

Semnalele periodice sunt acele semnale care se repetă la un interval fix de timp denumit perioadă.

Forma matematică generală a unui semnal periodic este:

$$s(t) = A\sin(\omega t + \varphi) \tag{1}$$

unde A reprezintă amplitudinea semnalului, ω frecvența unghiulară (pulsația), iar ϕ defazarea semnalului.

Frecvența unui semnal poate fi exprimată în două moduri:

$$\omega\left[\frac{radiani}{s}\right] \quad \text{sau} \quad f\left[\frac{cicluri}{s} = Hz\right] \tag{2}$$

Alegerea formei depinde de aplicație. În calcule este mai potrivit să se folosească frecvența unghiulară ω deoarece funcțiile trigonometrice și programele de calcul acceptă implicit unghiurile măsurate în radiani. În laborator și în aplicațiile practice se folosește, însă, frecvența exprimată în hertzi deoarece toate aparatele de măsură (generatoare de semnal, osciloscoape, analizoare de spectru, etc.) folosesc și ele aceeași unitate de măsură. Legătura dintre cele două forme este următoarea:

$$\omega = 2\pi f \tag{3}$$

O ultimă relație care ne va fi de ajutor și trebuie amintită pornește de la modul în care este definită frecvența unghiulară. Mai exact, ea este exprimată ca fiind schimbarea valorii unui unghi într-un interval de timp:

$$\omega = \frac{d\theta}{dt} \rightarrow \theta = \omega t \tag{4}$$

Folosindu-ne de această relație putem exprima semnalul ca o funcție de timp și ajungem la forma standard menționată la început.

$$\begin{cases} \sin\theta \\ \theta = \omega t \end{cases} \rightarrow s(t) = A\sin(\omega t + \varphi) \tag{5}$$

O privire de ansamblu a semnalelor periodice ce pot fi generate în Python reiese din exemplul următor:

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

#Declarare semnale
t=np.linspace(0,1,1000) #definire timp
f=3 #frecventa semnalului
s=np.sin(2*np.pi*f*t) #semnal sinusoidal
c=np.cos(2*np.pi*f*t) #semnal cosinusoidal
p=signal.square(2*np.pi*f*t,0.5) #semnal dreptunghiular
v=signal.sawtooth(2*np.pi*f*t,0.5) #semnal triunghiular

#Afisare
plt.subplot(4,1,1) #figura cu 4 linii si o coloana. Se reprezinta graficul 1
plt.plot(t,s,linewidth=2)
plt.title("Semnal sinusoidal")
plt.xlabel("timp [s]")
plt.ylabel("s(t)")
plt.subplot(4,1,2) #figura cu 4 linii si o coloana. Se reprezinta graficul 2
plt.plot(t,c,linewidth=2)
plt.title("Semnal cosinusoidal")
plt.xlabel("timp [s]")
plt.ylabel("c(t)")
plt.subplot(4,1,3) #figura cu 4 linii si o coloana. Se reprezinta graficul 3
plt.plot(t,p,linewidth=2)
plt.title("Semnal dreptunghiular")
plt.xlabel("timp [s]")
plt.ylabel("p(t)")
plt.subplot(4,1,4) #figura cu 4 linii si o coloana. Se reprezinta graficul 4
plt.plot(t,v,linewidth=2)
plt.title("Semnal triunghiular")
plt.xlabel("timp [s]")
plt.ylabel("v(t)")

