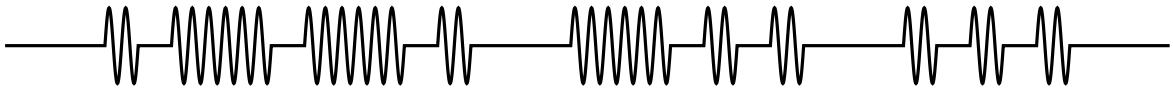
KERTÉSZ Csaba Zoltán

PRELUCRAREA DIGITALĂ A SEMNALELOR

ÎNDRUMAR DE LABORATOR



Editura
Universității
Transilvania
din Brașov
2025

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr. 41A
500091 Brașov
Tel.: 0268 476 050
Fax: 0268 476 051
E-mail: editura@unitbv.ro

Editură recunoscută CNCSIS, cod 81

ISBN 978-606-19-1819-5 (ebook)

Lucrarea a fost avizată de Consiliul Departamentului de Electronică și Calculatoare, Facultatea de Inginerie Electrică și Știința Calculatoarelor a Universității Transilvania din Brașov.

Cuvânt înainte

Prelucrarea digitală a semnalelor stă la baza nenumăratelor tehnologii curente care definesc viața noastră de zi cu zi, de la telefoane mobile, prin dispozitive biomedicale și componentele autovehiculelor, până la sisteme de comunicații, toate recurg într-un fel sau altul la prelucrarea semnalelor. Provenind din niște concepții matematice abstracte, bazate pe analiza Fourier, calcul operațional și sisteme de ecuații liniare, din secolele XVII-XVIII, mult înaintea descoperirii electromagnetismului, prelucrarea semnalelor a devenit cât se poate de practică în tehnologia electronică modernă.

Odată cu răspândirea calculatoarelor electronice, și prelucrarea semnalelor a fost supusă unei transformări radicale de la continuu la discret, de la analiza semnalelor din lumea analogică la reprezentarea digitală și prelucrarea în calculator. Mai ales după inventarea transformatei Fourier rapide și a microprocesoarelor din ce în ce mai puternice, prelucrarea digitală a semnalelor a devenit mult mai facilă decât prelucrarea analogică. Sistemele cu calculator încorporat care interacționează cu semnalele analogice din lumea reală prin discretizarea lor și prelucrarea în interiorul procesorului sunt acum mult mai ieftine, mult mai mici și mult mai ușor de dezvoltat (prin prisma implementării, testării și depanării unor algoritmi software) decât corespondentul analogic.

În lumea digitală de astăzi, folosirea tehnicilor de prelucrare digitală a semnalelor a devenit indispensabilă. Orice sistem electronic care interacționează cu lumea reală are nevoie de aceste tehnici, fie că vorbim despre achiziția datelor, prelucrarea și controlul în timp real, îmbunătățirea semnalelor audio/video sau comunicații radio. De aceea, Prelucrarea Digitală a Semnalelor, ca disciplină de sine stătătoare, face parte din orice curriculă educațională în domeniul Inginerie Electronică și Telecomunicații.

Acest îndrumar se adresează în primul rând studenților de la programele de studii din acest domeniu, Electronică Aplicată și Tehnologii și Sisteme de Telecomunicații, dar și din domeniile conexe, de exemplu, Calculatoare, Tehnologia Informației, Inginerie Electrică, etc. La programele de studii Electronică Aplicată, respectiv Tehnologii și Sisteme de Telecomunicații din cadrul Facultății de Inginerie Electrică și Știința Calculatoarelor a Universității Transilvania din Brașov, disciplina Prelucrarea Digitală a Semnalelor este o disciplină impusă în anul 3 de studii și urmărește familiarizarea studenților cu sisteme discrete liniare și invariante în timp, cu reprezentarea discretă a semnalelor în timp și în spectru, cu filtre digitale FIR și IIR, și nu în ultimul rând cu diferite aplicații practice ale acestor operații.

Îndrumarul este o reeditare a vechiului îndrumar din anul 2009, conținutul fiind adaptat la schimbările din planul de învățământ petrecute între timp. Cea mai importantă diferență în planul de învățământ este că studenții de anul 3 sunt deja familiarizați cu noțiunile introductive de prelucrare digitală a semnalelor prin intermediul laboratorului aferent cursului de Prelucrare a Semnalelor, curs care urmărește mai ales fundamentele matematice ale prelucrării semnalelor, dar în cadrul laboratorului se folosește calculatorul pentru a rezolva aceste probleme. Astfel, cei care parcurg acest îndrumar sunt deja familiarizați cu Matlab și/sau Octave, toolul care este folosit și în cadrul acestui îndrumar pentru implementarea algoritmilor de prelucrare digitală a semnalelor studiați.

Îndrumarul poate fi împărțit în două părți: prima parte conține noțiunile de bază, iar a doua parte niște aplicații mai practice. În prima parte sunt prezentate aspectele particulare ce țin de prelucrarea în digital, discretizarea semnalelor și reprezentarea lor pe calculator, transformata Fourier discretă și transformata z, implementarea rapidă a transformatei Fourier, respectiv proiectarea și implementarea filtrelor digitale. Accentul este pus pe problemele specifice introduse de prelucrarea digitală, compromisurile pe care trebuie să le ia inginerii și diferențele principale față de prelucrarea analogică.

A doua parte a îndrumarului se concentrează asupra unor aplicații practice. Sunt prezentate aplicații pentru semnale DTMF, semnale audio și analiza spațio-temporală a semnalelor. Acestea au ca scop oferirea unui fundament pentru viitoarele discipline de specialitate în care se vor aprofunda studenții de inginerie electronică și telecomunicații. Studenții pot folosi aceste fundamente pentru a aprofunda mai bine în materiile Telefonie, Sisteme multimedia, Radiocomunicații, Sisteme încorporate și nu numai.

Cuprins

Lucrarea 1

Reprezentarea semnalelor în Matlab

1.1. Semnale analogice și prelucrarea digitală

Semnalele analogice sunt semnale continue ce pot fi reprezentate din punct de vedere matematic ca funcții reale continue în timp. Pentru a putea prelucra un semnal cu ajutorul unui calculator — indiferent că se folosește un procesor de uz general sau un procesor dedicat prelucrărilor de semnal (DSP – Digital Signal Processor) — semnalele continue trebuie discretizate într-un format potrivit pentru reprezentarea numerică dintr-un procesor.

Semnalul discretizat este reprezentat ca o secvență finită de numere cu valori discrete. Pentru conversia unui semnal analogic într-un semnal digital se folosește un convertor analog-digital (ADC), care realizează două operații fundamentale: discretizare în timp, adică **eșantionare** și discretizare în amplitudine, adică **cuantizare**. Convertorul analog-digital, prezentat pe figura 1.1, este alcătuit dintr-un circuit de eșantionare-memorare (SH - Sample and Hold), care prelevă semnalul de la intrare la momente discrete de timp și memorează valoarea pe parcursul conversiei până la următoarea eșantionare și un circuit de cuantizare care convertește tensiunea analogică de la intrare într-un număr binar pe N biți.

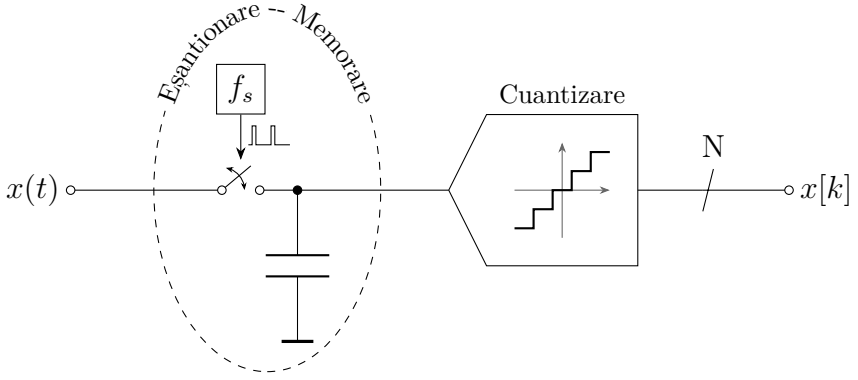


Figura 1.1. *Discretizarea semnalelor analogice*

Eșantionarea

Eșantionarea reprezintă discretizarea în timp a unui semnal continuu și este realizată prin prelevarea amplitudinii semnalului la momente discrete de timp, așa cum e demonstrat pe figura 1.2.

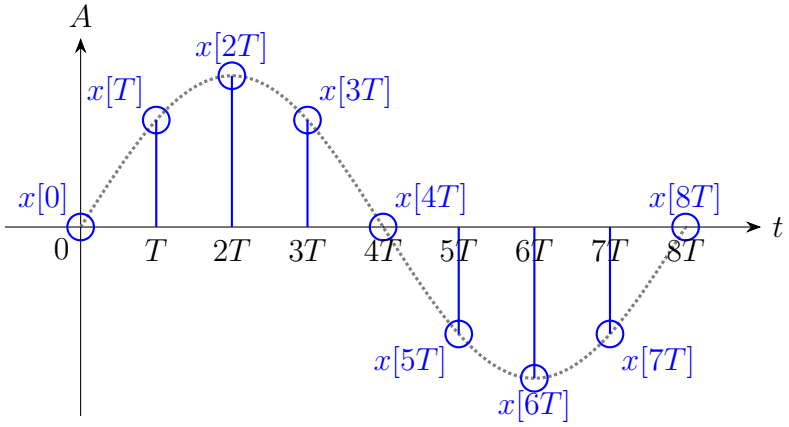


Figura 1.2. *Eșantionarea unui semnal sinusoidal*

Rezultatul eșantionării uniforme a semnalului continuu $x(t)$ este un șir de eșantioane $x[nT]$, unde T este intervalul de timp între două eșantioane și se numește perioadă de eșantionare.

Frecvența de eșantionare $f_s = \frac{1}{T}$ reprezintă frecvența cu care sunt prelevate eșantioanele semnalului analogic. Se mai numește și rata eșantionării.

Conform teoremei eșantionării, un semnal poate fi refăcut complet din eșantioanele sale, dacă este satisfăcut criteriul Nyquist și anume frecvența de eșantionare este cel puțin dublul frecvenței maxime f_m din spectrul semnalului:

$$f_s \geq 2 \cdot f_m \tag{1.1}$$

Bineînțeles, spectrul semnalului trebuie să fie mărginit, de forma celui din figura 1.3, ca să putem considera frecvența maximă. Alegerea frecvenței de eșantionare este detaliată mai mult la lucrarea 7.

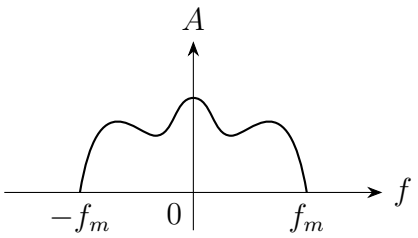


Figura 1.3. *Spectrul unui semnal mărginit*

Reprezentarea semnalului eșantionat în Matlab poate fi făcută sub forma unui vector, a cărui elemente sunt eșantioanele semnalului $x[n]$ prelevate la momentele de timp date de frecvența de eșantionare. Informația despre frecvența de eșantionare în sine însă nu este stocată în niciun fel în acest vector.

Dacă avem un semnal sinusoidal de forma $A\sin(2\pi f_0 t)$ și eșantionăm la frecvența de eșantionare $f_s = 8f_0$ obținem eșantioanele $[0, \frac{A}{\sqrt{2}}, A, \frac{A}{\sqrt{2}}, 0, -\frac{A}{\sqrt{2}}, -A, -\frac{A}{\sqrt{2}}, 0]$, similar cu semnalul eșantionat de pe figura 1.2. Exact același vector de eșantionare obținem însă și atunci când eșantionăm un alt semnal sinusoidal cu frecvența f_1 dacă alegem corespunzător și o altă frecvență de eșantionare $f_s' = 8f_1$.

Acest lucru înseamnă că prelucrarea digitală a semnalelor se poate realiza prin niște operații matematice asupra unor vectori, cu condiția ca la operațiile care fac referire la frecvența sau perioada semnalului, trebuie să știm, pe lângă valorile eșantioanelor din vector, și frecvența de eșantionare folosită pentru obținerea acelui vector. De multe ori, pentru operațiile asupra vectorilor este suficient și folosirea unor frecvențe relative, în care componentele de frecvență le exprimăm adimensional raportat la jumătatea frecvenței de eșantionare (frecvența Nyquist) indiferent de care ar fi aceasta.

De exemplu, considerând un semnal sinusoidal cu frecvența de 1000 Hz și amplitudinea de 1 V, pe o durată de 1 ms și folosind frecvența de eșantionare de 8 kHz vom obține următoarele eșantioane:

```
>> fs = 8e3;           % frecvența de eșantionare
>> dt = 1e-3;          % durata semnalului eșantionat
>> t = 0 : 1/fs : dt; % momentele de timp de prelevare a semnalului
>> f = 1000;          % frecvența semnalului
>> A = 1;             % amplitudinea semnalului
>> s = A * sin (2 * pi * f * t); % eșantioanele semnalului
```

afișând valorile în linia de comandă:

```
>> s
   s =
       0  0.7071  1.0000  0.7071  0.0000 -0.7071 -1.0000 -0.7071 -0.0000
```

Exact același vector obținem și atunci când specificăm un semnal cu o frecvență relativă ($\frac{1}{8}$ din frecvența de eșantionare, fără a preciza însă valorile absolute ale acestor frecvențe):

```
>> sin(2 * pi * 1/8 * (0:8))
   ans =
       0  0.7071  1.0000  0.7071  0.0000 -0.7071 -1.0000 -0.7071 -0.0000
```

Reprezentarea vizuală a semnalului (figura 1.4) se poate obține cu comanda:

```
>> stem(t, s)
```

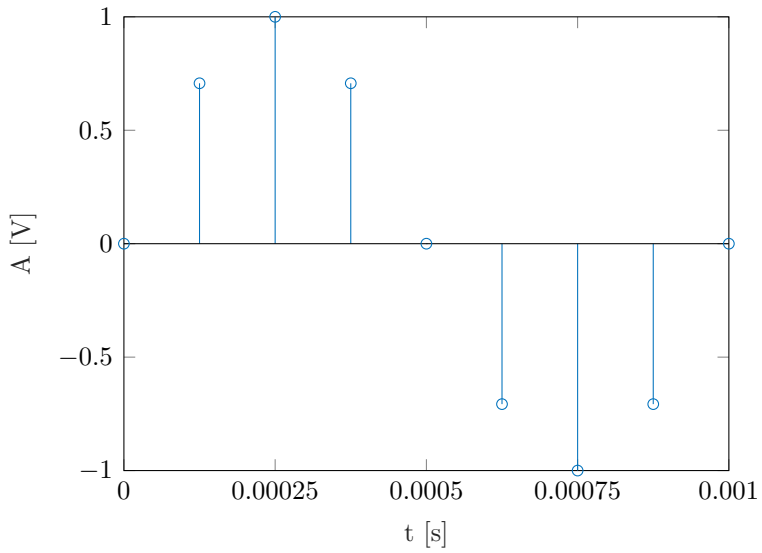


Figura 1.4. *9 eșantioane dintr-un semnal sinusoidal de 1 kHz eșantionat la 8 kHz*

Cuantizarea

Cuantizarea reprezintă discretizarea în amplitudine a valorilor eșantioanelor. Aceasta înseamnă înlocuirea valorilor eșantioanelor cu o valoare apropiată dintr-un set finit de valori posibile, numite nivele de cuantizare. Putem descrie cuantizarea cu o funcție:

$$y \leftarrow q_i, \text{pentru } d_i \leq x < d_{i+1} \tag{1.2}$$

unde $q_0 \dots q_{N-1}$ sunt nivelele de cuantizare, iar $d_0 \dots d_N$ sunt nivelele de decizie sau praguri de cuantizare, dispuse între nivelele de cuantizare vecine.

De cele multe ori nivelele de cuantizare sunt distribuite uniform între două nivele de amplitudine (minim și maxim) cu praguri exact la mijloc între două nivele de cuantizare. În acest caz, numit cuantizare uniformă, diferența între oricare două nivele de cuantizare vecine $q = q_{i+1} - q_i$ este aceeași și se numește pas de cuantizare sau cuantă. Astfel, vom avea:

$$q = \frac{A_{max} - A_{min}}{N - 1} \tag{1.3}$$

$$q_i = A_{min} + q \cdot i, \quad i = \overline{0, N} \tag{1.4}$$

$$d_i = q_i - \frac{q}{2} \quad d_{i+1} = q_i + \frac{q}{2} \tag{1.5}$$

Această cuantizare uniformă poate fi descrisă cu o funcție reprezentată grafic pe figura 1.5.

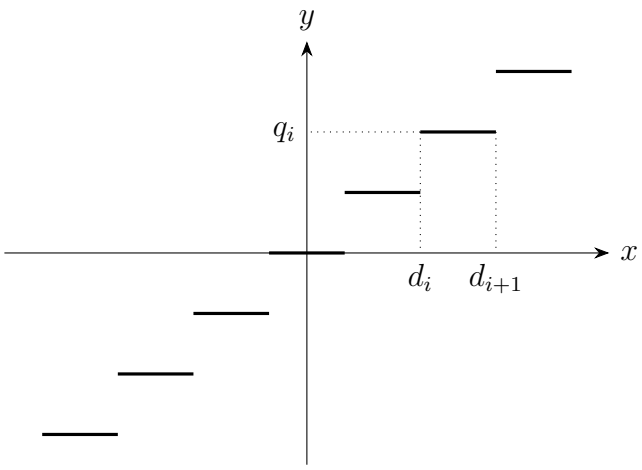


Figura 1.5. *Funcție de cuantizare uniformă*

Cuantizarea uniformă corespunde unei operații de rotunjire a semnalului de intrare către cel mai apropiat nivel de cuantizare.

Următorul exemplu arată cuantizarea unui semnal sinusoidal pe 17 nivele de cuantizare distribuite uniform pe intervalul $\pm 1\,\mathrm{V}$:

```
>>> t = 2*pi*(0:255)/256;
>>> s = sin(t);
>>> q = round(s*8)/8;
>>> plot(t,s,t,q,'.')
>>> axis([0,2*pi,-1,1])
```

Graficul rezultat se vede pe figura 1.6.

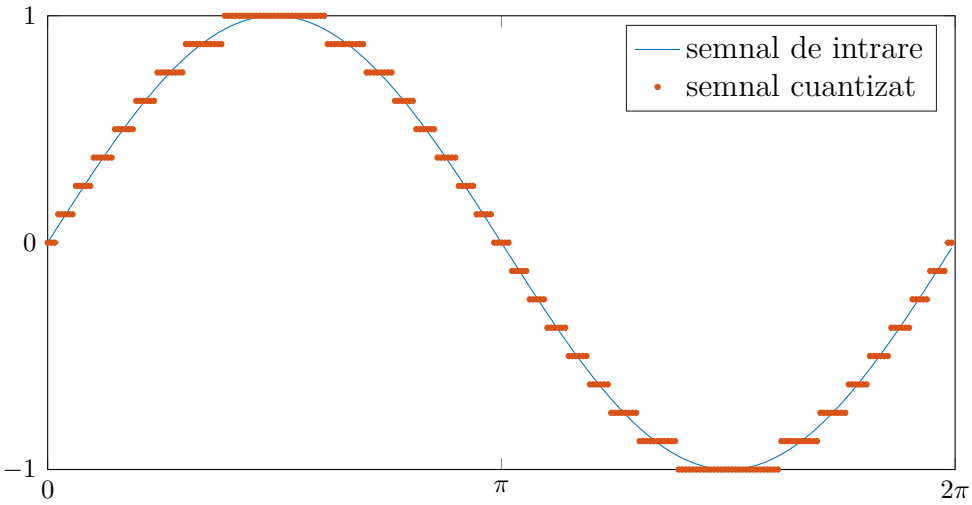


Figura 1.6. *Semnal sinusoidal cuantizat*

Pentru realizarea operației de cuantizare se folosește convertorul analog-digital (ADC). Din motive practice și pentru simplitatea implementării, aceasta convertește tensiunea analogică de la intrare într-un număr întreg binar, reprezentat pe un număr întreg de biți. Astfel numărul nivelelor de cuantizare va fi întotdeauna de forma 2^n, unde n este numărul de biți folosiți pentru reprezentare. Valoarea convertită se regăsește în intervalul $0 \dots 2^n - 1$ în cazul conversiei unipolare și $-2^{n-1} \dots 2^{n-1} - 1$ în cazul conversiei bipolare. Se poate vedea că capetele intervalului de conversie sunt incomplete pentru a putea reprezenta și valoarea 0. De aceea de obicei convertoarele analog-digitale definesc intervalul de amplitudine între A_{min} și $A_{max} - 1LSB$. Aceasta introduce o mică eroare de neliniaritate în conversie.

Impactul major asupra prelucrării semnalelor în cazul cuantizării este cauzat de faptul că funcția de rotunjire cu care operează conversia nu este bijectivă. Din această cauză, semnalul original nu mai poate fi refăcut după cuantizare, rezultând în prezența unei erori în semnalul cuantizat:

$$\varepsilon = |x_q[n] - x_s[n]| \tag{1.6}$$

unde x_s este semnalul analogic eșantionat, iar x_q este semnalul cuantizat.

Eroarea aceasta se asimilează unui zgomot aditiv introdus asupra semnalului original și se numește *zgomot de cuantizare*. Zgomotul de cuantizare pentru semnalul sinusoidal din exemplul precedent este prezentat pe figura 1.7.

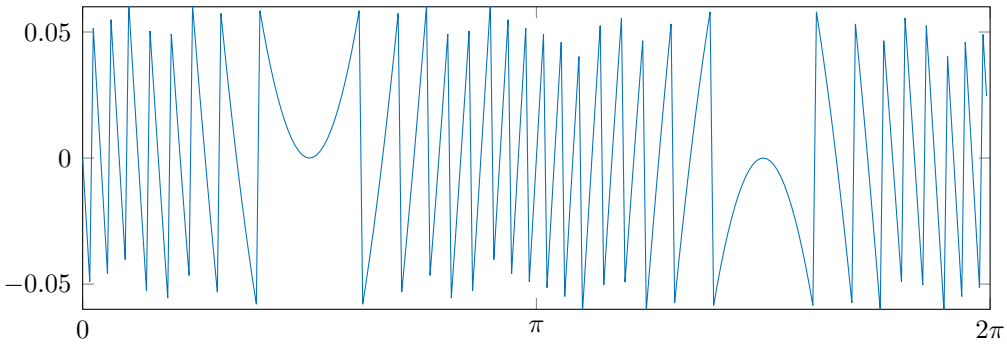


Figura 1.7. *Zgomotul de cuantizare*

Pentru caracterizarea acestui zgomot, se folosește raportul semnal-zgomot sau SNR (*signal-to-noise ratio*). Acesta este definit ca raportul între valoarea efectivă a semnalului și valoarea efectivă a zgomotului, și este de obicei exprimat în dB:

$$\mathrm{SNR} = 20 \cdot \lg \frac{V_{ef\mathrm{semnal}}}{V_{ef\mathrm{zgomot}}} \tag{1.7}$$

Valoarea efectivă a semnalului este definită ca media pătratică a semnalului și se mai notează și cu RMS (root-mean-square). Aceasta se calculează cu formula:

$$V_{ef} = \sqrt{\frac{1}{\Delta t} \int_0^{\Delta t} x^2(t)\, dt} \tag{1.8}$$

sau în cazul unui semnal discretizat:

$$V_{ef} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x^2[n]} \tag{1.9}$$

Considerând semnalul sinusoidal cu amplitudinea A pe durata unei perioade întregi, valoarea efectivă a semnalului rezultă în:

$$\begin{aligned} V_{ef\text{semnal}} &= \sqrt{\frac{1}{2\pi}\int_0^{2\pi} (A\sin x)^2\,dx} = \sqrt{\frac{A^2}{2\pi}\int_0^{2\pi}\frac{1-\cos 2x}{2}\,dx} = \\ &= \sqrt{\frac{A^2}{2\pi}\left(\frac{x}{2}\Big|_0^{2\pi} - \frac{\sin 2x}{4}\Big|_0^{2\pi}\right)} = \sqrt{\frac{A^2}{2\pi}\frac{2\pi}{2}} = \frac{A}{\sqrt{2}} \end{aligned} \tag{1.10}$$

Pentru caracterizarea valorii efective a zgomotului putem considera valoarea efectivă a diferențelor față de nivelul de cuantizare între fiecare prag de cuantizare. Considerând un convertor analog-digital bipolar și cuantizare uniformă pe n biți, vom avea $N = 2^n$ nivele de cuantizare distribuite pe intervalul $\overline{-A, A}$. Aceasta rezultă în valoarea unei cuante $q = \frac{2A}{N}$ și nivelele de cuantizare q_i respectiv pragurile de cuantizare d_i și d_{i+1} definite anterior. Din aceste valori rezultă valoarea efectivă a zgomotului:

$$\begin{aligned} V_{ef\text{zgomot}} &= \sqrt{\frac{1}{2A}\sum_{j=0}^{N-1}\int_{d_j}^{d_{j+1}} (x-q_j)^2\,dx} = \sqrt{\frac{1}{2A}\sum_{j=0}^{N-1}\frac{1}{3}(x-q_j)^3\Big|_{d_j}^{d_{j+1}}} = \\ &= \sqrt{\frac{1}{2A}\sum_{j=0}^{N-1}\frac{1}{3}\left[(d_{j+1}-q_j)^3 - (d_j-q_j)^3\right]} = \sqrt{\frac{1}{2A}\frac{1}{3}\sum_{j=0}^{N-1}\left(\frac{q}{2}\right)^3 - \left(-\frac{q}{2}\right)^3} = \\ &= \sqrt{\frac{1}{2A}\cdot\frac{1}{3}\cdot N\cdot 2\cdot\left(\frac{q}{2}\right)^3} = \sqrt{\frac{1}{A}\cdot\frac{1}{3}\cdot N\cdot\left(\frac{A}{N}\right)^3} = \sqrt{\frac{1}{3}\cdot\frac{A^2}{N^2}} = \frac{A}{N\sqrt{3}} = \frac{q}{2\sqrt{3}} \end{aligned} \tag{1.11}$$

Raportul semnal-zgomot rezultat pentru acest semnal va fi:

$$\text{SNR} = 20\cdot\lg\frac{\frac{A}{\sqrt{2}}}{\frac{A}{N\sqrt{3}}} = 20\cdot\lg 2^n\frac{\sqrt{3}}{\sqrt{2}} \approx 6,02\cdot n + 1,76 \tag{1.12}$$

Din această relație putem observa că zgomotul de cuantizare nu depinde de caracteristicile proprii ale semnalului (amplitudine și frecvență), nici de frecvența de eșantionare, ci doar de numărul de biți folosiți la cuantizarea semnalului.

1.2. Exerciții

1. Reprezentați grafic 1024 de eșantioane ale unui semnal alcătuit din 2 sinusoide (una de frecvența de 220 Hz și amplitudinea de 0.5 V, iar cealaltă de frecvența de 50 Hz și amplitudinea de 0.4 V, defazat față de primul cu $\frac{\pi}{4}$) folosind o frecvență de eșantionare de 8 kHz.

2. Cuantizați acest semnal pe 8 respectiv, 16 biți, reprezentați semnalul cuantizat alături de zgomotul de cuantizare și calculați media pătratică a zgomotului și raportul semnal-zgomot.

Lucrarea 2

Analiza spectrală a semnalelor

Spectrul semnalelor sau densitatea spectrală de putere a unei funcții $x(t)$ reprezentând semnalul fizic este o reprezentare a distribuției puterii în componente cu frecvențe discrete sau continue pe un interval de frecvențe. Conform analizei Fourier, orice semnal fizic poate fi descompus într-un număr de sinusoide cu frecvențe distincte (discrete sau continue pe un interval). Pentru analiza semnalelor, puterea acestor sinusoide poate fi reprezentată grafic în funcție de frecvențele lor.

Prin analiza spectrală putem identifica vizual componentele sinusoidale principale care alcătuiesc semnalul. Pe figura 2.1 se poate vedea o captură de ecran de pe un analizor spectral Tektronix SA2600, care arată banda radio FM (88-108 MHz). Se pot observa componentele spectrale asociate unor posturi radio prezente în zona de măsurare.

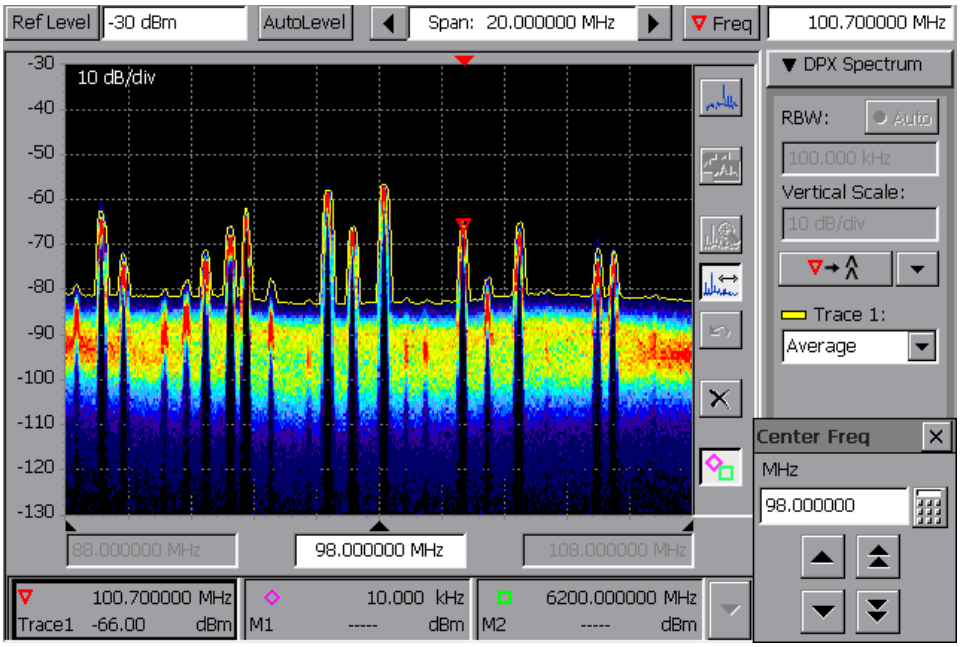


Figura 2.1. *Ecranul unui analizor spectral vizualizând banda radio FM*

2.1. Transformata Fourier Discretă

Pentru a calcula spectrului unui semnal se folosește transformata Fourier definită prin ecuația:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t}\, dt \tag{2.1}$$

Funcția $x(t)$, care definește semnalul analizat, trebuie să fie integrabilă în modul, adică integrala modulului funcției să existe și să fie finită:

$$\int_{-\infty}^{+\infty} |x(t)|\, dt < \infty \tag{2.2}$$

Pentru a calcula integrala (mai ales în cazul funcțiilor periodice) putem considera rezultatul ca fiind o funcție de distribuție, inclusiv impulsuri Dirac. De exemplu, pentru un semnal cosinusoidal $x(t) = cos(\omega_0 t)$ vom avea transformata Fourier:

$$\begin{aligned}
X(\omega) &= \int_{-\infty}^{+\infty} cos(\omega_0 t) e^{-j\omega t}\, dt \\
&= \int_{-\infty}^{+\infty} \left[\frac{e^{j\omega_0 t} + e^{-j\omega_0 t}}{2}\right] e^{-j\omega t}\, dt \\
&= \frac{1}{2} \int_{-\infty}^{+\infty} \left[e^{j\omega_0 t} e^{-j\omega t} + e^{-j\omega_0 t} e^{-j\omega t}\right] dt \\
&= \frac{1}{2} \left[\int_{-\infty}^{+\infty} e^{-j(\omega-\omega_0)t}\, dt + \int_{-\infty}^{+\infty} e^{-j(\omega+\omega_0)t}\, dt\right] \\
&= \frac{1}{2} \left[2\pi\delta(\omega-\omega_0) + 2\pi\delta(\omega+\omega_0)\right] \\
&= \pi\delta(\omega-\omega_0) + \pi\delta(\omega+\omega_0)
\end{aligned} \tag{2.3}$$

ceea ce se prezintă grafic (figura 2.2) ca două impulsuri Dirac așezate pe frecvența proprie a sinusoidei (ω_0) și pe frecvența imaginară ($-\omega_0$). Trebuie notat faptul că transformata Fourier este definită pe funcții complexe, rezultând apariția unor frecvențe imaginare (frecvențe negative) în domeniul transformat. În cazul unor semnale reale în timp, componentele imaginare din domeniul transformat vor fi exact o imagine în oglindă a componentelor reale.

Este de remarcat și faptul că, de obicei, dacă vorbim despre spectru, ne referim la spectrul de magnitudine a semnalului. Dat fiind faptul că rezultatul transformării este

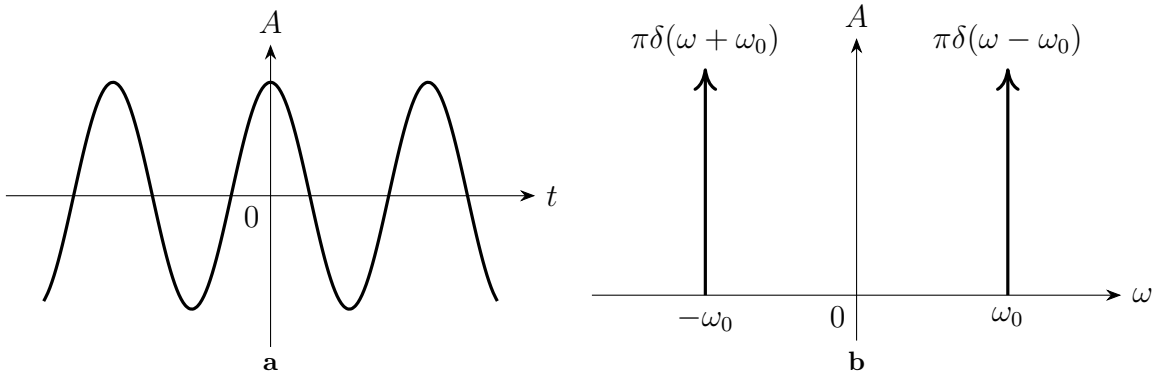


Figura 2.2. *Semnal cosinusoidal a și spectrul Fourier b*

o funcție complexă, aceasta poate fi definită în forma polară (ec. 2.4) cu magnitudinea și faza spectrală dată de modulul ($|X(\omega)|$) și argumentul ($\theta(\omega)$) sinusoidelor complexe în care este descompus semnalul.

$$X(\omega) = |X(\omega)|e^{j\theta(\omega)} \tag{2.4}$$

Transformata Fourier este o operație reversibilă, semnalul original poate fi calculat din spectrul său prin transformata Fourier inversă:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) e^{+j\omega t}\, d\omega \tag{2.5}$$

Calcularea spectrului Fourier folosind un calculator presupune discretizarea operației de transformare. Astfel, dacă semnalul în timp este un semnal eșantionat, vom avea transformata Fourier de timp discret (DTFT – Discrete Time Fourier Transform) de forma:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} \tag{2.6}$$

Discretizând operația DTFT și în domeniul transformat obținem transformata Fourier discretă (DFT – Discrete Fourier Transform) de forma:

$$X[k] = X(e^{j\omega}) \Big|_{\omega=2\pi\frac{k}{N}} = \sum_{n=0}^{N-1} x[n] e^{-j2\pi\frac{kn}{N}}, \quad k = \overline{0, N-1} \tag{2.7}$$

Transformata Fourier inversă poate fi discretizată la fel, obținând transformata Fourier discretă inversă (IDFT – Inversed Discrete Fourier Transform):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi\frac{kn}{N}}, \quad n = \overline{0, N-1} \tag{2.8}$$

În mod uzual, pentru realizarea acestor operații pe un calculator, se folosește algoritmul rapid de calcul al transformatei Fourier (FFT - Fast Fourier Transform) respectiv a transformatei inverse (IFFT - Inverse Fast Fourier Transform) discutat în detaliu în Lucrarea 3. Astfel, pentru a calcula spectrul unui semnal în Matlab, vom folosi funcția `fft`, iar pentru operația inversă funcția `ifft`.

De exemplu, pornind de la un semnal sinusoidal:

```
>> t = 0 : 1/8000 : 1023/8000;
>> x = sin(2*pi*500*t);
```

putem calcula transformata Fourier discretă cu comanda:

```
>> X = fft(x);
```

Deoarece transformata Fourier este definită în domeniul complex, chiar dacă avem un semnal real la intrare, semnalul din domeniul transformat va fi alcătuit din valori complexe. Aceste valori nu pot fi afișate direct pe un grafic bidimensional, trebuie afișată separat partea reală și partea imaginară, așa cum se vede pe figura 2.3.

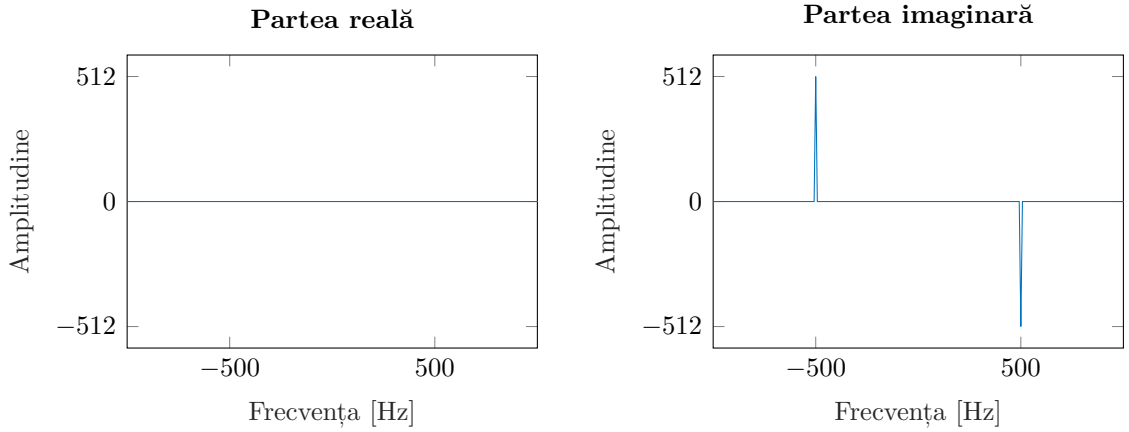


Figura 2.3. *Spectrul Fourier a unui semnal sinusoidal de 500Hz (partea reală și partea imaginară)*

Interpretarea acestora poate să prezinte o problemă însă, energia semnalului regăsindu-se în componenta reală sau imaginară în funcție de faza sinusoidei. De aceea, o abordare mai bună este folosirea formei trigonometrice a valorilor complexe, care oferă direct informația despre amplitudinea (prin modulul numărului complex) respectiv faza (prin argumentul numărului complex) a sinusoidelor care alcătuiesc semnalul analizat. Modulul unui număr complex se poate calcula cu funcția `abs`, iar argumentul cu funcția `angle`. Aceasta din urmă rezultă în faza definită în intervalul $\pm\pi$ și poate suferi de salturi vizuale la trecerile lângă π, de aceea, de multe ori e de preferat să fie desfășurat într-o funcție continuă prin `unwrap`.

Așadar putem afișa spectrul semnalului obținut mai sus prin:

```
>> w = 0 : 8000/1024 : 8000*1023/1024;
>> subplot(2,1,1),plot(w,abs(X)),subplot(2,1,2),plot(w,unwrap(angle(X)));
```

Graficul rezultat din aceste comenzi este prezentat în Figura 2.4.