plt.tight_layout() #ajustarea marginilor graficelor
plt.show()
```

Rezultatul executării secvenței de cod anterioare este prezentat în Figura 1.

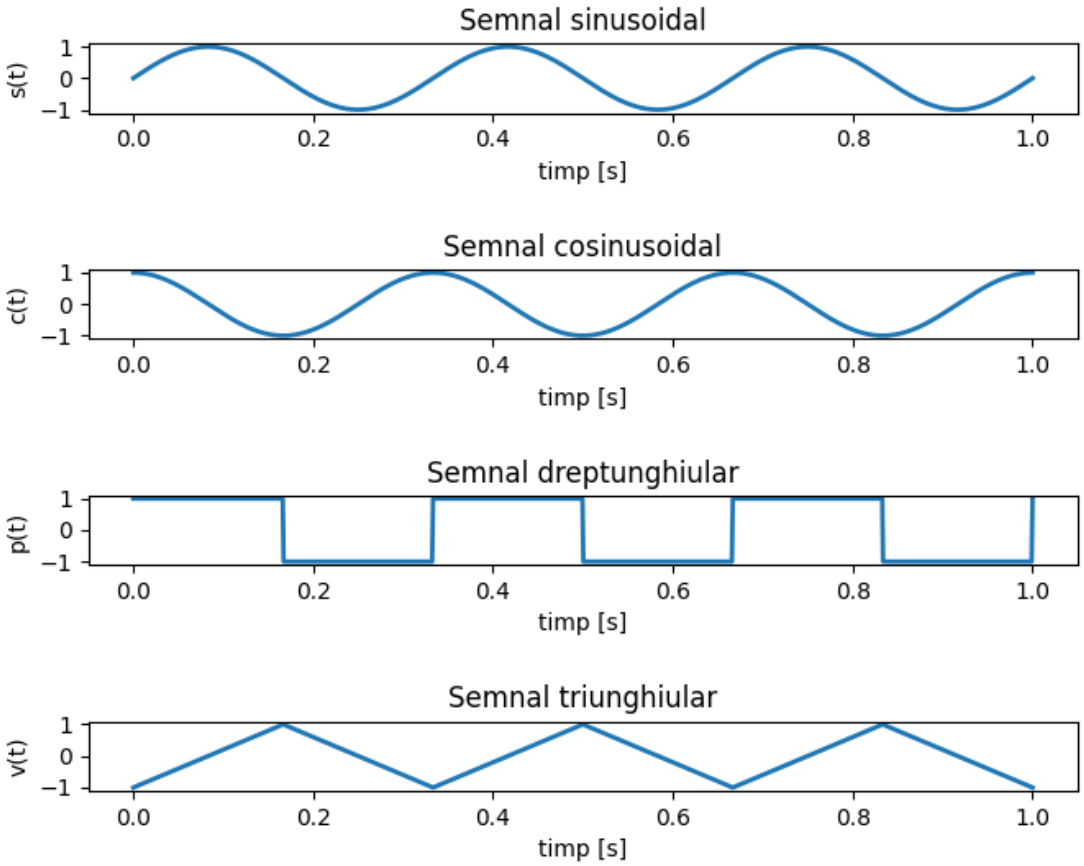

Figura 1. Semnale periodice generate cu Python

Se observă că pentru definirea semnalului dreptunghiular s-a folosit comanda *signal.square(t,d)*, unde t reprezintă variabila în funcție de care se definește semnalul dreptunghiular, iar d este factorul de umplere exprimat în procente (valoare implicită a acestuia este de 0.5).

Semnalul triunghiular a fost generat ca un caz particular al semnalului dinte de fierăstrău prin intermediul funcției *signal.sawtooth(t,w)*. În acest caz t reprezintă variabila în funcție de care se definește semnalul dinte de ferăstrău, iar w reprezintă localizarea maximului semnalului raportat la perioada acestuia (valoarea implicită este 1). Pentru a defini un semnal triunghiular am considerat că maximul semnalului este la jumătatea perioadei.

1.2 Analiza semnalelor periodice – Seria Fourier

Semnalele periodice sunt acele semnale care se repetă la un interval fix de timp denumit perioadă.

Orice semnal periodic s(t) poate fi reprezentat ca o sumă de semnale sinusoidale și cosinusoidale.

$$s(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n \cos(n\Omega t) + b_n \sin(n\Omega t)) \quad (6)$$

Această reprezentare poartă denumirea de *serie Fourier trigonometrică*. Variabila Ω reprezintă frecvența fundamentală a semnalului definită astfel:

$$\Omega = 2\pi f_0 = \frac{2\pi}{T_0} \quad (7)$$

unde f_0 este frecvența fundamentală exprimată în hertzi, iar T_0 perioada fundamentală a semnalului. Coeficienții a_0, a_n și b_n poartă denumirea de coeficienți Fourier și se calculează astfel:

$$a_0 = \frac{2}{T}\int_{-T/2}^{T/2} s(t)\, dt \quad (8)$$

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} s(t)\cos(n\Omega t)\, dt \quad (9)$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} s(t)\sin(n\Omega t)\, dt \quad (10)$$

Semnalele periodice pot fi definite și prin intermediul *seriei Fourier armonice*:

$$s(t) = a_0 + \sum_{n=1}^{\infty} A_n \cos(n\Omega t + \varphi_n) \quad (11)$$

unde coeficienții A_n și φ_n reprezintă amplitudinea, respectiv defazarea componentei n și sunt definite astfel:

$$A_n = \sqrt{a_n^2 + b_n^2} \quad (12)$$

$$\varphi_n = -arctg\left(\frac{b_n}{a_n}\right) \quad (13)$$