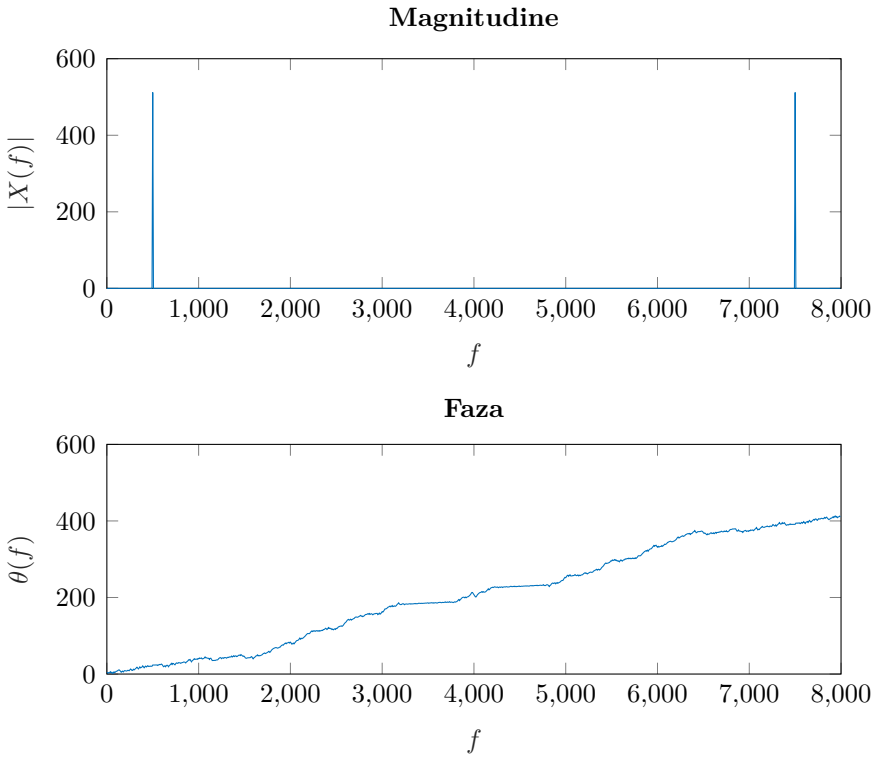


Figura 2.4. *Magnitudinea și faza spectrală a unui semnal sinusoidal de 500 Hz*

Trebuie să notăm două aspecte particulare ce rezultă din modul de implementare a transformatei Fourier rapide din Matlab:

- Spectrul semnalului calculat conține același număr de elemente ca numărul de eșantioane din semnalul original organizat în felul următor: prima jumătate a componentelor spectrale indică spectrul frecvențelor între 0 și jumătatea frecvenței de eșantionare (spectrul real), iar celelalte componente indică spectrul frecvențelor negative (spectrul imaginar). În cazul semnalelor reale în domeniul timp, spectrul imaginar va fi o imagine în oglindă a spectrului real.
- Datorită naturii integrative a transformatei, magnitudinea spectrală calculată se scalează cu numărul de eșantioane. Transformata inversă (`ifft`) are grijă de acest aspect prin aplicarea unei normalizări cu $\frac{1}{N}$. Dacă vream să afișăm magnitudinile spectrale în așa fel încât să redea amplitudinile componentelor, atunci trebuie să facem manual normalizarea.

În cazul primului aspect putem aplica funcția `fftshift` pentru a realinia spectrul astfel încât componentele de frecvență negativă să fie afișate înaintea componentelor de frecvență pozitivă:

```
>> w = (-512 : 511) * 8000/1024;
>> plot(w,fftshift(abs(X)));
```

Aceasta rezultă în figura 2.5.

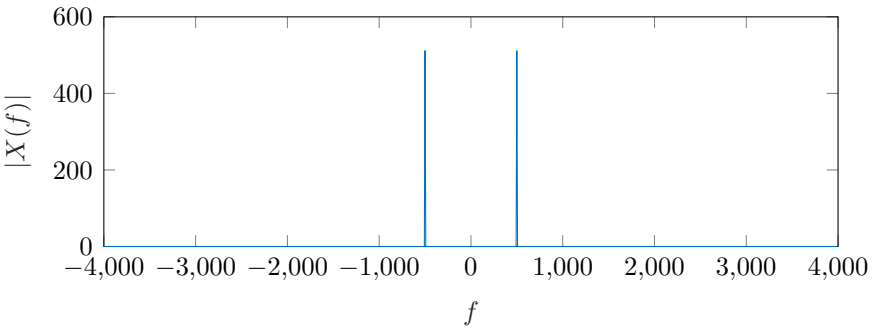


Figura 2.5. *Spectrul semnalului realiniat*

Având în vedere că în mod normal lucrăm cu semnale reale, a căror spectru imaginar este exact imaginea în oglindă a spectrului real, de multe ori nici nu ne interesează afișarea spectrului imaginar. Următorul cod afișează doar spectrul real al semnalului:

```
>> w = (0 : 511) * 8000 / 1024;
>> plot(w,abs(X)(1:512).*[1 repmat(2,1,511)]/1024);
```

În acest exemplu am realizat și normalizarea spectrului astfel încât componentele de magnitudine să aibă aceeași amplitudine ca și componentele semnalului în timp. Rezultatul se găsește pe figura 2.6.

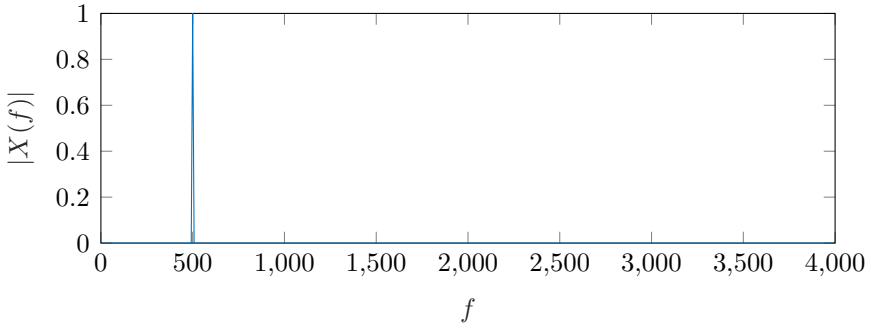


Figura 2.6. *Spectrul real a unei sinusoide de 500 Hz*

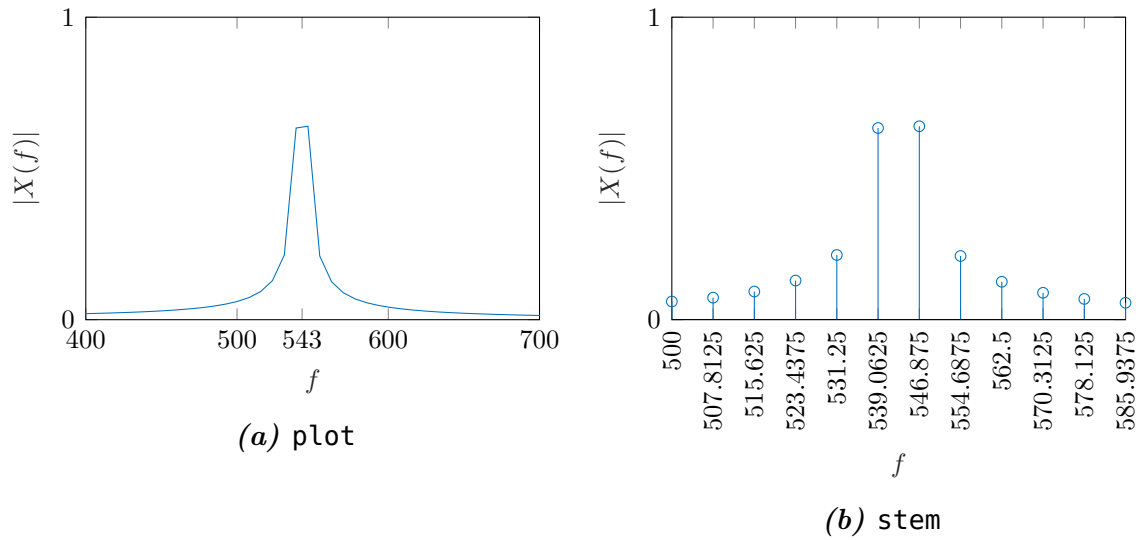
Normalizarea se face cu $\frac{1}{N}$ în cazul componentei continue, celelalte componente însă, datorită faptului că energia lor este împărțită între componenta reală și componenta imagine, au nevoie de un factor de normalizare $\frac{2}{N}$ pentru a afișa componenta reală la amplitudinea originală.

Mai există o problemă de reprezentare a spectrului datorată naturii discrete a transformatei. Dacă încercăm să afișăm spectrul unui semnal sinusoidal de exemplu de 543 Hz:

```
>> t = 0 : 1/8000 : 1023/8000;
>> x = sin(2*pi*543*t);
>> w = 0 : 8000/1024 : 8000*1023/1024;
>> plot(w,abs(fft(x))/512); axis([450 600 0 1]);
```

putem observa un grafic (figura 2.7a) care prezintă o distribuție normală în loc de impuls Dirac. Folosind funcția `stem` în loc de `plot` putem înțelege mai bine (figura 2.7b).

```
>> stem(w,abs(fft(x))/512); axis([500 585.9375 0 1]);
```



(a) `plot` *(b)* `stem`

Figura 2.7. *Spectrul unei sinusoide de 543 Hz în două moduri de afișare*

Din figura putem observa că în loc de componenta de 543 Hz, două componente mai mari la 539.0625 Hz și la 546.875 Hz și un număr de componente laterale din ce în ce mai mici.

Motivul pentru această dispersie a puterii spectrale este fenomenul numit *scurgere spectrală*. Aceasta se datorează unei ferestruiri ascunse în eșantioanele semnalului folosite pentru calculul transformatei. Practic, transformata Fourier discretă este calculată de-a lungul unui număr finit de eșantioane, tot restul semnalului fiind considerat 0.

Având N eșantioane și o frecvență de eșantionare f_s practic rezultă într-o fereastră de lungime $\frac{N}{f_s}$. Dacă socotim transformata Fourier a acestuia, rezultă într-o funcție de tip sinus cardinal:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt = \int_{0}^{\frac{N}{f_s}} e^{-j2\pi ft} = \frac{e^{-j2\pi f\frac{N}{f_s}} - 1}{-j2\pi f} = \frac{-2j\sin(\pi f\frac{N}{f_s})e^{-j\pi f\frac{N}{f_s}}}{-j2\pi f} = \frac{\sin(\pi f\frac{N}{f_s})}{\pi f\frac{N}{f_s}}\frac{N}{f_s}e^{-j\pi f\frac{N}{f_s}} \quad (2.9)$$

sau dacă ne uităm doar la magnitudine

$$|X(f)| = \frac{N}{f_s}\left|\text{sinc}(\pi f\frac{N}{f_s})\right| \quad (2.10)$$

Funcția rezultată poate fi vizualizată pe figura 2.8. Se poate observa că semiperioada funcției de sinus cardinal (trecerile prin 0) cade pe multipli lui $\frac{f_s}{N}$.

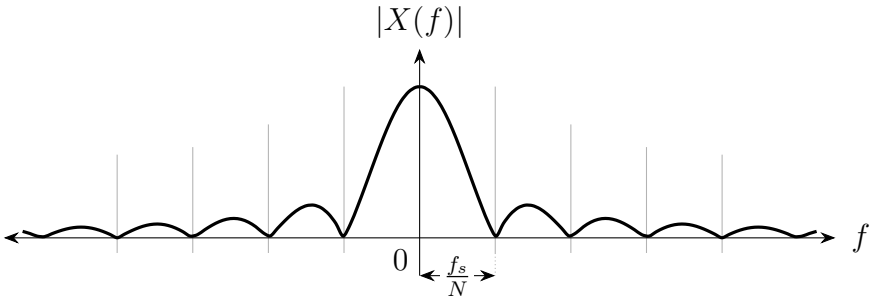


Figura 2.8. *Magnitudinea spectrală a unei ferestre rectangulare*

Rezultatul ferestruirii unui semnal în domeniul timp este o convoluție în domeniul spectral între componentele spectrale ale semnalului și acest sinus cardinal, ceea ce rezultă într-o translație pe axa frecvenței a funcției de sinus cardinal deasupra componentei semnalului.

Transformata Fourier Discretă este caracterizată prin faptul că spectrul calculat este discretizat, având același număr de componente spectrale (bin-uri) câte eșantioane am avut în domeniul timp. Având în vedere că domeniul spectral cuprinde intervalul $(-\frac{f_s}{2}, \frac{f_s}{2})$, intervalul de frecvență care caracterizează un bin spectral va fi $\frac{f_s}{N}$, ceea ce coincide exact cu dimensiunea lobilor din funcția sinus cardinal.

În cazul în care componentele sinusoidale ale semnalului au frecvența care coincide cu una din componentele discrete spectrale, atunci spectrul rezultat va conține o singură componentă de magnitudine maximă corespunzătoare cu frecvența semnalului și restul componentelor spectrale vor fi 0 (așa cum se vede pe figura 2.9a). Dacă frecvența semnalului nu coincide cu frecvența uneia din componente spectrale, vom obține

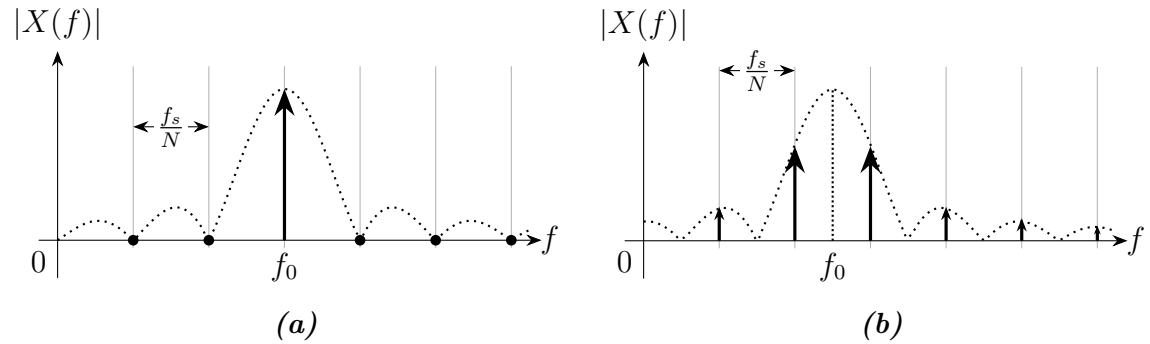


Figura 2.9. *Coincidența scurgerii spectrale cu eșantionarea spectrală*

componente spectrale mai mici cu magnitudinea prelevată din lobul principal și din lobii laterali la frecvențele discrete ale componentelor spectrale (figura 2.9b).

Fenomenul scurgerii spectrale putem să studiem și în Matlab, dacă realizăm creșterea rezoluției spectrale prin adăugarea unor eșantioane de valoare 0 la sfârșitul semnalului. Practic se păstrează fereastra rectangulară la lungimea originală, dar putem vizualiza mai multe bin-uri spectrale intercalate între cele originale:

```
»» f0 = 543;
»»
»» t = (0:1023)/8000;
»» x = sin(2*pi*f0*t);
»» X = abs(fft(x))/512;
»» w = (0:1023)*8000/1024;
»» stem(w, X);
»» hold on
»»
»» xt = [x zeros(1,7168)];
»» Xt = abs(fft(xt))/512;
»» wt = (0:8191)*8000/8192;
»» plot(wt, Xt);
»» hold off
»»
»» xlabel('f');
»» ylabel('|X(f)|');
»» axis([f0-40 f0+40 0 1]);
»» xticks(ceil((f0-40)*1024/8000)*8000/1024:8000/1024:f0+40)
```

Prezentând spectrul obișnuit și acest spectru supraeșantionat pe același grafic rezultă în figura 2.10 pe care putem observa mai clar de unde provin componentele spectrale văzute pe figura 2.7b.

Scurgerea spectrală poate fi redusă prin aplicarea a unei ferestre diferite. O importantă parte din scurgerea spectrală este cauzată de saltul brusc la marginea ferestrei.

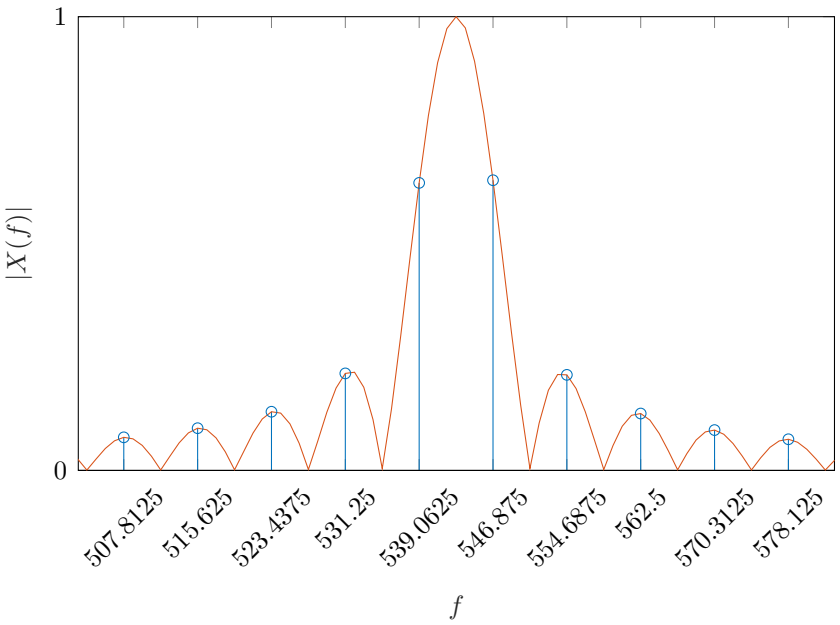


Figura 2.10. *Scurgerea spectrală în cazul unei sinusoide de 543 Hz*

Aplicând o fereastră care are o trecere lină către 0, de exemplu de forma unui clopot, putem reduce lobii laterali, reducând astfel puterea dispersată la componente de frecvență mai îndepărtate.

Ca exemplu putem observa o fereastră de tip Hamming sau Hann, definită generic prin formula:

$$w[n] = a_0 - (1 - a_0)cos\left(\frac{2\pi n}{N}\right) \quad n = \overline{0, N} \tag{2.11}$$

Fereastra Hann are $a_0 = 0.5$ iar fereastra Hamming aproximativ $a_0 \approx 0.54$. Primul Hann, prezentat și pe figura 2.11a, are un răspuns în frecvență de forma:

$$W[f] = \frac{1}{2}\frac{sinc(\pi f)}{1 - f^2} \tag{2.12}$$

După cum se vede pe figura 2.11b, fereastra Hann are lobul principal mai lat și cu amplitudinea maximă de doar jumătate din amplitudinea lobului principal al ferestrei rectangulare. Folosirea unei astfel de ferestre poate fi oportună în analiza spectrală a semnalelor alcătuite din mai multe componente (cu componentele definite într-un interval de frecvență restrâns), unde scurgerea spectrală de la o componentă la alta ar afecta de altfel identificarea corectă a acestora. Astfel de exemplu sunt semnalele de radiocomunicații prezentate pe analizorul spectral de pe figura 2.1.

Aplicând o fereastră Hann în exemplul nostru prin inserarea codului următor înainte de calculul transformatei:

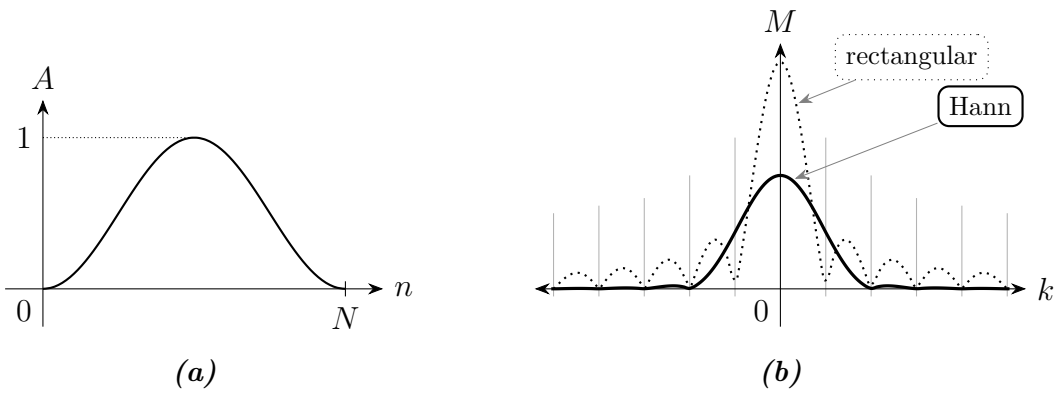


Figura 2.11. *Fereastra Hann în timp (a) și în spectru (b)*

```
>>> x = x.*hann(length(x))';
```

vom obține figura 2.12, pe care putem observa cele două componente spectrale adiacente mai mari, în rest însă, toate celelalte componente sunt mult mai mici decât fără ferestruire.

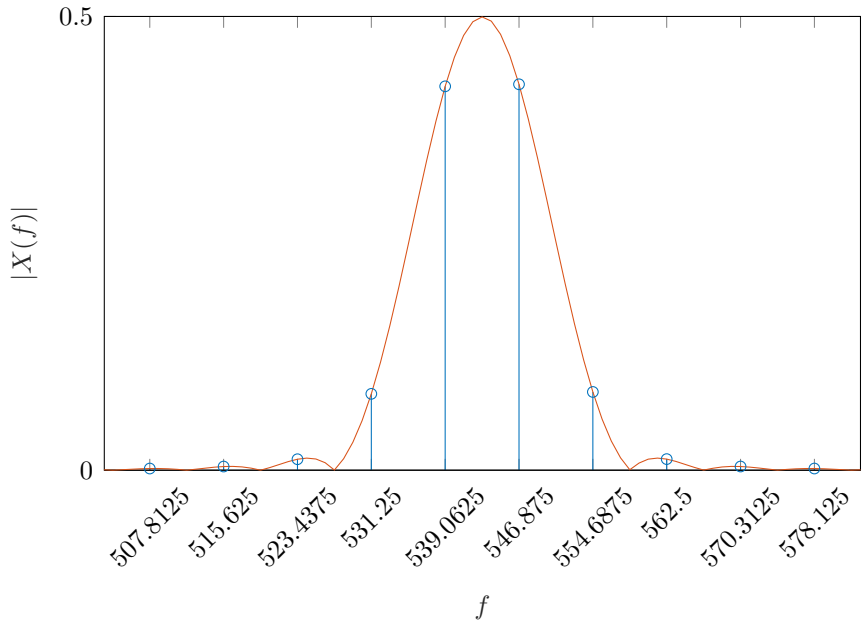


Figura 2.12. *Scurgerea spectrală în cazul unei sinusoide de 543 Hz ferestruit cu funcția Hann*

2.2. Transformata z

Transformata z este forma generalizată a transformatei Fourier discrete, similar cu transformata Laplace. Aceasta poate fi aplicată și pentru unele secvențe de semnal pentru care transformata Fourier nu există (de exemplu pentru că criteriul de convergență nu este satisfăcut). De asemenea, similar cu transformata Laplace, și transformata z poate fi folosită pentru operații algebrice în domeniul transformat.

Transformata z este definită prin formula:

$$X(z) = \sum_{n=-\infty}^{+\infty} x[n] z^{-n} \tag{2.13}$$

unde z este un număr complex. Exprimând acest număr în coordonate polare, $z = re^{-j\omega}$, obținem:

$$X(re^{j\omega}) = \sum_{n=-\infty}^{+\infty} x[n] r^{-n} e^{-j\omega n} \tag{2.14}$$

În cazul în care $r = 1$, adică dacă aplicăm pe cercul unitate din planul complex z, transformata z este echivalent cu transformata Fourier.

În Matlab transformata z poate fi calculată cu ajutorul funcției `czt`. Funcția are sintaxa:

```
>> X = czt(x, m, w, a);
```

unde `x` este secvența din domeniul timp, `m` este numărul de eșantioane spectrale pe care vrem să calculăm, `w` este o spirală definită pe planul z, de-a lungul căreia calculăm transformata și `a` este punctul de plecare a acestei spirale pe planul z. Valorile implicite pentru `m`, `w` și `a` sunt `length(x)`, `exp(-2j*pi/m)` respectiv `1` ceea ce rezultă calculul spectrului de-a lungul cercului unitate, adică echivalent cu `fft`. Astfel comanda

```
>> X = czt(x);
```

va rezulta în exact aceeași spectru ca și spectrul calculat de `X = fft(x);`

Putem însă definii și o altă curbă pe care să calculăm transformata z. Se poate, de exemplu, să facem un zoom într-o interval de frecvență mai restrânsă și să analizăm spectrul și între eșantioanele spectrale de la transformata Fourier. De exemplu, pentru a vizualiza scurgerea spectrală discutată anterior, putem folosi codul:

```
>> t = 0 : 1/8000 : 1023/8000;
>> x = sin(2 * pi * 100 * t);
>> X = czt(x);
>> W = (0:1023)/1024*8000;
>> f1 = 100-511/16; f2 = 100+512/16;
```

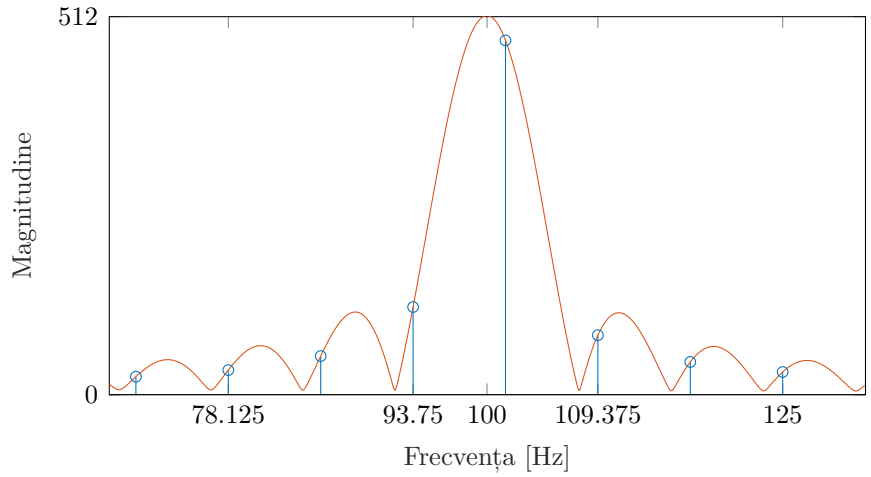


Figura 2.13. *Vizualizarea spectrului mărit într-un interval redus de frecvență (roșu) printre eșantioanele Fourier (albastru)*

```
>> w = exp(-j*2*pi*(f2-f1)/1024/8000);
>> a = exp(j*2*pi*f1/8000);
>> Xz = czt(x,1024,w,a);
>> Wz = (0:1023)/1024*(f2-f1)+f1;
>> stem(W,abs(X));
>> hold on;
>> plot(Wz,abs(Xz));
>> hold off
>> axis([f1, f2, 0, 512]);
```

rezultând în graficul de pe figura 2.13.

Utilitatea cea mai importantă a transformatei z este în analiza și sinteza filtrelor digitale. Sistemele liniare invariante în timp (cum sunt și filtrele digitale) sunt caracterizate de o funcție de transfer, care în domeniul transformat poate fi scrisă sub forma:

$$H[z] = \frac{Y[z]}{X[z]} = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + \cdots + b_N \cdot z^{-N}}{a_0 - a_1 \cdot z^{-1} - a_2 \cdot z^{-2} - \cdots - a_M \cdot z^{-M}} \tag{2.15}$$

Socotind rădăcinile polinoamelor din numărător și numitor, vom obține zerourile și polii care caracterizează sistemul LIT.

În Matlab pentru a afișa funcția de transfer în domeniul transformat a unui sistem LIT se folosește funcția `freqz(b, a)` iar pentru a afișa polii și zerourile pe planul z, funcția `zplane(b, a)`.

De exemplu un LIT cu funcția de transfer treaptă unitară, definită cu relația:

$$\mu(z) = \frac{1}{1 - \alpha z^{-1}} \tag{2.16}$$

va avea o caracteristică de transfer din figura 2.14, obținută cu comanda:

```
>> freqz(1, [1, -10]);
```

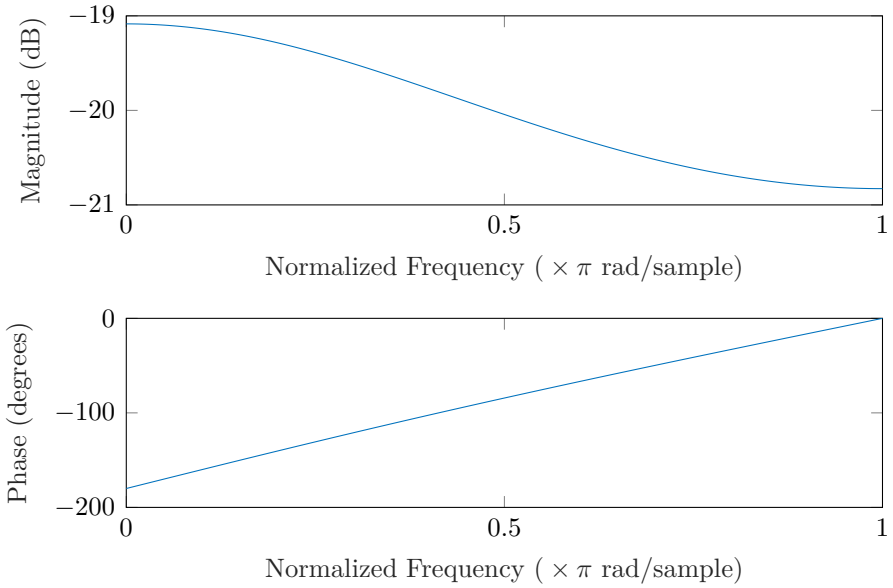


Figura 2.14. *Funcție de transfer în domeniul spectral*

2.3. Exerciții

1. Observați spectrul și identificați componentele spectrale ale următoarelor semnale (fiecare eșantionat la 8 kHz și având 1024 de eșantioane):
   - dinte de fierăstrău de frecvența de 125 Hz
   - dreptunghiular de frecvența de 125 Hz
   - sinusoidal de frecvența de 125 Hz trecut printr-un redresor dublă alternanță
   - dinte de fierăstrău de frecvența de 1250 Hz
2. Afișați răspunsul în frecvență și identificați frecvența de tăiere al unui filtru notch definit cu următoarele coeficienți:

```
>> b = [ 0.9859 -1.9703 0.9859 ];
>> a = [ 1.0000 -1.9703 0.9718 ];
```

Lucrarea 3

Transformata Fourier Rapidă

În această lucrare vom studia avantajele folosirii algoritmului rapid de calculare a transformatei Fourier și modurile de implementare a acestuia.

3.1. Transformata Fourier Discretă

Transformata Fourier discretă este varianta eșantionată (atât în timp cât și în frecvență) a transformatei Fourier continue. Transformata Fourier discretă a unui semnal $x[n]$ format din N eșantioane este calculată cu formula:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = \overline{0, N-1} \tag{3.1}$$

Pentru calculul unei singure valori din spațiul transformatei sunt necesare N operații, prin urmare complexitatea algoritmului de calcul al transformatei Fourier pentru o secvență de N eșantioane, este $O(N^2)$.

Pentru a interpreta această complexitate, putem implementa un cod simplu în Matlab, care calculează transformata Fourier a unui semnal exact după formula (3.1):

dft.m

```
% DFT(x) - calculează transformata Fourier discretă a semnalului x
%          costul de memorie și timp de calcul sunt de ordinul O(N^2)
function [X] = dft(x)
  N = length(x);
  k = 0:N-1;
  n = 0:N-1;
  % înmulțire matricială de dimensiunea N × N
  X = x * exp(-2j * pi / N * k' * n);
endfunction
```

complexitatea $O(N^2)$ la această implementare se manifestă prin memoria folosită pentru înmulțirea matricială și rezultă într-un timp de procesare foarte mare comparativ

cu funcția `fft` studiată anterior, cum poate fi observat prin măsurarea timpului pentru cele două operații:

```
>> tic; X = dft(x); toc
>> tic; X = fft(x); toc
```

Creșterea timpului de calcul pentru DFT va fi mult mai semnificativă odată cu creșterea lungimii semnalului.

Complexitatea calculului transformatei Fourier poate fi însă redusă prin aplicarea unui algoritm inventat în 1965 de James Cooley și John Tukey numit FFT[1] care poate realiza operația de transformare cu o complexitate $O(N \log_2 N)$ prin exploatarea proprietăților de simetrie și periodicitate a funcțiilor *sin* și *cos*) și refolosirea coeficienților la înmulțirile complexe. Prin această refolosire a coeficienților, calculele pot fi realizate pe jumătăți de semnale (semnal decimat), iar prin aplicarea operației de decimare în mod repetitiv se va reduce timpul total de calcul. Decimarea poate fi făcută atât dinspre reprezentarea semnalului în timp cât și dinspre reprezentarea semnalului în frecvență, rezultând în două moduri diferite de implementare a algoritmului FFT.

3.2. Algoritmul FFT cu decimare în timp

Pentru a realiza această operație de decimare, vom calcula separat prima jumătate a spectrului respectiv cea de a doua jumătate a spectrului. Pentru fiecare eșantion spectral se vor realiza calculele pe semnalul reprezentat în timp decimat în două, separând astfel elementele pare și elementele impare:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = \overline{0, \frac{N}{2} - 1} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{N}k2n} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{N}k(2n+1)} \end{aligned} \tag{3.2}$$

Dacă reorganizăm coeficienții, atunci putem observa că ecuația se simplifică într-o operație de transformată Fourier pe o jumătate de secvență:

$$\begin{aligned} X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{N}2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{N}2kn} \cdot e^{-j\frac{2\pi}{N}k} \\ &= \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{X_{par}(k)} + \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn} \cdot e^{-j\frac{2\pi}{N}k}}_{X_{impar}(k)} \end{aligned} \tag{3.3}$$

[1] Fast Fourier Transform - Transformată Fourier Rapidă

Am notat cu $X_{par}(k)$ transformata Fourier discretă a eșantioanelor pare ale secvenței $x(n)$ iar cu $X_{impar}(k)$ a celor impare. Așadar pentru a calcula prima jumătate a transformatei Fourier unei secvențe x, va trebui să calculăm transformata Fourier a elementelor pare, respectiv a elementelor impare și să le însumăm (elementele impare fiind ponderate cu rădăcinile unității de ordinul N):

$$X(k) = X_{par}(k) + e^{-j\frac{2\pi}{N}k} \cdot X_{impar}(k), \quad k = \overline{0, \frac{N}{2} - 1} \tag{3.4}$$

Deși până aici formula de calcul pentru transformata Fourier a devenit puțin mai complicată, adevăratul avantaj iese la iveală dacă aplicăm aceeași decimare și pentru jumătatea superioară a spectrului calculat:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = \overline{\frac{N}{2}, N - 1} \tag{3.5}$$

modificăm intervalul de parcurgere a k pentru a avea o formulă comparabilă cu prima jumătate deja calculată:

$$X(k + \frac{N}{2}) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}(k+\frac{N}{2})n}, \quad k = \overline{0, \frac{N}{2} - 1} \tag{3.6}$$

după care procedăm la o rearanjare a coeficienților similar cu prima parte:

$$\begin{aligned}
X(k + \frac{N}{2}) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{N}(k+\frac{N}{2})2n} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{N}(k+\frac{N}{2})(2n+1)} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{N}2kn} \cdot e^{-j\frac{2\pi}{N}\frac{N}{2}2n} + \\
&\quad \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{N}2kn} \cdot e^{-j\frac{2\pi}{N}\frac{N}{2}2n} \cdot e^{-j\frac{2\pi}{N}k} \cdot e^{-j\frac{2\pi}{N}\frac{N}{2}} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn} \cdot \underbrace{e^{-j2\pi n}}_{1} + \\
&\quad \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn} \cdot \underbrace{e^{-j2\pi n}}_{1} \cdot e^{-j\frac{2\pi}{N}k} \cdot \underbrace{e^{-j\pi}}_{-1} \\
&= \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{X_{par}(k)} - \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \cdot e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{X_{impar}(k)} \cdot e^{-j\frac{2\pi}{N}k}
\end{aligned} \tag{3.7}$$

așadar pentru cealaltă jumătate a spectrului Fourier vom avea nevoie tot așa de calculul transformatei Fourier pe elementele pare respectiv cele impare din secvența originală:

$$X(k+\frac{N}{2}) = X_{par}(k) - e^{-j\frac{2\pi}{N}k} \cdot X_{impar}(k), \quad k = \overline{0, \frac{N}{2}-1} \tag{3.8}$$

Astfel transformata Fourier discretă a unui semnal poate fi rescrisă prin formula:

$$\begin{cases} X(k) & = X_{par}(k) + w_N^k \cdot X_{impar}(k) \\ X(k+\frac{N}{2}) & = X_{par}(k) - w_N^k \cdot X_{impar}(k) \end{cases}, \quad k = \overline{0, \frac{N}{2}-1} \tag{3.9}$$

unde pentru simplitate rădăcinile unității de ordinul N s-au notat cu w_N^k.

Această formulă poate fi reprezentată grafic ca aripile unei fluture (de aici vine denumirea celulei din FFT: butterfly) reprezentată grafic în Figura 3.1.

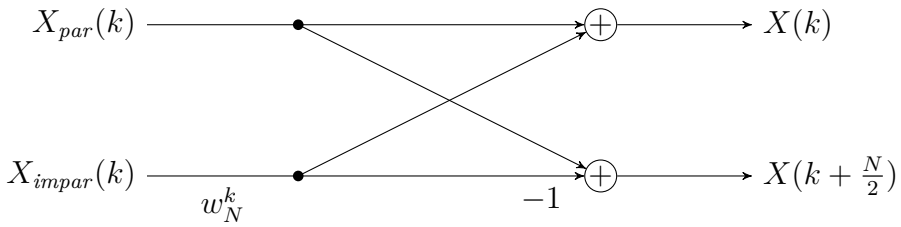


Figura 3.1. *Celulă de tip fluture*

Bineînțeles, o singură celulă tip fluture nu ne ajută în reducerea complexității algoritmului, dar putem observa faptul că pentru transformările jumătăților pare și impare necesare putem să aplicăm recursiv aceeași metodă de decimare. Astfel, prin decimări succesive putem ajunge la o secvență cu un singur eșantion a cărei transformată Fourier coincide cu însăși eșantionul respectiv:

$$X(0) = x(0) \cdot e^{-j\frac{2\pi}{1}0} = x(0), \quad \text{pentru } N = 1 \tag{3.10}$$

De aici se poate reconstrui foarte ușor transformatele Fourier intermediare și cel final cu $log_2(N)$ niveluri de celule fluture. Descompunerea recursivă în elemente pare, respectiv impare și fluturele corespunzătoare transformării Fourier a unui semnale de 8 eșantioane poate fi văzută în Figura 3.2. Pentru decongestionarea figurii, înmulțirile cu constantele w_N^k s-au marcat cu — iar ramura înmulțită suplimentar și cu -1 cu —.

Se poate observa că pentru calculul transformatei Fourier a semnalului de 8 eșantioane, vom avea nevoie de 3 niveluri de câte 8 operații (adunare ponderată), adică vom avea o complexitate $O(N \log_2 N)$. De asemenea, trebuie notat faptul că putem atinge această reducere în complexitate numai dacă se poate realiza decimarea eșantioanelor până la capăt, adică numărul de eșantioane din semnalul de la intrare este o putere a lui 2.