Calculând acești coeficienți putem reprezenta grafic spectrele de amplitudine și de fază ale semnalului periodic.

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

#Definire semnal
dt=0.001
t=np.arange(0+dt,2+dt,dt)
s=5*np.sin(2*np.pi*2*t)+2*np.sin(2*np.pi*5*t)+0.5*np.sin(2*np.pi*10*t)

#Reprezentarea in timp a semnalului
plt.subplot(1,2,1)
plt.plot(t,s,linewidth=2)
plt.title("Semnalul periodic")
plt.xlabel("timp [s]")
plt.ylabel("s(t)")

#Definire vectori Fourier
n=30 #Numarul de frecvente analizate
A=np.zeros((n))
B=np.zeros((n))
M=np.zeros((n))
P=np.zeros((n))

#Calcularea coeficientilor Fourier
for i in range(1,n):
    A[i]=sum(s*np.cos(2*np.pi*i*t))*dt
    B[i]=sum(s*np.sin(2*np.pi*i*t))*dt
    M[i]=np.sqrt(np.power(A[i],2)+np.power(B[i],2))
    P[i]=-np.arctan(B[i]/A[i])

#Reprezentarea spectrelor
plt.subplot(2,2,2)
plt.stem(M)
plt.title("Spectru amplitudini")
plt.xlabel("Frecventa [Hz]")
plt.ylabel("Amplitudine [V]")
plt.subplot(2,2,4)
plt.stem(P)
plt.title("Spectru faze")
plt.xlabel("Frecventa [Hz]")
plt.ylabel("Faza [rad]")

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

Rezultatele analizei Fourier sunt ilustrate în Figura2.

Secvența de cod conține o buclă *for*. În Python există mai multe moduri de a folosi acest tip de buclă. În acest caz prin intermediul comenzii *range()* se definește valoarea inițială și cea finală a contorului.

Atenție! Toate comenzile ce se doresc a fi incluse în bucla *for* **trebuie scrise** la **un tab distanță** față de rândul pe care se declară bucla.

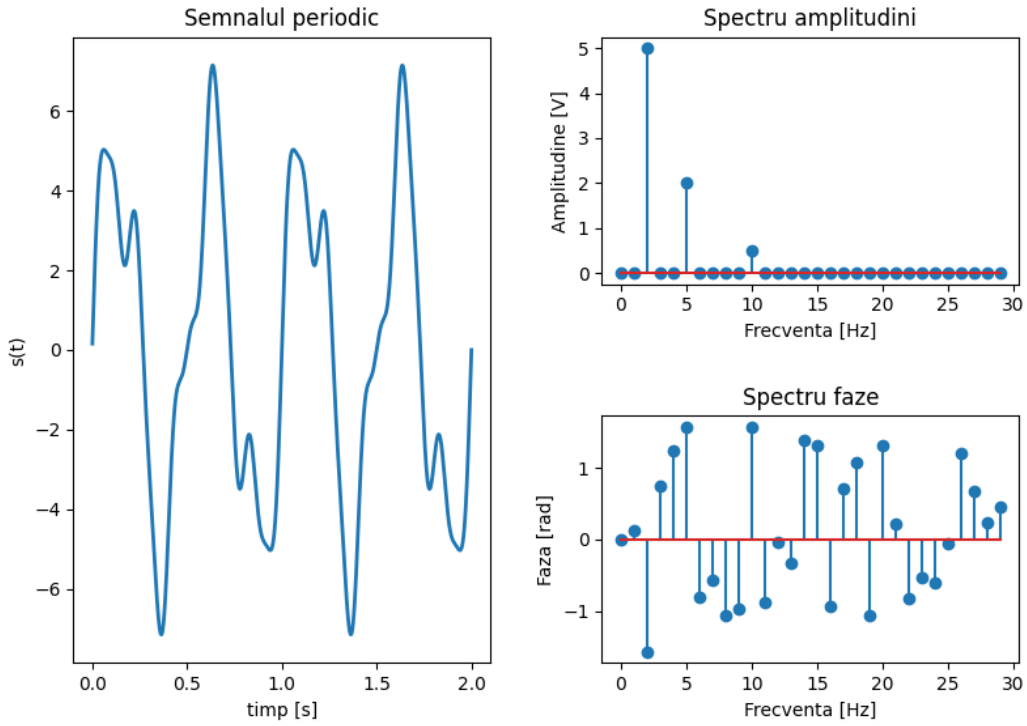

Figura 2. Spectrele de amplitudini și fază generate de seria Fourier

1.3 Semnale discrete

Semnalele discrete sunt reprezentate prin secvențe ordonate de numere. Ele se deosebesc de semnalele continue prin faptul că iau valori doar la anumite momente de timp. Acest tip de semnale pot fi notate în mai multe moduri:

$$\{x(nT)\} = x(n) = x(nT) = x[n] \tag{14}$$

unde T reprezintă perioada de eșantionare.

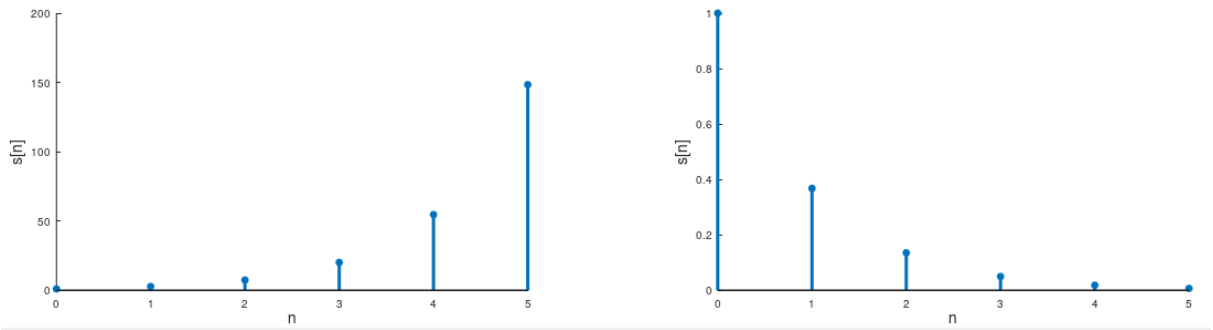

Figura 3. Exemple de semnale discrete

Secvențele s[n] pot fi finite sau infinite, periodice sau aperiodice. O secvență periodică este complet definită dacă se dă o perioadă a sa. Spunem că două secvențe sunt egale atunci când acestea au aceeași lungime, iar valorile elementelor lor sunt egale.

Asupra semnalelor discrete pot fi efectuate mai multe operații. Acestea sunt prezentate în tabelul de mai jos.