Această abordare recursivă de decimare cu rearanjarea vectorului în elemente pare și impare poate fi implementată foarte ușor într-o funcție recursivă de forma:

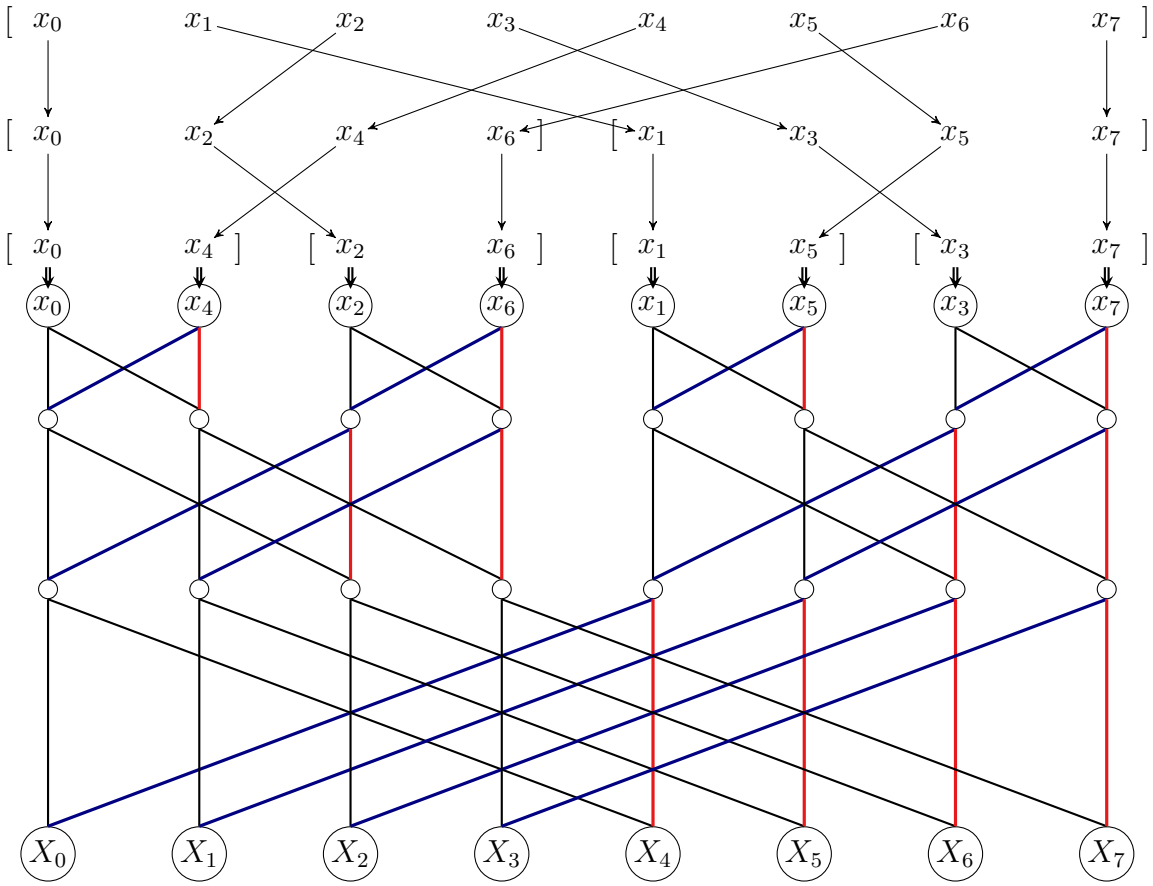


Figura 3.2. *Fluturele de calcul al FFT cu decimare în timp pentru 8 eșantioane*

ditfft.m

```
% DITFFT(x) - calculează transformata Fourier a lui x prin decimare
%             în timp
function [X] = ditfft(x)
  N = length(x); % lungimea lui x, trebuie să fie putere a lui 2
  if N <= 1
    X = x;
  else
    % împărţim în elemente pare respectiv impare şi aplicăm recursiv
    % metoda de decimare în timp
    Xpar = ditfft(x(1:2:end));
    Ximpar = ditfft(x(2:2:end)) .* exp(-2j * pi * (0:N/2-1) / N);
    % reconstruim spectrul din cele două jumătăţi
    X = [Xpar + Ximpar Xpar - Ximpar];
  endif
endfunction
```

Comparând timpul de execuție al transformatei Fourier discrete implementată cu algoritmul $O(N^2)$, respectiv cel al transformatei Fourier rapide implementată cu algo-

ritmul $O(N \log_2 N)$, putem observa reducerea de timp semnificativă a celui din urmă odată cu creșterea numărului de eșantioane (vezi Figura 3.3).

fft_time.m

```
n = 8:13;
tdft = [];
tfft = [];
for N = n
  x = rand(1,2^N);
  tic; X = dft(x); tdft = [tdft toc];
  tic; X = ditfft(x); tfft = [tfft toc];
endfor
plot(n, tdft, n, tfft);
```

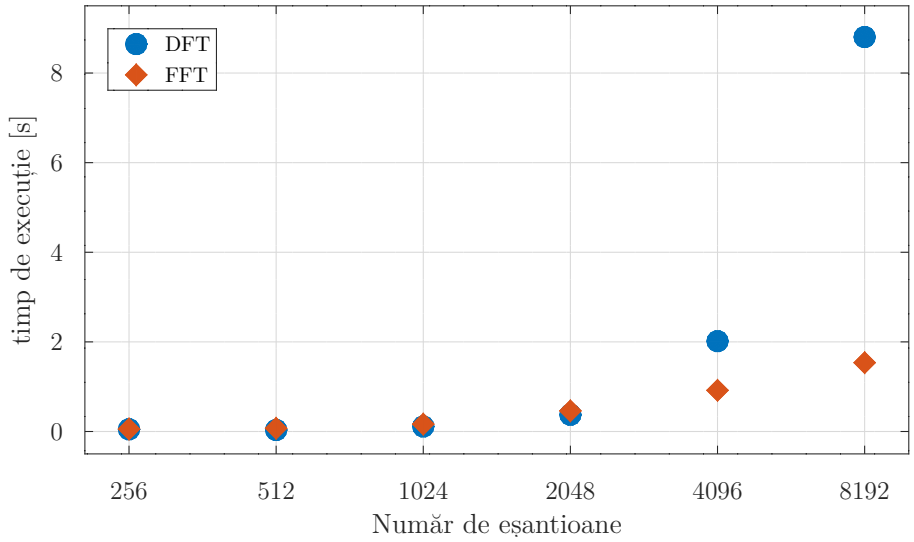


Figura 3.3. *Timpul de execuție a algoritmului FFT comparativ cu DFT*

3.3. Algoritmul FFT cu decimare în frecvență

A doua variantă de a ajunge la reducerea timpului de calcul a transformatei FFT este să realizăm decimarea în domeniul transformat. Pentru aceasta dezvoltăm separat termenii pari și impari ai transformatei Fourier:

$$\begin{cases} X(2k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}2kn}, & (3.11) \\ X(2k+1) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}(2k+1)n}, & (3.12) \end{cases} \qquad k = \overline{0, \frac{N}{2} - 1}$$

În continuare procedăm similar cu decimarea în frecvență, dar de data aceasta desfacem ecuațiile în jumătăți inferioare și jumătăți superioare. Astfel, pentru termenii pari (din ec. 3.11) vom avea:

$$\begin{aligned}
X(2k) &= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn} + \sum_{n=\frac{N}{2}}^{N-1} x(n)e^{-j\frac{2\pi}{N}2kn} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}2k(n+\frac{N}{2})} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}2kn}e^{-j\frac{2\pi}{N}2k\frac{N}{2}} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}2kn}\underbrace{e^{-j2\pi k}}_{1} \\
&= \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{X_{inf}(k)} + \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{X_{sup}(k)}
\end{aligned} \tag{3.13}$$

Se poate observa că practic termenii pari ai unei transformate Fourier pot fi calculați ca suma simplă a transformatei Fourier din prima jumătate a vectorului cu transformata Fourier din a doua jumătate a vectorului.

Dacă dezvoltăm asemănător și termenii impari (din ec. 3.12) obținem:

$$\begin{aligned}
X(2k+1) &= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}(2k+1)n} + \sum_{n=\frac{N}{2}}^{N-1} x(n)e^{-j\frac{2\pi}{N}(2k+1)n} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}(2k+1)n} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}(2k+1)(n+\frac{N}{2})} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn}e^{-j\frac{2\pi}{N}n} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}2kn}e^{-j\frac{2\pi}{N}2k\frac{N}{2}}e^{-j\frac{2\pi}{N}n}e^{-j\frac{2\pi}{N}\frac{N}{2}} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{N}2kn}e^{-j\frac{2\pi}{N}n} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{N}2kn}\underbrace{e^{-j2\pi k}}_{1}e^{-j\frac{2\pi}{N}n}\underbrace{e^{-j\pi}}_{-1} \\
&= \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(n)e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{\text{“}X_{inf}(k)\text{”}} e^{-j\frac{2\pi}{N}n} - \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2})e^{-j\frac{2\pi}{\frac{N}{2}}kn}}_{\text{“}X_{sup}(k)\text{”}} e^{-j\frac{2\pi}{N}n}
\end{aligned} \tag{3.14}$$

Se poate vedea imediat similaritatea cu secvența elementelor impare, unde am și notat partea inferioară și partea superioară întâlnită în ecuația 3.13, dar imediat observăm că deși ar fi la îndemână o notare de genul

$$\begin{cases} X(2k) &= X_{inf}(k) &+ X_{sup}(k) \\ X(2k+1) &= X_{inf}(k) \cdot w_N^n &- X_{sup}(k) \cdot w_N^n \end{cases}$$

nu putem să realizăm aceeași metodă recursivă pe care am aplicat în cazul decimării în timp, deoarece coeficienții alcătuiți din rădăcinile unității (w_N^n) nu sunt independenți de suma cu care calculăm transformatele Fourier a jumătăților de semnal. De aceea am notat în ghilimele expresia corespunzătoare acestor transformate în ecuația 3.14.

Pentru a putea aplica formula aceasta trebuie să realizăm înmulțirea cu coeficienții w_N^n în timpul calculului jumătăților de secvență și nu după realizarea acestor calcule. Asta înseamnă că va trebui să abordăm un calcul iterativ, pornind de la secvențele alcătuite dintr-un singur element și la fiecare pas trecând la o secvență de lungime dublă. Abordarea aceasta iterativă poate fi urmărită pe fluturele reprezentat în figura 3.4. Bineînțeles pentru simplitate și pe această figură a fost marcat înmulțirea cu coeficienți prin colorarea traseului.

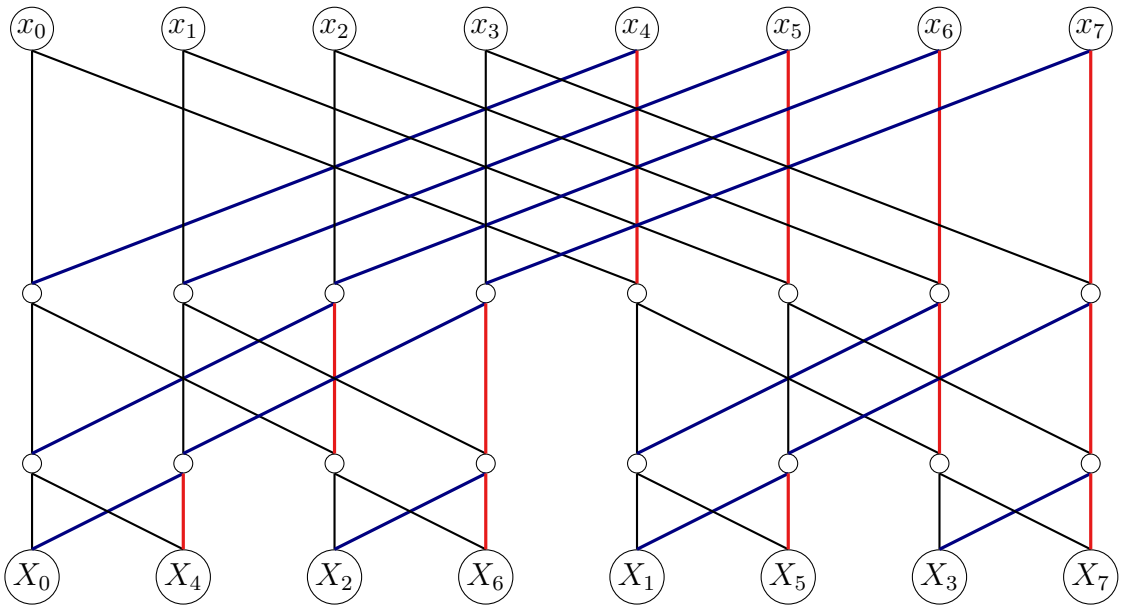


Figura 3.4. *Fluturele de calcul al FFT cu decimare în frecvență pentru 8 eșantioane*

Se poate observa că aplicând această metodă iterativă putem de asemenea realiza operația cu o complexitate $O(N \log_2 N)$, printr-o buclă de $\log_2 N$ pași fiecare realizând N operații de înmulțire-adunare (presupunând toți coeficienții fiind calculați în prealabil).

Singurul impediment este că la finalul operației vectorul rezultat va avea elementele amestecate. Dacă însă ne uităm mai atent la ordinea elementelor în secvența rezultată

putem observa o regulă simplă: indexul elementelor va fi în reprezentare binară exact valoarea ordinii corecte cu biții în ordine inversă.

0	000	000	0
4	100	001	1
2	010	010	2
6	110	011	3
1	001	100	4
5	101	101	5
3	011	110	6
7	111	111	7

Inversarea aceasta a ordinul biților poate să fie costisitoare dacă e implementat în software fiind dependent de numărul exact de biți necesari pentru reprezentarea elementelor din vector, dar poate fi implementat foarte ieftin în hardware. În acest sens există anumite procesoare dedicate prelucrării digitale a semnalelor (DSP - Digital Signal Processor) care beneficiază de un set de instrucțiuni extinse cu scopul de a implementa mai ușor algoritmi specifici prelucrărilor de semnale. În acest sens, pentru optimizarea calculului transformatei Fourier, vom avea o operație de oglindire a biților, în general implementat direct pe magistrala de adrese, adică fiind posibil accesarea elementelor în ordinea corectă într-un singur tact. De asemenea există și instrucțiuni de tip MAC (Multiply-and-ACcumulate), care poate realiza într-un singur tact înmulțirea cu un coeficient și adunarea rezultatului într-un acumulator. Aceste instrucțiuni pot să lucreze și pe mai multe date în paralel, datorită tehnicii SIMD (Single Instruction Multiple Data), adică o adunare se poate realiza deodată și pe partea reală și pe partea imaginară a unor numere complexe.

Practic putem avea o implementare foarte eficientă a transformatei Fourier, bazându-ne pe un pseudocod de forma:

$N \leftarrow len_x$
while $\frac{N}{2} \geq 1$ **do**
  for $m \leftarrow 0$ **to** $\frac{N}{2} - 1$ **do**
    $w \leftarrow w_N(m \cdot \frac{len_x}{N})$, $\omega \leftarrow \bar{w}$
    for $n \leftarrow m$ **to** $len_x - 1$ **by** N **do**
      $a \leftarrow x(n)$
      $b \leftarrow x(n + \frac{N}{2})$
      $x(n) \leftarrow a + b$ `// ADD2 a.re,b.re,x.re; a.im,b.im,x.im`
      $x(n + \frac{N}{2}) \leftarrow aw - bw$ `// SUB2 a.re,b.re,c.re; a.im,b.im,c.im`
      `// ADD2 c.re,w.re,x.re; c.re,w.im,x.im`
      `// MAC2 c.im,ω.im,x.re; c.im,w.re,x.im`
  $N \leftarrow \frac{N}{2}$
$X \leftarrow x[br]$ `// bit reversed addressing`

3.4. Exerciții

1. Implementați în Matlab algoritmul FFT prin decimare în frecvență și comparați timpul de execuție cu timpii de execuție a celorlalte metode prezentate în lucrare.

2. Implementați în Matlab algoritmul FFT prin decimare în timp în mod iterativ și comparați timpul de execuție cu algoritmul implementat recursiv. Explicați diferențele.

3. Studiați setul de instrucțiuni de la diferite procesoare digitale de semnale (TI TMS320C6xxx, FreeScale StarCore, AD SHARC) și identificați facilitățile oferite pentru implementarea optimizată a transformatei Fourier rapide.

Lucrarea 4

Filtrarea semnalelor

Filtrarea este o operație fundamentală de procesare dintr-un sistem de procesare a semnalelor. Este utilizată pentru a elimina componentele spectrale nedorite din semnal. În funcție de tipul filtrului (trece jos, trece bandă sau trece sus), acesta va lăsa să treacă doar componentele semnalului util cu frecvențele dintr-o bandă specificată, rejectând toate celelalte componente cu frecvențe în afara benzii de trecere. Folosirea filtrelor este omniprezentă în lanțurile de procesare, mai ales pentru eliminarea:

- zgomotului – provenind din zgomotul termic a componentelor, zgomotul de comutație în circuitele de semnal mixt, zgomotul de cuantizare la conversia analog-digitală, etc. – manifestând de obicei prin componente de frecvență înaltă ce pot fi eliminate cu filtre trece jos
- componentelor provenite din interferență electromagnetică – de exemplu componenta de 50Hz de la rețeaua electrică – ce poate fi eliminat cu filtru oprește bandă
- componentelor care ar produce alterarea semnalului util datorită violării criteriului Nyquist – ce se elimină cu filtre anti-aliere
- componentelor nedorite în unele faze ale prelucrării – de exemplu una din benzile laterale a semnalului modulat – ce pot fi eliminate cu filtre trece bandă

4.1. Noțiuni teoretice

Operația de filtrare reprezintă, de regulă, trecerea unui semnal $x(t)$ printr-un sistem liniar invariant în timp, a cărui funcție pondere $h(t)$ este cunoscută (vezi Figura 4.1).

Un sistem se numește liniar, dacă respectă principiul superpoziției, adică dacă la intrarea sistemului aplicăm o combinație liniară de două (sau mai multe) semnale $ax_1(t) + bx_2(t),\ a, b \in \mathbb{R}$, atunci semnalul de la ieșirea sistemului poate fi scris ca fiind $ay_1(t) + by_2(t)$, unde $y_1(t)$ și $y_2(t)$ reprezintă răspunsurile sistemului la semnalele $x_1(t)$ respectiv $x_2(t)$.

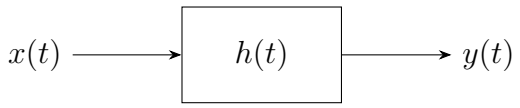


Figura 4.1. *Sistem liniar invariant în timp*

Dacă răspunsul sistemului este mereu același pentru un același semnal de intrare, indiferent de momentul de timp la care este aplicat semnalul respectiv la intrarea sistemului, atunci sistemul se numește invariant în timp.

Sistemul are asociat o funcție pondere $h(t)$, care reprezintă răspunsul la impuls al sistemului, adică ieșirea corespunzătoare unui impuls Dirac aplicat la intrare. Operația realizată de un bloc de filtrare nu este altceva decât o operație de convoluție dată de următoarea formulă:

$$y(t) = h(t) * x(t) = \int_{-\infty}^{+\infty} h(\tau) \cdot x(t-\tau)\,\mathrm{d}\tau = \int_{-\infty}^{+\infty} h(t-\tau) \cdot x(\tau)\,\mathrm{d}\tau \tag{4.1}$$

Aplicând transformata Fourier asupra ecuației 4.1 va rezulta într-un produs simplu al intrării și a funcției pondere din domeniul transformat:

$$Y(\omega) = H(\omega) \cdot X(\omega) \tag{4.2}$$

unde $Y(\omega)$, $H(\omega)$ și $X(\omega)$ reprezintă spectrele Fourier ale lui $y(t)$, $h(t)$ respectiv $x(t)$. $H(\omega)$ poartă numele de funcție de transfer a sistemului/filtrului.

Prin alegerea corespunzătoare a funcției de transfer putem elimina o parte din spectrul semnalului de la intrare și astfel se poate realiza o operație de filtrare. De exemplu putem considera un filtru trece jos ideal care va avea ca funcție de transfer o treaptă unitară, cu amplitudinea de 1 sub o frecvență de tăiere $\omega_0 = 2\pi f_0$ și amplitudinea de 0 peste această frecvență (fig. 4.2). Aplicând această funcție la spectrul semnalului de intrare, componentele spectrale din intervalul $(0, f_0)$ se păstrează nealterat, iar componentele spectrale cu $f > f_0$ vor fi eliminate complet.

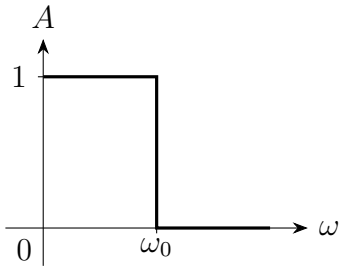


Figura 4.2. *Funcția de transfer a unui filtru trece jos ideal*

Realizând transformata Fourier inversă asupra răspunsului în frecvență ideal, obținem funcția de pondere $h(t) = \frac{\omega_0}{\pi}\,\mathrm{sinc}(\omega_0(t-t_0))$, prezentat pe fig. 4.3.

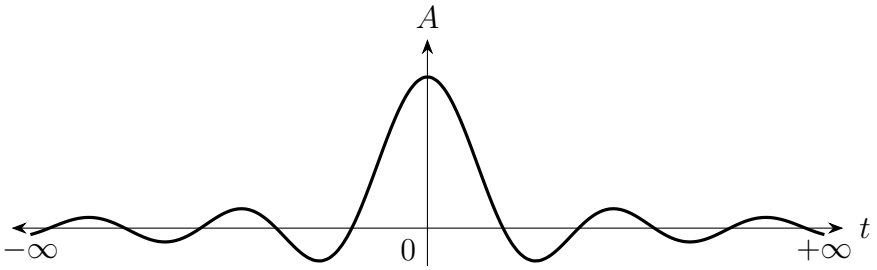


Figura 4.3. *Funcția pondere a unui filtru trece jos ideal*

4.2. Filtrarea digitală a semnalelor

În domeniul discret, convoluția, care definește operația de filtrare realizată printr-un sistem liniar invariant în timp, va avea următoarea formă:

$$y(n) = \sum_{k=-\infty}^{\infty} h(k) \cdot x(n-k) \tag{4.3}$$

unde $x(n)$ este semnalul de la intrare, iar $h(k)$ este funcția de transfer sau funcția pondere care caracterizează sistemul LIT.

Evident, pentru a putea fi calculat pe calculator, lungimea funcției de pondere trebuie să fie finită, astfel convoluția va deveni:

$$y(n) = \sum_{k=0}^{K} h(k) \cdot x(n-k) \tag{4.4}$$

K fiind lungimea vectorului cu ponderile funcției de transfer. Această operație este echivalentă cu înmulțirea a două polinoame cu coeficienți din vectorii x și h, ceea ce este simplu de realizat cu ajutorul unui procesor.

Așa cum am văzut însă în secțiunea 4.1, pentru a realiza un filtru trece jos ideal vom avea nevoie de o funcție sinus cardinal, care în domeniul discret va avea forma:

$$h(n) = \frac{\omega_0}{\pi} \operatorname{sinc}(\omega_0 n), \quad n = \overline{-\infty, \infty} \tag{4.5}$$

Această funcție are o lungime infinită și nu este cauzală, deci o filtrare (adică o convoluție) cu această funcție nu este realizabilă în practică.

Filtrarea cu răspuns finit

O metodă simplă de a realiza filtrarea digitală în practică este să limităm în timp funcția sinus cardinal, așa cum se vede pe figura 4.4, aplicând:

$$h(n) = \begin{cases} \frac{\omega_0}{\pi} \operatorname{sinc}(\omega_0 n), & \text{dacă } |n| \leq T_0 \\ 0, & \text{altfel} \end{cases} \tag{4.6}$$

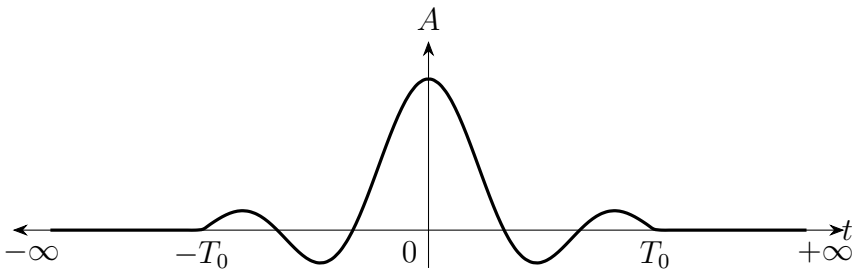


Figura 4.4. *Funcția pondere cu lungime finită*

Bineînțeles, această formă în continuare violează condiția de cauzalitate, pentru a putea aplica în practică. De aceea, mai trebuie să aplicăm și o translație, astfel încât toți coeficienții nenuli ai filtrului să ajungă în dreapta de origine. Dacă intervalul $(-T_0, T_0)$ este împărțit în N eșantioane, atunci funcția de transfer va deveni:

$$h(n) = \frac{\omega_0}{\pi} \operatorname{sinc}(\omega_0(n - \tfrac{N}{2})), \quad n = \overline{0, N-1} \tag{4.7}$$

Prin introducerea acestei trunchieri a răspunsului filtrului în timp, funcția de transfer va avea de suferit o deviere față de răspunsul ideal (vezi fig. 4.5):

- în loc de trecere bruscă din 1 în 0 a răspunsului vom avea o tranziție mai lină; panta funcției de transfer la tranziția din banda de trecere către banda de tăiere depinde de lungimea răspunsului în timp, pentru a obține o pantă mai abruptă trebuie crescut numărul de eșantioane
- apare oscilații în forma funcției de transfer, numit ripluri, cauzate de trecerea bruscă în 0 a funcției pondere; apariția riplurilor poate fi redus prin aplicarea unor ferestre mai line de trecere către 0 la trunchierea răspunsului în timp, acest lucru însă va introduce o curbă mai mare și o zonă mai largă de tranziție între banda de trecere și banda de tăiere.

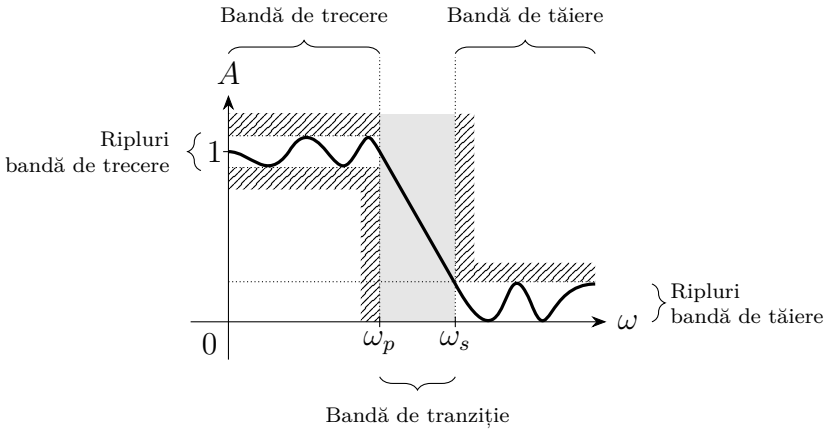


Figura 4.5. *Funcția de transfer a unui filtru trece jos real*

Pentru a vizualiza efectul asupra caracteristicii filtrului folosind Matlab, putem porni de la o treaptă unitară (fig. 4.6), având grijă ca să construim și imaginea în oglindă a frecvențelor negative necesară pentru transformata Fourier inversă:

```
>> w = -pi : pi/512 : pi;
>> Hideal = zeros(size(w));
>> Hideal(abs(w) <= pi/4) = 1;
>> plot(w,Hideal);
```

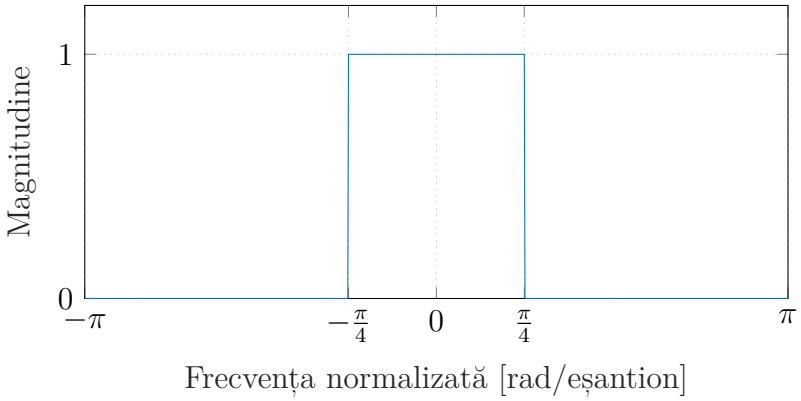


Figura 4.6. *Funcția de transfer în frecvență a unui FTJ ideal cu frecvența de tăiere* $\frac{f_s}{8}$

Pe funcția de transfer ideală aplicăm transformata Fourier inversă pentru a obține răspunsul impuls (fig. 4.7). Pentru a rearanja vectorii în ordinea preferată de transformata Fourier rapidă se folosește `fftshift`:

```
>> hideal = ifft(fftshift(Hideal));
>> plot(-512:512,real(fftshift(hideal)));
```

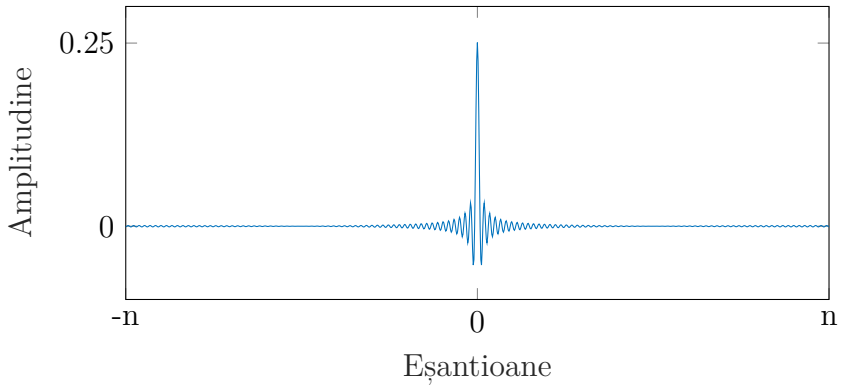


Figura 4.7. *Răspunsul la impuls a unui FTJ ideal*

După ce am obținut funcția pondere, trunchiem aceasta la un număr $2N + 1$ de eșantioane (fig. 4.8):

```
>> N = 8;
>> i = [1:N+1 length(hideal)-(N-1:-1:0)];
>> hreal = zeros(size(hideal));
>> hreal(i) = hideal(i);
>> plot(-512:512,real(fftshift(hideal)));
```

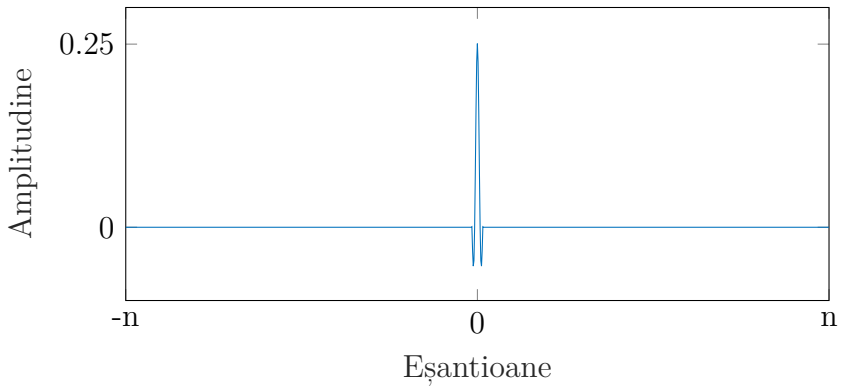


Figura 4.8. *Răspunsul la impuls a unui FTJ trunchiat la* $2 \cdot 8 + 1$ *eșantioane*

Iar la final calculăm funcția de transfer în frecvență a filtrului obținut aplicând transformata Fourier pe răspunsul la impuls trunchiat (fig. 4.9):

```
>> Hreal = fft(hreal);
>> plot(w,abs(fftshift(Hreal)));
```

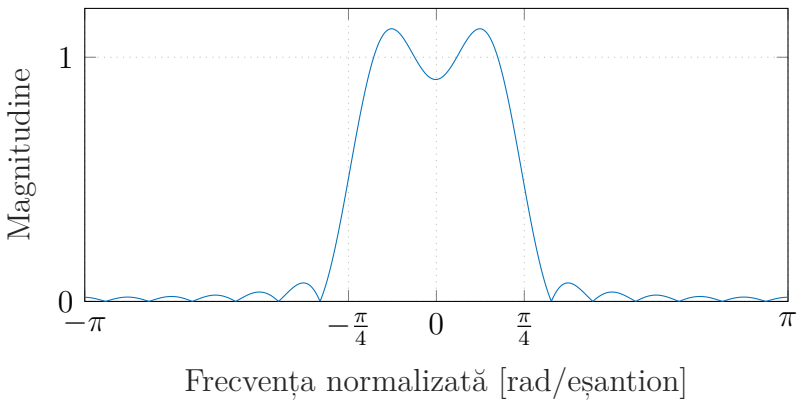


Figura 4.9. *Funcția de transfer a unui FTJ cu răspuns finit la impuls de* $2 \cdot 8 + 1$ *eșantioane și frecvența de tăiere* $\frac{f_s}{8}$

Aplicând trunchierea cu diferite valori N putem observa și efectul lungimii răspunsului la impuls asupra funcției de transfer în frecvență. Pe figura 4.10 putem observa că funcția de transfer se apropie de cel ideal pe măsură ce se mărește foarte mult lungimea răspunsului la impuls.

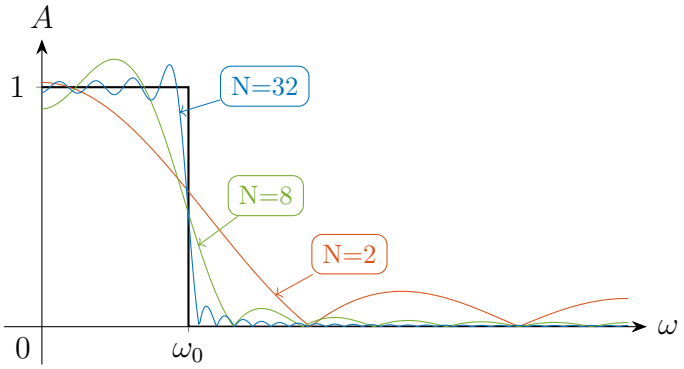


Figura 4.10. *Funcțiile de transfer a unor filtre trece jos cu lungimi diferite*

Filtrarea cu răspuns infinit

Creșterea prea mare a numărului de coeficienți însă are dezavantajul creșterii timpului de calcul, care poate provoca probleme în implementările practice. Ca o soluție la această problemă se poate implementa operația de filtrare folosind un sistem alcătuit din două sisteme LIT într-o configurație cu reacție negativă (fig. 4.11).

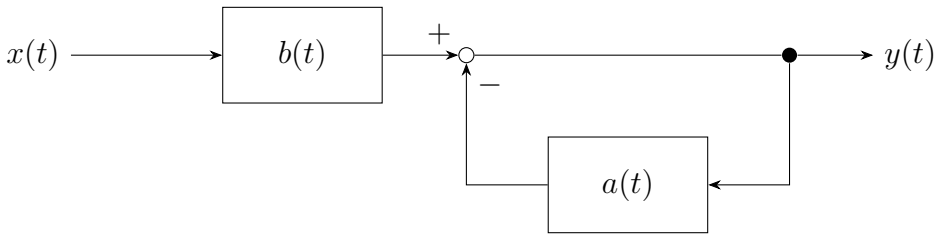


Figura 4.11. *Construirea unui filtru cu răspuns infinit la impuls din două sisteme LIT*

Astfel, operația de filtrare cu răspuns infinit se poate descrie cu două convoluții finite sub forma:

$$\begin{aligned} a(1)\cdot y(n) = b(1)\cdot x(n) + b(2)\cdot x(n-1) + \cdots + b(len_b)\cdot x(n-len_b+1) - \\ - a(2)\cdot y(n-1) - \cdots - a(len_a)\cdot y(n-len_a+1) \end{aligned} \tag{4.8}$$

Folosind un număr mic de coeficienți, într-o structură ce folosește blocuri de întârziere și amplificare atât la intrare cât și la ieșire, ca în figura 4.12, se poate obține un răspuns la impuls care tinde spre infinit.

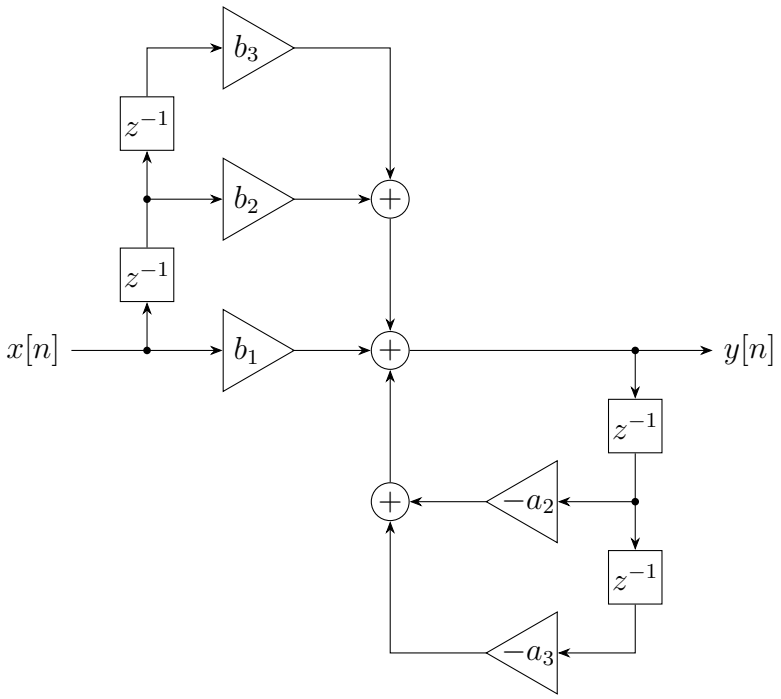


Figura 4.12. *Implementarea unui filtru cu blocuri de întârziere și amplificare*

Aplicând transformata Z asupra ecuației 4.8 obținem funcția de transfer:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_1 + b_2 z^{-1} + \cdots + b_{len_B} z^{-len_B+1}}{1 + a_2 z^{-1} + \cdots + a_{len_A} z^{-len_A+1}} \tag{4.9}$$

Astfel, filtrul digital cu răspuns infinit putem considera echivalentul discret al unui filtru analogic. De exemplu, un filtru RC trece jos cu un singur pol (prezentat în figura 4.13) are funcția de transfer (exprimată în domeniul transformat Laplace):

$$H(s) = \frac{1}{1 + RCs} \tag{4.10}$$

care determină o pantă de $-20dB/$decadă, care trece prin $-3dB$ la pulsația de tăiere $\omega_0 = \frac{1}{RC}$, adică frecvența de tăiere $f_0 = \frac{1}{2\pi RC}$, așa cum se vede pe caracteristica din figura 4.14.