Tabelul 1. Operații cu semnale discrete

Denumirea operației	**Operația**
Adunare	$z[n] = x[n] + y[n]$
Înmulțire cu un scalar	$z[n] = ax[n]$
Deplasare în timp	$z[n] = x[n \pm n_0]$
Înmulțire	$z[n] = x[n]y[n]$
Operații liniare	$z[n] = ax[n] + by[n]$

Se disting următoarele secvențe care sunt utile în operațiile cu semnale discrete:

a. Secvența impuls unitate

$$\delta[n] = \begin{cases} 1, n = 0 \\ 0, altfel \end{cases} \tag{15}$$

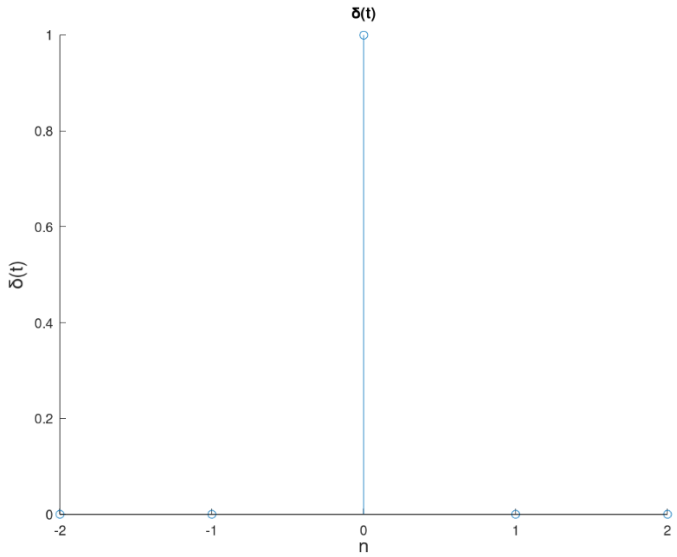

Figura 4. Secvența impuls unitate

b. Secvența impuls unitate întârziat

$$\delta[n-k] = \begin{cases} 1, n = k \\ 0, altfel \end{cases} \quad (16)$$

c. Secvența treaptă unitate

$$u[n] = \begin{cases} 1, n \geq 0 \\ 0, altfel \end{cases} \quad (17)$$

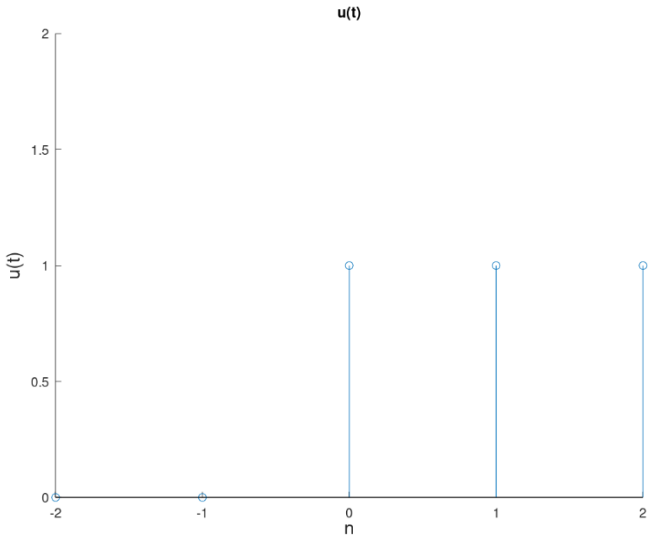

Figura 5. Secvența treaptă unitate

1.4 Cuantificarea și eșantionarea semnalelor

Deoarece sistemele digitale nu pot prelucra semnale continue, acestea trebuie transformate în semnale discrete. Această transformare poartă denumirea de *conversie analogic-numerică* (CAN) și se realizează prin operațiile de eșantionare și cuantizare.

Eșantionarea reprezintă evaluarea valorii unui semnal continuu la anumite intervale de timp. Pasul de evaluare poate fi uniform (cu interval fix de timp între eșantioane) sau neuniform (cu interval variabil de timp între eșantioane). În general se folosește eșantionarea cu pas uniform. Pasul de eșantionare poartă denumirea de perioadă de eșantionare (*T*). Frecvența de eșantionare reprezintă inversul perioadei de eșantionare.

Pentru a garanta refacerea corectă a semnalului eșantionat, frecvența de eșantionare trebuie să respecte **teorema Nyquist-Shannon**:

$$f_e \geq 2f_s \quad (18)$$

unde f_e este frecvența de eșantionare, iar f_s este frecvența semnalului. Mai exact, frecvența de eșantionare trebuie să fie cel puțin dublă față de frecvența semnalului eșantionat.

Codul Python de mai jos și Figura 6 ilustrează importanța alegerii corecte a frecvenței de eșantionare:

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

#Definire semnal
t=np.linspace(0,2,2000)
s=2*np.cos(2*np.pi*2*t)

#Definire perioade de esantionare (ms)
T1=400
T2=250
T3=100

#Definire semnale esantionare
u1=np.zeros((len(s)))
u2=np.zeros((len(s)))
u3=np.zeros((len(s)))

for i in range(1,len(s)//T1): #Se pastreaza doar partea intreaga
   u1[i*T1]=1

for i in range(1,len(s)//T2):
   u2[i*T2]=1