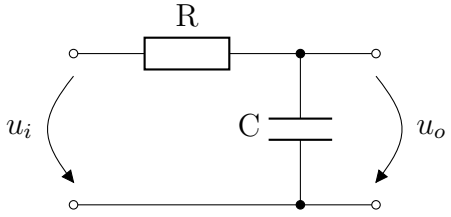


Figura 4.13. *Filtru RC trece jos cu un singur pol*

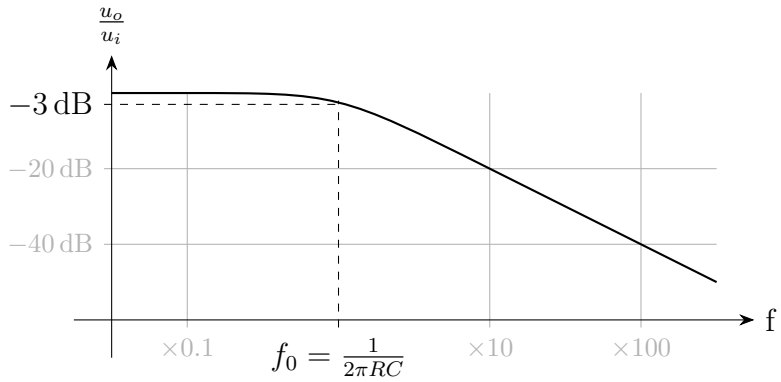


Figura 4.14. *Caracteristica de transfer a unui filtru RC trece jos*

Funcția de transfer al filtrului analogic poate fi discretizată utilizând transformata Z. Dacă aplicăm transformarea biliniară între s și z folosind perioada de eșantionare T_s:

$$s = \frac{2}{T_s}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{4.11}$$

vom obține funcția de transfer în planul Z:

$$\begin{aligned} H(z) &= \frac{1}{1 + RC\dfrac{2}{T_s}\left(\dfrac{1 - z^{-1}}{1 + z^{-1}}\right)} \\ &= \frac{1 + z^{-1}}{1 + z^{-1} + \dfrac{2}{T_s}RC - \dfrac{2}{T_s}RCz^{-1}} \\ &= \frac{1 + z^{-1}}{\left(1 + \dfrac{2}{T_s}RC\right) + \left(1 - \dfrac{2}{T_s}RC\right)z^{-1}} \\ &= \frac{\dfrac{1}{1 + \dfrac{2}{T_s}RC} + \dfrac{1}{1 + \dfrac{2}{T_s}RC}z^{-1}}{1 + \dfrac{1 - \dfrac{2}{T_s}RC}{1 + \dfrac{2}{T_s}RC}z^{-1}} \end{aligned} \tag{4.12}$$

Echivalând cu ecuația 4.9 și înlocuind $\frac{1}{T_s} = f_s$ pentru simplitate, obținem coeficienții:

$$b_1 = b_2 = \frac{1}{1 + 2f_sRC} \quad a_2 = \frac{1 - 2f_sRC}{1 + 2f_sRC} \tag{4.13}$$

Dacă de exemplu considerăm un filtru RC cu rezistența de 150 kΩ respectiv condensatorul de 100 nF (valori tipice pentru aceste componente), obținem o frecvență de tăiere $f_0 = \frac{1}{2\pi RC} \approx 10.61$ Hz. Echivalentul digital al acestui filtru va avea coeficienții $b_1 = b_2 \approx 4.1494 \cdot 10^{-3}$ respectiv $a_2 \approx -0.9917$, pentru o frecvență de eșantionare de 8 kHz. Putem vizualiza funcția de transfer a acestui filtru cu codul Matlab (rezultând în figura 4.15):

```
>> b = [4.1494e-3 4.1494e-3]; a = [1, -0.9917];
>> w = logspace(-1,3,1023);
>> h = 20*log10(freqz(b, a, w, 8e3));
>> semilogx(w, h)
```

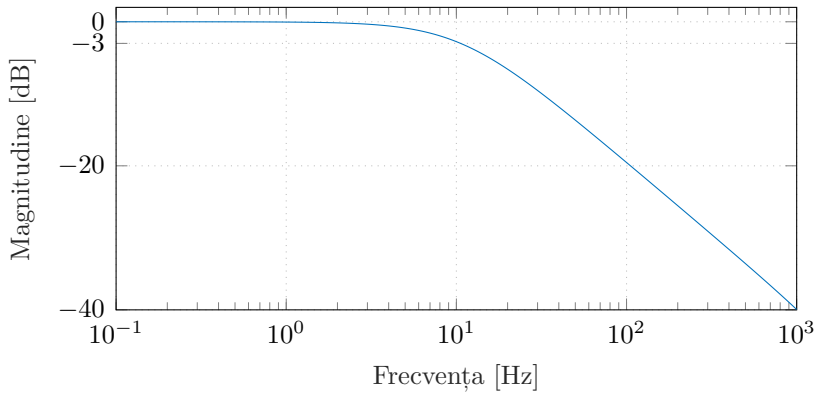


Figura 4.15. *Caracteristica de transfer a unui filtru trece jos cu parametri* $R = 150\,\text{k}\Omega$ *și* $C = 100\,\text{nF}$

4.3. Filtrarea în Matlab

Filtrarea unui semnal se realizează în Matlab cu funcția `filter`:

```
>> y = filter(b, a, x);
```

În acest caz `x` este vectorul ce conține semnalul de la intrare, iar `b` și `a` sunt doi vectori ce conțin coeficienții filtrului. Dacă vrem să realizăm o filtrare cu o funcție de transfer de lungime finită (adică o singură convoluție) putem folosi următorul apel al funcției `filter`:

```
>> y = filter(h, 1, x);
```

în care folosim 1 în loc de `a`, deoarece prin convenție coeficientul `a(1)` (care nu apare în formula de calcul a convoluției din reacție) se folosește pentru normarea rezultatului.

Pentru exemplificarea acestei funcții, putem considera filtrul RC prezentat în capitolul anterior. Pornind de la un semnal de frecvență joasă (de exemplu de 1 Hz) peste care este suprapusă o interferență de frecvență înaltă, putem observa efectul filtrării (figura 4.16) aplicând următoarea secvență Matlab:

```
>> fs = 8000;
>> t = 0:1/fs:1;
>> x = sin(2*pi*t);
>> tz = 0:1/fs:0.1;
>> z = 0.5*hamming(length(tz))'.*sin(2*pi*200*tz);
>> x(length(x)/2-length(z)/2:length(x)/2+length(z)/2-1) += z;
>> subplot(2,1,1), plot(t,x)
>> b = [4.1494e-3 4.1494e-3]; a = [1, -0.9917];
>> y = filter(b,a,x);
>> subplot(2,1,2), plot(t,y)
```

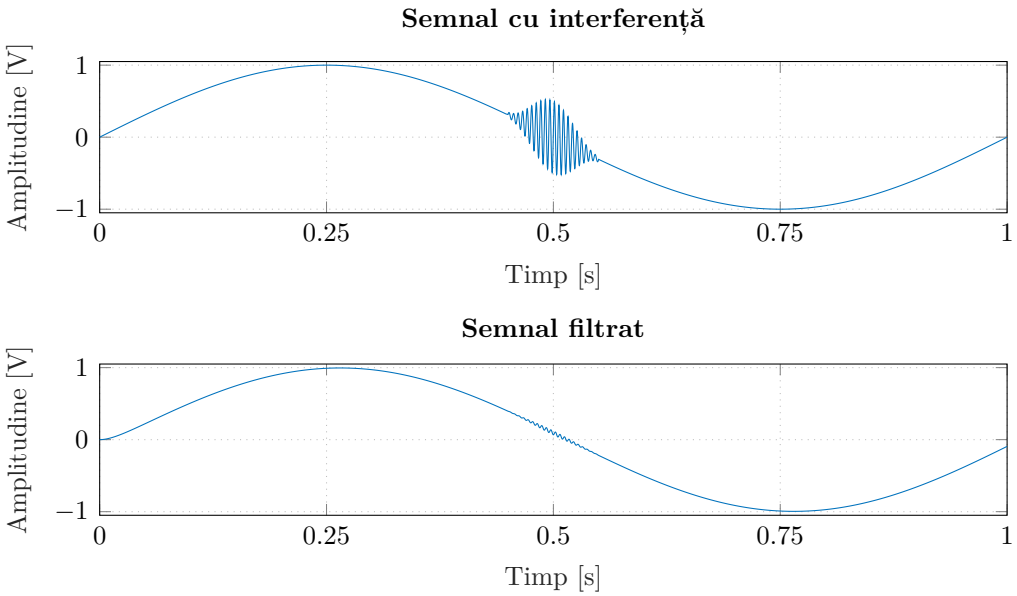


Figura 4.16. *Rezultatul filtrării unui semnal de 1 Hz având un zgomot de interferență*

Se poate observa pe semnalul rezultat că operația de filtrare introduce o oarecare defazaj față de semnalul original. Acest lucru se datorează faptului că pentru a satisface proprietatea de cauzalitate a operației de filtrare, funcția de răspuns a filtrului ideal (simetric față de origine) este deplasată pe axa Ox astfel încât să nu existe coeficienți la momente de timp negative. Un filtru este cauzal dacă semnalul de ieșire apare după aplicarea semnalului de la intrare, adică este îndeplinită condiția $h(t) = 0, pentrut < 0$.

Având în vedere însă că pentru filtrarea semnalului în Matlab deja dispunem de tot vectorul de semnal, putem aplica și o filtrare non-cauzală cu defazaj zero prin aplicarea

filtrului pe semnalul de la intrare pe direcția normală și încă o dată pe direcția inversă de timp. Pentru a realiza acest lucru se folosește funcția `filtfilt`:

```
>>> y = filtfilt(b,a,x)
```

Folosind această linie în exemplul precedent, rezultă în figura 4.17, pe care putem observa că rezultatul filtrării rămâne în fază cu semnalul original.

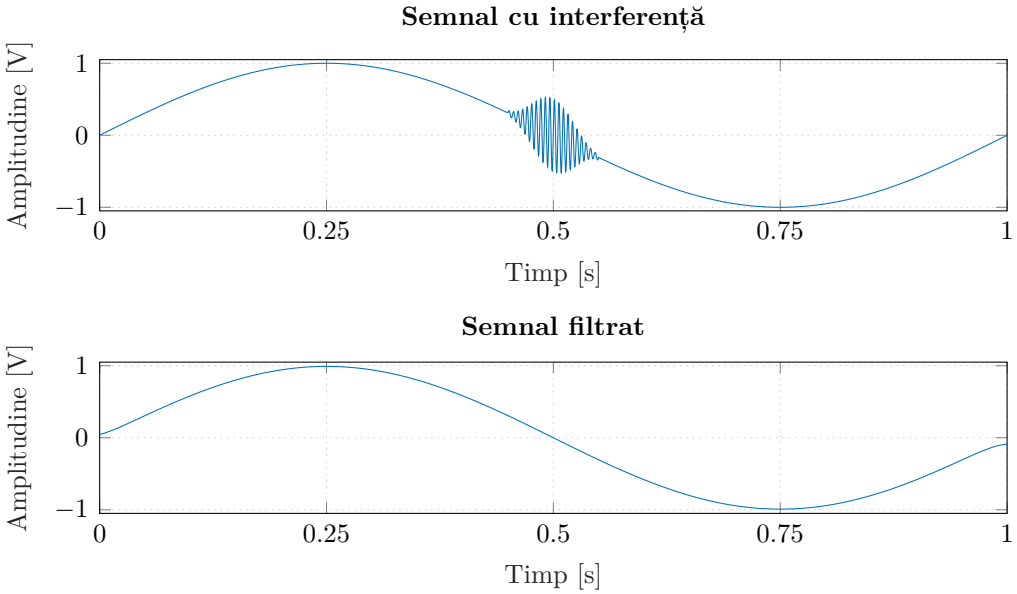


Figura 4.17. *Rezultatul filtrării cu defazaj zero*

4.4. Exerciții

1. Implementați o funcție de filtrare cu formatul `y = myfilter(b, a, x)` și comparați rezultatele cu cele obținute folosind funcția matlab `filter`.

2. Observați efectul filtrării asupra unui semnal folosind un filtru notch (filtru oprește bandă, folosit frecvent pentru eliminarea componentei de 50Hz din semnale cauzate de interferențe electromagnetice cu rețeaua de alimentare). Ca și semnal de test puteți folosi semnalul generat la lucrarea 1, iar pentru coeficienții filtrului folosiți următoarele valori:

```
>>> b = [ 0.9859  -1.9703   0.9859 ];
>>> a = [ 1.0000  -1.9703   0.9718 ];
```

Lucrarea 5

Proiectarea filtrelor FIR

Filtrele FIR (Finite Impulse Response) sunt filtre implementate sub forma unui sistem liniar invariant în timp cu funcție de transfer de lungime finită. Așa cum am văzut în lucrarea precedentă, aceste filtre sunt obținute pornind de la un filtru ideal printr-o limitare în timp a funcției de transfer. Filtrul ideal are o funcție de transfer de tip sinus cardinal, cu o lungime infinită. Pentru limitare în timp se aplică o fereastră de lungime finită, care însă va rezulta în prezența unor ripluri în banda de trecere, respectiv în banda de tăiere, așa cum se vede pe figura 5.1.

Aceste ripluri determină o deviere (δ_p respectiv δ_s) față de răspunsul ideal, adică în banda de trecere putem avea o atenuare/amplificare de până la δ_p a semnalului util, iar în banda de tăiere poate să rămână parțial semnalul nedorit.

De asemenea, tranziția între banda de trecere și banda de tăiere nu este instantanee, ci există o bandă de tranziție între ω_p și ω_s, în care componentele utile sunt tăiate prea mult, componentele nedorite sunt tăiate prea puțin. Semnalul de la intrare nu ar trebui să conțină componente în această bandă de tranziție pentru a avea o filtrare eficientă.

Pentru a proiecta un filtru FIR, trebuie să găsim un compromis acceptabil între

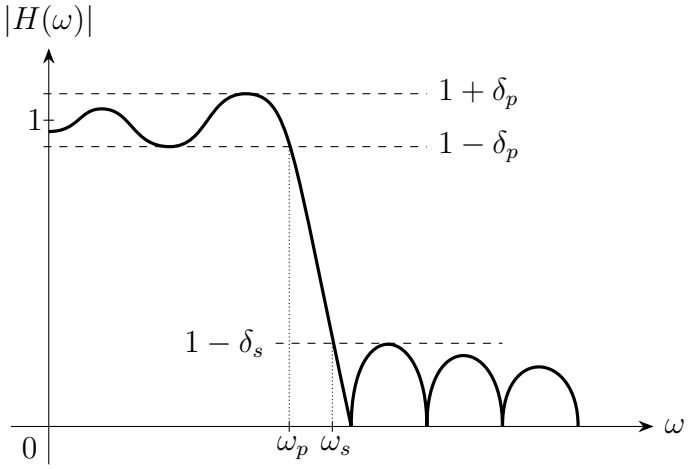


Figura 5.1. *Funcția de transfer a unui filtru trece jos real*

mărimea riplurilor și lățimea benzii de tranziție, ținând cont de componentele semnalului pe care dorim să le filtrăm. Pentru a îmbunătăți răspunsul, putem recurge la două metode de proiectare:

- folosirea unei ferestre, care oferă o trecere mai lină către 0 a semnalului, reducând astfel riplurile rezultate de fenomenul Gibbs (oscilații spectrale în cazul unor treceri bruște din 1 în 0 a semnalului din domeniul timp), folosind diferite tipuri de ferestre se poate găsi compromisul bun între mărimea riplurilor și forma răspunsul în banda de tranziție
- folosirea unui algoritm de optimizare pentru a reduce riplurile în benzile de trecere și tăiere, ignorând răspunsul în banda de tranziție

5.1. Filtre FIR bazate pe ferestre

Pentru a proiecta un filtru FIR de formă standard în Matlab se folosește funcția `fir1`. Aceasta primește ca parametri ordinul filtrului, frecvența de tăiere și opțional tipul filtrului, respectiv fereastra folosită. Sintaxa funcției este:

```
>> h = fir1(N, Wn, type, window);
```

Parametrii sunt:

- N — ordinul filtrului, specifică numărul de poli în funcția de transfer a filtrului, în cazul filtrelor digitale aceasta se manifestă în lungimea vectorului de coeficienți h care va avea $N + 1$ elemente
- *Wn* — frecvența de tăiere normalizată la frecvența maximă (jumătatea frecvenței de eșantionare); acest parametru poate fi și un vector cu mai multe frecvențe de tăiere (obligatoriu în ordine crescătoare), caz în care filtrul creat va fi de tip trece bandă sau oprește bandă
- *type* — un șir de caractere care specifică tipul filtrului, poate fi una din `'low'` (filtru trece jos), `'high'` (filtru trece sus), `'bandpass'` (filtru trece bandă) sau `'stop'`(filtru oprește bandă); dacă acest parametru lipsește atunci se va crea implicit un filtru trece jos în cazul în care *Wn* este scalar sau filtru trece bandă în cazul în care *Wn* este vector
- *window* — fereastra folosită pentru proiectarea filtrului, trebuie să fie un vector de lungime $N + 1$; ferestre uzuale pot fi create cu funcțiile `boxcar`, `bartlett`, `hamming`, `hann`, `kaiser`, `chebwin` sau `blackman`; dacă acest parametru nu este specificat, atunci `fir1` va folosi o fereastră Hamming creat implicit pentru ordinul filtrului specificat

De exemplu, pentru a crea un filtru trece jos, cu ordinul 64 și frecvența de tăiere la 25 % din jumătatea frecvenței de eșantionare, vom folosi comanda:

```
>> h = fir1(64, 0.25);
```

Aceasta va rezulta într-un vector de 65 de elemente, care conține caracteristica de sinus cardinal de lungime finită, așa cum se vede pe figura 5.2.

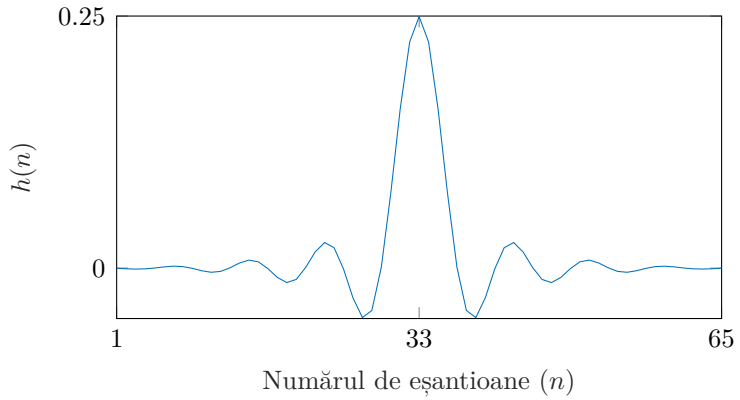


Figura 5.2. *Funcția de transfer în timp a unui filtru FIR*

Răspunsul în frecvență a acestui filtru putem afla cu comanda :

```
>> freqz(h);
```

rezultând în caracteristicile de magnitudine și fază de pe figura 5.3.

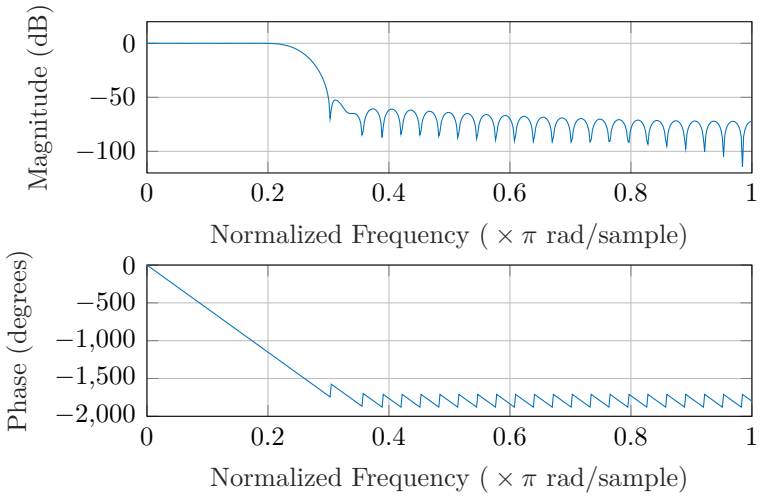


Figura 5.3. *Răspunsul în frecvență a filtrului de pe figura 5.2*

Așa cum se vede și pe caracteristică, filtrele FIR sunt caracterizate de un răspuns în fază liniar, care în majoritatea cazurilor poate fi ignorat, analiza acestor filtre impunând mai mult studierea caracteristicii în magnitudine.

Putem vizualiza doar această caracteristică, dacă preluăm rezultatele funcției `freqz` și afișăm doar magnitudinea spectrală:

```
>> [m, w] = freqz(h);
>> plot(w, abs(m));
```

Pe figura 5.4 se vede răspunsul pe o scară liniară.Se poate observa riplurile în banda de trecere și panta răspunsului din banda de tranziție.