for i in range(1,len(s)//T3):
   u3[i*T3]=1

#Esantionarea
e1=u1*s
e2=u2*s
e3=u3*s

#Afisare
plt.subplot(3,1,1)
plt.plot(t,s,'--')
plt.title("$T_e$=400ms")
plt.xlabel("timp [s]")
plt.ylabel("s(t)")
plt.xticks(np.arange(0, 2.1, 0.1)) #marcheaza axa Ox incepand de la 0 la 2.1
cu pas de 0.1 (2.1 nu este reprezentat)
plt.plot(t,e1)

plt.subplot(3,1,2)
plt.plot(t,s,'--')
plt.title("$T_e$=250ms")
plt.xlabel("timp [s]")
plt.ylabel("s(t)")
plt.xticks(np.arange(0, 2.1, 0.1))
plt.plot(t,e2)

plt.subplot(3,1,3)
plt.plot(t,s,'--')
plt.title("$T_e$=100ms")
```

```
plt.xlabel("timp [s]")
plt.ylabel("s(t)")
plt.xticks(np.arange(0, 2.1, 0.1))
plt.plot(t,e3)

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

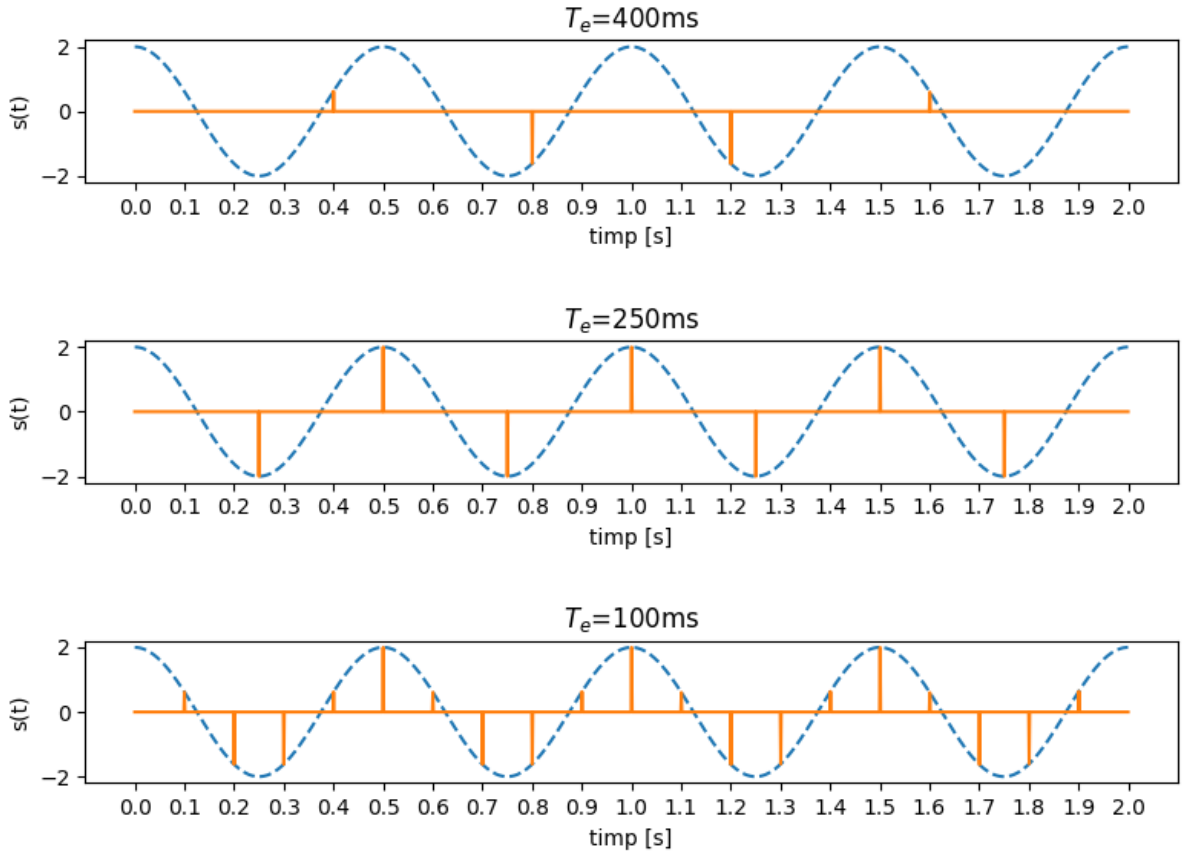

Figura 6. Eșantionarea unui semnal de 2 Hz cu diferite frecvențe de eșantionare

Întrucât un semnal analogic ia o infinitate de valori, dar sistemele de calcul pot reprezenta doar un număr finit de valori, este nevoie ca semnalul analogic să fie adaptat atunci când este digitalizat. Această operație de reprezentare a unei cantități continue pe un suport discret de biți poartă numele de **cuantificare**.

Numărul de valori ce poate fi cuantificat este dat de numărul de biți disponibili pentru reprezentare. Variația minimă a semnalului ce poate fi înregistrată de sistemul digital poartă denumirea de **rezoluție** sau pas de cuantificare. Acesta se calculează astfel:

$$Q_S = \frac{V_{MAX} - V_{min}}{2^N - 1} \tag{19}$$

unde V_{MAX} este valoarea maximă a semnalului, V_{min} valoarea minimă a semnalului, iar N este numărul de biți pe care se face cuantificarea.

1.5 Analiza semnalelor discrete – Transformata Fourier Discretă și Rapidă

Așa cum semnalele continue pot fi analizate prin intermediul transformatei Fourier, semnalele discrete pot fi studiate cu echivalentul discret al acesteia: *transformata Fourier discretă* (TFD). Înțelegerea acestei transformate este esențială în prelucrarea digitală a semnalelor prin faptul că dispozitivele digitale lucrează în mod exclusiv cu semnale discrete.

Fie un semnal discret de forma $x_0, x_1, x_2, \dots x_{N-1}$. Se numește transformata Fourier discretă secvența:

$$TFD\{x(n)\} = X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} = \sum_{n=0}^{N-1} x(n) W_N^{-nk} \tag{20}$$

unde k=0, 1, … N-1, iar $W_N=\exp(j2\pi/N)$ și reprezintă frecvența fundamentală.

Inversa transformatei Fourier discrete are forma:

$$TFD^{-1}\{X(k)\} = x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k) e^{j\frac{2\pi kn}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} X(k) W_N^{nk} \tag{21}$$

Transformata Fourier discretă poate fi scrisă și sub formă matriceală:

$$\begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \cdots & W_N^{-(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{-(N-1)} & W_N^{-1} & \cdots & W_N^{-(N-1)^2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix} \tag{22}$$

sau mai compact:

$$[X] = \left[\widehat{W_N}\right][x] \tag{23}$$

unde $[W_N]$ se poate obține din $[\hat{W}_N]$ prin înlocuirea termenilor W_N^{-k} cu W_N^{k}. Pentru generalitate se poate considera că termenii celor două secvențe sunt numere complexe.

În mod normal, rezolvarea sistemelor (22) respectiv (23) implica N(N-1) operații de înmulțire si același număr de operații de adunare, ceea ce înseamnă că este o problema de complexitate N^2.

Transformata Fourier rapidă (TFR) reprezintă orice algoritm care reduce număr de calcule necesar aflării TFD. În realitate există mai mulți algoritmi de acest tip optimizați pentru diferite valori ale lui N. De exemplu, dacă N este o putere a lui 2 se pot folosii algoritmi de decimare în timp, care reduc numărul de calcule la NlogN operații.

Programele de calcul matematic vin cu funcții predefinite ce realizează calculul transformatei Fourier discrete prin algoritmi TFR. Pachetul scipy conține funcția *fft(X,n)*, unde X reprezintă semnalul analizat, iar n numărul de puncte pentru care se face analiza. Dacă n nu este specificat, vectorul rezultat va avea aceleași dimensiuni ca semnalul analizat. Exemplul de mai jos și Figura 7 demonstrează o aplicație simplă a aceste funcții.

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

Fe=1000 #Frecventa de esantionare
dx=1/Fe
t=np.arange(0,2+dx,dx)
s=5*np.sin(2*np.pi*2*t)+2*np.sin(2*np.pi*5*t)+0.5*np.sin(2*np.pi*10*t)
#semnalul analizat

F=fft(s,1024) #transformata Fourier Rapida cu 1024 de puncte
M=abs(F)*2*dx   #Prelucrarea   amplitudinilor   (normarea   cu   frecventa   de
esantionare, inmultirea pentru spectrul complet)
f=Fe*(np.arange(0,31,1))/1024 #Definirea axei de frecvente de la 0 la 30 Hz


plt.plot(f,M[0:31],color='red',linewidth='3')
plt.title("Spectrul de Amplitudini calculat cu FFT")
plt.xlabel("Frecventa [Hz]")
plt.ylabel("Amplitudinea")

plt.show()
```

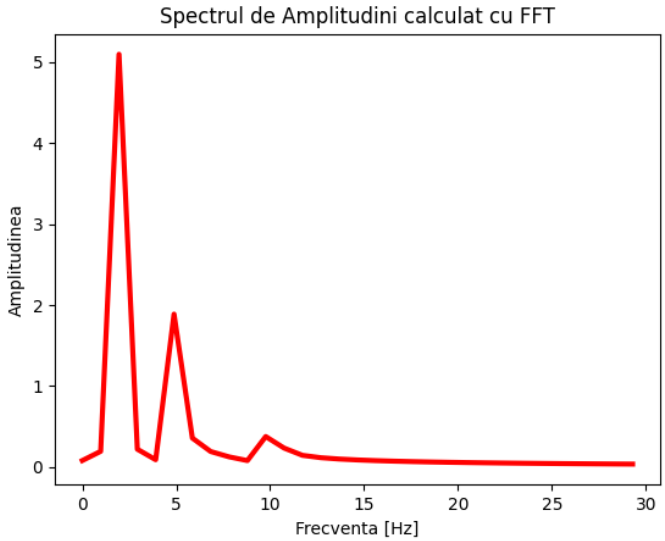

Figura 7. Spectrul de amplitudini generat prin FFT

2. Modul de lucru

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

1) Definiți și afișați un semnal dreptunghiular de amplitudine 5 și frecvență 2 Hz.
2) Definiți și afișați un semnal triunghiular de amplitudine 1,5 și frecvență 4 Hz.
3) Definiți și afișați un semnal dinte de fierăstrău de amplitudine 3 și frecvență 3 Hz.
4) Afișați cele trei semnale definite anterior într-o singură figură (unul sub celălalt)
5) Reprezentați spectrele de amplitudine și fază pentru semnalul dreptunghiular.
6) Reprezentați spectrele de amplitudine și fază pentru semnalul triunghiular.
7) Eșantionați semnalul dreptunghiular și reprezentați grafic această operație. – Țineți cont de frecvența minimă de eșantionare necesară)
8) Calculați și reprezentați grafic spectrul de amplitudini al semnalului triunghiular folosind transformata Fourier discretă.

3. Lucrarea 2 în MATLAB/Octave

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

Pentru a putea realiza operații cu semnale în Octave este nevoie, ca înainte de scrierea aplicațiilor, să fie instalat și încărcat pachetul ce conține mai multe funcții utile. Mai întâi se verifică dacă pachetul este deja instalat prin comanda:

```
pkg list
```

Fereastra de comandă va fi populată cu o listă a tuturor pachetelor instalate. Pentru aplicațiile din această lucrare de laborator căutăm pachetul *signal*. Dacă acesta apare listat, putem trece la încărcarea acestuia. În caz contrar, instalăm pachetul prin comanda:

```
pkg install -forge signal
```

După instalare pachetul poate fi utilizat încărcând-l prin comanda:

```
pkg load signal
```

3.1 Generarea semnalelor periodice

```
t=0:0.001:1; %definire timp
f=3; %frecventa semnalului
s=sin(2*pi*f*t); %semnal sinusoidal
c=cos(2*pi*f*t); %semnal cosinusoidal
p=square(2*pi*f*t); %semnal dreptunghiular
v=sawtooth(2*pi*f*t,0.5); %semnal triunghiular

subplot(4,1,1) %figura cu 4 linii si o coloana. Se reprezinta graficul 1
plot(t,s,'linewidth',2)
title('Semnal sinusoidal')
xlabel('timp [s]')
ylabel('s(t)')
subplot(4,1,2) %figura cu 4 linii si o coloana. Se reprezinta pe graficul 2
plot(t,c,'linewidth',2)
title('Semnal cosinusoidal')
xlabel('timp [s]')
ylabel('c(t)')
subplot(4,1,3) %figura cu 4 linii si o coloana. Se reprezinta pe graficul 3
plot(t,p,'linewidth',2)
title('Semnal dreptunghiular')
xlabel('timp [s]')
ylabel('p(t)')
ylim([-1.5 1.5]) %definirea limitelor axei Oy
subplot(4,1,4) %figura cu 4 linii si o coloana. Se reprezinta pe graficul 4
plot(t,v,'linewidth',2)
title('Semnal triunghiular')
```

```
xlabel('timp [s]')
ylabel('v(t)')
```

3.2 Analiza Fourier

```
clear; %sterge toate variabilele declarate pana acum
close all; %inchide toate ferestrele deschise
clc; %curata fereastra de comanda

%Definire semnal
dt=0.001;
L=2;
t=(0+dt:dt:2)*L; %interval pe care se reprezinta semnalul
s=5*sin(2*pi*2*t)+2*sin(2*pi*5*t)+0.5*sin(2*pi*10*t); %semnal periodic

%Reprezentarea in timp a semnalului
subplot(1,2,1)
plot(t,s,'linewidth',2)
title('Semnalul periodic')
xlabel('timp [s]')
ylabel('s(t)')

%Calcularea coeficientilor Fourier
for n=1:30
A(n)=sum(s.*cos(2*pi*n*t))*dt;
B(n)=sum(s.*sin(2*pi*n*t))*dt;
M(n)=sqrt(A(n)^2+B(n)^2);
P(n)=-atan(B(n)/A(n));
end

%Reprezentarea spectrelor
subplot(2,2,2)
stem(1:30,M)
title('Spectru amplitudini')
xlabel('Frecventa [Hz]')
ylabel('Amplitudine [V]')
subplot(2,2,4)
stem(1:30,P)
title('Spectru faze')
xlabel('Frecventa [Hz]')
ylabel('Faza [rad]')
```

3.2 Eșantionare

```
clear;
close all;
clc;

t=linspace(0,2,2000);
s=2*cos(2*pi*2*t);
%Definire perioade de esantionare
T1=400;
T2=250;
T3=100;

%Definire semnale de esantionare
u1=zeros(1,length(s));
u2=zeros(1,length(s));
u3=zeros(1,length(s));
for i=1:length(s)/T1
```

```
u1(i*T1)=1;
end
for i=1:length(s)/T2
u2(i*T2)=1;
end
for i=1:length(s)/T3
u3(i*T3)=1;
end

%Esantionare
e1=u1.*s; %Inmulteste fiecare element din vectorul u1 cu elementul de pe acelasi
loc din vectorul s
e2=u2.*s;
e3=u3.*s;

%Reprezentare grafica
subplot(3,1,1)
plot(t,s,'--')
title('T_e=400 ms')
xlabel('timp [s]')
ylabel('s(t)')
xticks([0:0.1:2]) %Marcheaza pe axa Ox valoriile cu pas de 0.1
hold on
plot(t,e1)
subplot(3,1,2)
plot(t,s,'--')
title('T_e=250 ms')
xlabel('timp [s]')
ylabel('s(t)')
xticks([0:0.1:2])
hold on
plot(t,e2)
subplot(3,1,3)
plot(t,s,'--')
title('T_e=100 ms')
xlabel('timp [s]')
ylabel('s(t)')
xticks([0:0.1:2])
hold on
plot(t,e3)
```

3.3 Analiză FFT

```
clear all
clc
close all

Fe=1000; %Frecventa de esantionare
dx=1/Fe;
t=[0:dx:2];
s=5*sin(2*pi*2*t)+2*sin(2*pi*5*t)+0.5*sin(2*pi*10*t);

F=fft(s,1024); %Transformata Fourier Rapida cu 1024 de puncte
M=abs(F)*2*dx; %Prelucrarea amplitudinilor
f=Fe*(0:30)/1024; %Definirea axei frecventelor de la 0 la 30 Hz

plot(f,M(1:31),'linewidth',3,'color','red')
title('Spectrul de Amplitudini calculat cu FFT')
xlabel('Frecventa [Hz]')
ylabel('Ampltudinea')
```