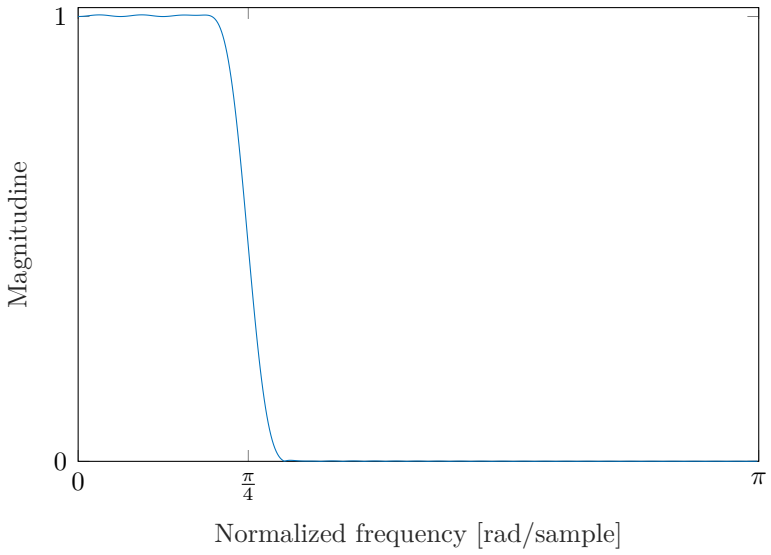


Figura 5.4. *Magnitudinea răspunsului în frecvență a filtrului de pe figura 5.2*

Banda de tăiere poate fi observată mai greu pe o scară liniară din cauza amplitudinilor mici. Pentru o mai bună vizualizare a acestora putem opta pentru o scară logaritmică, exprimând magnitudinea în decibeli:

```
>> plot(w, 20*log10(abs(m)));
```

Figura 5.5 prezintă răspunsul pe o scară logaritmică, pe care se pot observa și riplurile în banda de tăiere, riplul maxim fiind în jur de $-50\,\text{dB}$. Pe de altă parte, riplurile din banda de trecere sunt acum mai greu de observat. De asemenea, la interpretarea pantei în banda de tranziție trebuie să ținem cont și de curbarea caracteristicii datorită reprezentării logaritmice.

În ambele cazuri, caracteristica a fost reprezentată în funcție de pulsația normalizată pe intervalul $(0, \pi)$, exprimat în radiani pe eșantion. Filtrele sunt proiectate folosind frecvențe relative, normalizate la jumătatea frecvenței de eșantionare.

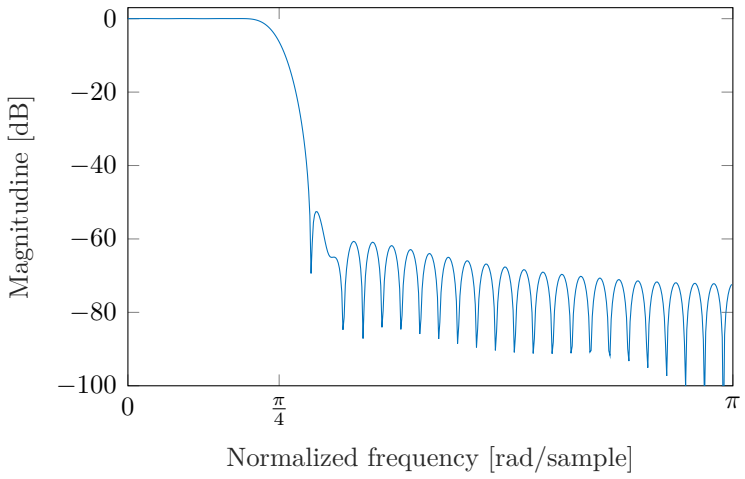


Figura 5.5. *Magnitudinea răspunsului în frecvență a filtrului pe scară logaritmică*

Celelalte tipuri de filtre, adică filtrul trece sus (fig. 5.6), filtrul trece bandă (fig. 5.7) respectiv oprește bandă (fig. 5.8) putem proiecta aplicând și al treilea parametru opțional la funcția `fir1`. De exemplu, pentru a crea niște filtre cu caracteristici similare cu cel prezentat anterior (banda de 25 percent din frecvența maximă și ordinul de 64) cu următoarele comenzi:

```
>> % trece sus
>> h_highpass = fir1(64, 0.75, 'high');
>> % trece bandă
>> h_bandpass = fir1(64, [0.25, 0.5]);
>> % oprește bandă
>> h_bandstop = fir1(64, [0.25, 0.5], 'stop');
```

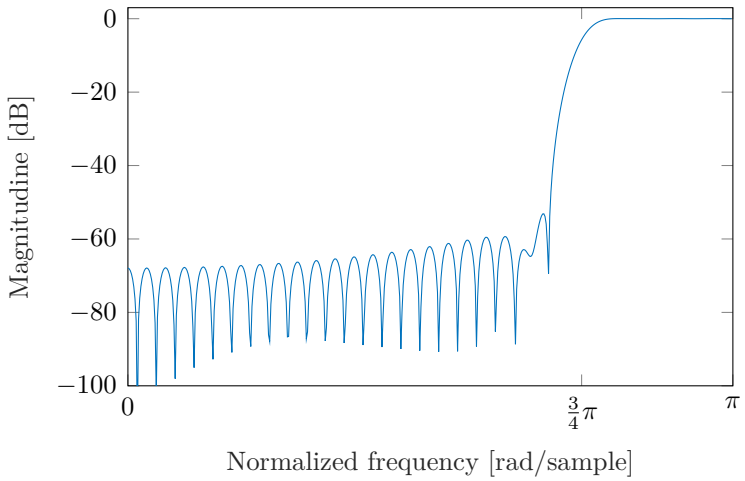


Figura 5.6. *Răspunsul în frecvență a unui filtru trece sus*

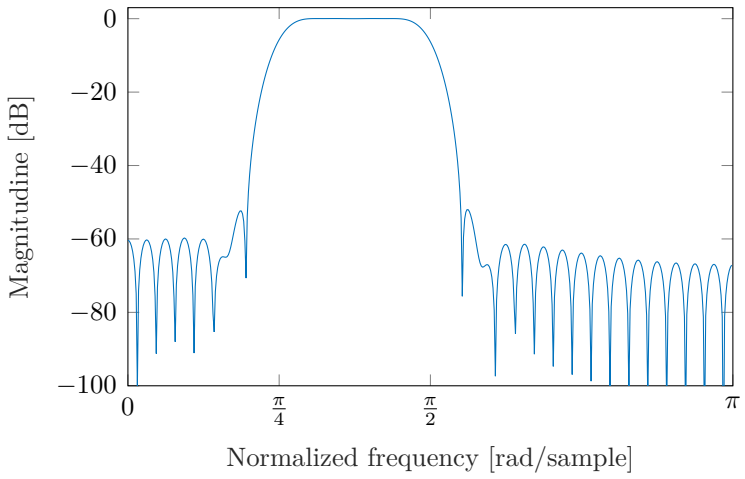


Figura 5.7. *Răspunsul în frecvență a unui filtru trece bandă*

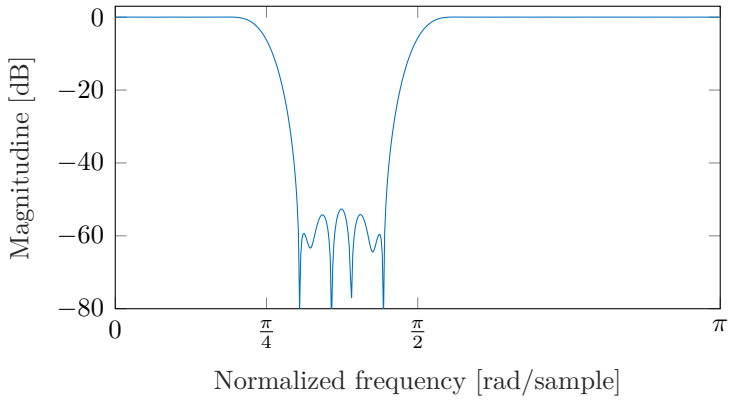


Figura 5.8. *Răspunsul în frecvență a unui filtru oprește bandă*

Pe lângă filtrele obișnuite (trece-jos, trece-sus, trece-bandă, oprește-bandă), se poate construi și filtre cu răspuns în frecvență arbitrar. Pentru a realiza acest lucru în Matlab se folosește funcția `fir2` cu următoarea sintaxă:

```
>> fir2(N, F, M)
```

Parametrii acestuia sunt:

- N — ordinul filtrului, la fel ca la `fir1`
- F — vector cu frecvențe normalizate, pentru care se dorește definirea unui răspuns, acest vector va avea valorile între 0 și 1, așezate în ordine crescătoare, valoarea 0 și 1 trebuie să fie prezent obligatoriu
- M — valorile din răspunsul dorit corespunzătoare punctelor de frecvență definite în F

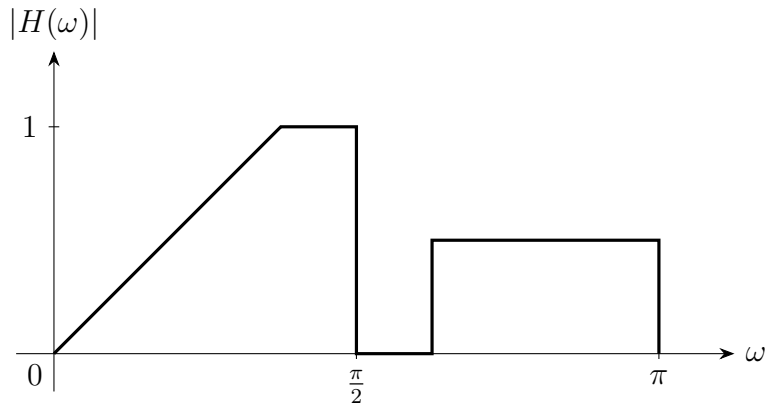


Figura 5.9. *Exemplu de un răspuns în frecvență arbitrar*

De exemplu, dacă vrem să obținem un răspuns în spectru de forma prezentată pe figura 5.9, putem să folosim următoarea secvență de cod:

```
>>> h = fir2(64, [0,3/8,4/8,4/8+1/1024,5/8-1/1024,5/8,1],
    ↪  [0,1,1,0,0,0.5,0.5]);
>>> [m, w] = freqz(h);
>>> plot(w, (abs(m)));
```

ceea ce rezultă în figura 5.10.

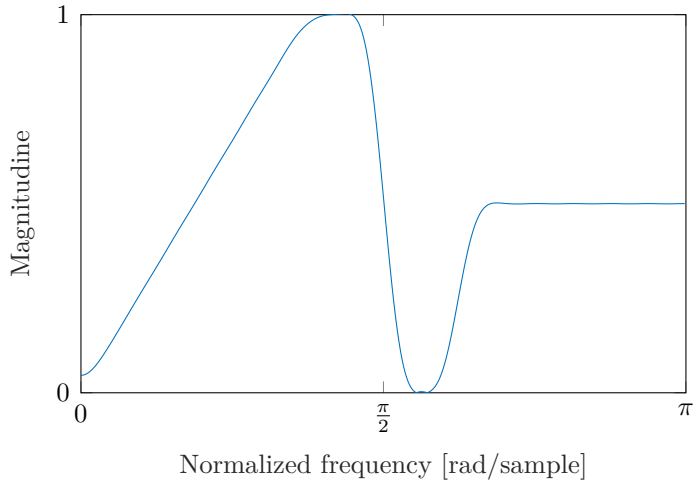


Figura 5.10. *Răspunsul în frecvență proiectat a unui filtru arbitrar*

5.2. Filtrul optim

Filtrele bazate pe ferestre lasă o deviație destul de mare față de forma dorită a răspunsului în frecvență din cauza riplurilor și a benzii de tranziție prezente. Putem să optăm însă pentru aplicarea unui algoritm iterativ de optimizare pentru a minimiza eroarea față de răspunsul dorit.

Considerând funcția de răspuns în frecvență de forma:

$$H(e^{j\omega}) = e^{-jN\omega/2} e^{j\beta} \breve{H}(\omega) \tag{5.1}$$

unde $\breve{H}(\omega)$ este funcția de răspuns reală, cu care putem defini funcția de eroare ponderată față de răspunsul dorit $D(\omega)$ cu:

$$\varepsilon(\omega) = W(\omega) \left[\breve{H}(\omega) - D(\omega) \right] \tag{5.2}$$

Funcția pondere $W(\omega)$ putem defini în funcție de amplitudinea maximă permisă a riplurilor în banda de trecere δ_p și în banda de tăiere δ_s sub forma:

$$W(\omega) = \begin{cases} 1, & \text{în banda de trecere} \\ \frac{\delta_p}{\delta_s}, & \text{în banda de tăiere} \end{cases} \tag{5.3}$$

Realizarea unui filtru optim poate fi făcută prin găsirea unui set de coeficienți care să conducă la minimizarea valorilor maxime ale funcției de eroare de-a lungul benzilor de trecere și de tăiere. Soluția acestei probleme este oferită de Parks și McClellan și este implementată și în Matlab prin funcția `firpm`. Aceasta are următoarea sintaxă (similar cu `fir2`):

```
>> firpm(N, F, M)
```

sau opțional:

```
>> firpm(N, F, M, W)
```

Parametrii acestei funcții sunt:

- N — ordinul filtrului, la fel cu `fir2`
- F — vector cu frecvențe normalizate, similar cu `fir2`, un vector care va avea valori între 0 și 1 inclusiv, în ordine crescătoare, diferența față de `fir2` este că în acest caz numărul elementelor din vector trebuie să fie obligatoriu un număr par, fiecare pereche de frecvență specificând defapt marginile unor benzi (de tăiere sau de trecere) între care se face optimizarea funcției de eroare, benzile rămase între aceste perechi e considerată bandă de tranziție și este ignorată de algoritmul de optimizare

- M — magnitudinile corespunzătoare frecvențelor enumerate în parametrul F, în benzile de interese (tăiere sau trecere) se consideră interpolarea liniară între magnitudinile specificate pentru cele două margini ale benzi, în benzile de tranziție magnitudinea este ignorată

- W — coeficienții funcției de pondere pentru fiecare bandă de interes, acest vector are jumătatea numărului de elemente din vectorul de frecvențe, și conține coeficientul considerat pentru fiecare pereche de frecvență, când lipsește acest parametru, se consideră un filtru echiriplu, adică ripluri permise egale atât în banda de trecere cât și în banda de tăiere

De exemplu, dacă vrem să construim un filtru echiriplu trece jos, de ordinul 15, cu banda de trecere până la $\frac{\pi}{4}$ și banda de tăiere peste $\frac{3\pi}{8}$, putem crea cu codul:

```
>> N = 15; F = [0 1/4 3/8 1]; M = [1 1 0 0]; W = [1 1];
>> h = firpm(N, F, M, W);
>> [m f] = freqz(h);
>> plot(F*pi, M, f, real(m.*exp(j*f*(N+1)/2)))
>> axis([0 pi -0.1 1.1])
```

În urma acestor comenzi obținem graficul de pe figura 5.11. Putem observa riplurile maxime de aceeași amplitudine și în banda de trecere și în banda de tăiere.

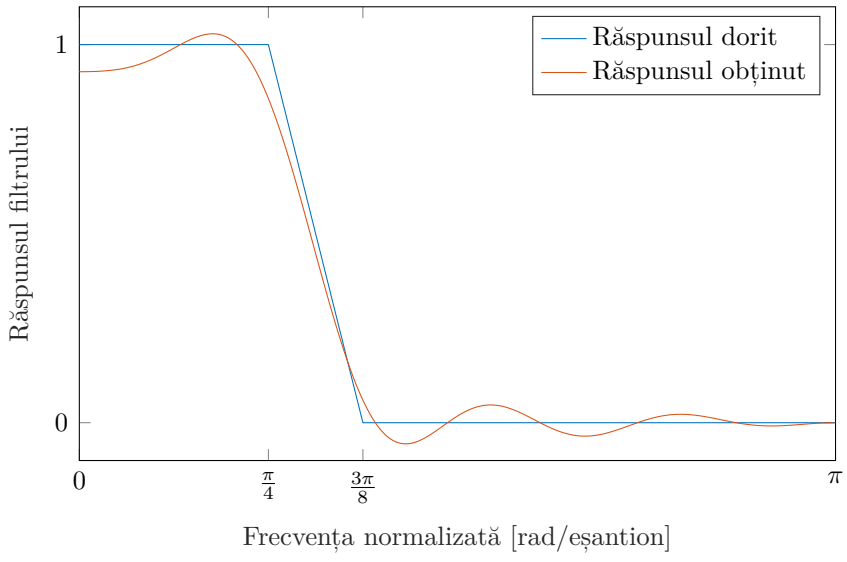


Figura 5.11. *Răspunsul în frecvență a unui filtru optim*

5.3. Exerciții

1. Studiați răspunsurile în frecvență a unor filtre FIR create cu ajutorul funcțiilor `fir1`, `fir2` și `firpm` cu diferite combinații de ordin a filtrului, lățimea de bandă și fereastră folosită pentru toate formele de răspuns (trace jos, trece sus, trece bandă, oprește bandă, multi-bandă).

2. Separați semnalul generat la lucrarea 1 în cele două componente alcătuitoare, prin proiectarea și folosirea unui filtru trece jos care păstrează componenta de 50 Hz și a unui filtru trece sus care păstrează componenta de 220 Hz.

3. Separați semnalul de 300 Hz din semnalul stocat în fișierul `s.mat`  (frecvența de eșantionare pentru acest semnal fiind de 8 kHz). Vizualizați semnalul înainte și după implementarea operației de filtrare.

Lucrarea 6

Proiectarea filtrelor IIR

Filtrele IIR (Infinite Impulse Response) sunt filtre recursive construite din două blocuri LIT conform figurii 6.1.

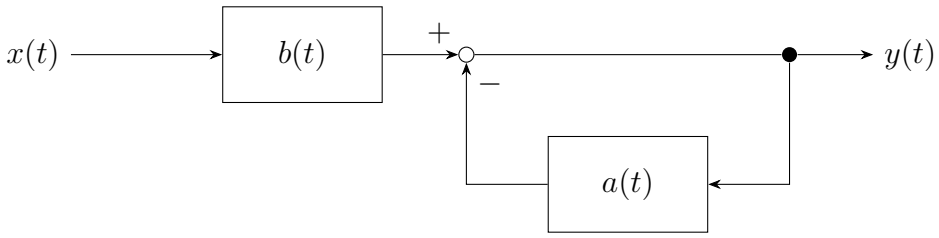


Figura 6.1. *Filtru IIR cu reacție*

Aceste filtre pot fi proiectate pornind de la corespondentul analogic (filtre Butterworth sau Chebyshev) definite prin funcția de răspuns în frecvență:

$$H(s) = \frac{P(s)}{D(s)} \tag{6.1}$$

Prin aplicarea transformatei biliniară (descrise în capitolul 4) putem obține transformata z a funcției de transfer, care poate fi rescrisă sub forma:

$$H(z) = \frac{b_1 + b_2 z^{-1} + \cdots + b_{N-1} z^{-N}}{1 + a_2 z^{-1} + \cdots + a_{N-1} z^{-N}} \tag{6.2}$$

din care se pot extrage coeficienții b și a ai celor două sisteme LIT și care pot fi folosiți mai departe în operația de filtrare cu funcția `filter(b,a,x)` din Matlab.

Ordinul N a filtrului IIR este dat de numărul de poli sau zerouri din funcția de transfer, și similar cu filtrele FIR, în cazul filtrelor digitale vom avea vectorii b și a de lungime $N + 1$.

Răspunsul în timp a acestor filtre tinde spre infinit datorită reacției în sistem, ceea ce permite să obținem un răspuns dorit cu un ordin mic. Artefactele (ripluri și interval de tranziție) prezentate în cazul filtrelor FIR (figura 5.1) sunt prezente și la filtrele IIR,

dar putem obține valori similare cu filtre FIR folosind un ordin mult mai mic. Asta înseamnă că filtrarea cu un filtru IIR poate fi realizată mult mai eficient computațional decât cu un filtru FIR.

Pe de altă parte, un dezavantaj al filtrelor IIR sunt problemele de stabilitate. Spre deosebire de filtrele FIR, filtrele IIR pot deveni instabile datorită reacției în sistem.

6.1. Tipuri de filtre IIR

În funcție de caracteristicile răspunsului în frecvență, mai ales în ce privește liniaritatea, respectiv lipsa sau prezența riplurilor, filtrele IIR pot fi categorizate în filtre Butterworth, filtre Chebyshev (tipul I și tipul II) și filtre eliptice.

Filtrul Butterworth

Filtrele de tip Butterworth sunt caracterizate prin faptul că ele oferă un răspuns liniar (fără ripluri semnificative) în banda de bază și banda de tăiere, în același timp însă prezintă o bandă de tranziție foarte largă. Răspunsul acestor filtre se aseamănă cu cel al filtrelor FIR, având nevoie însă de un ordin mult mai mic pentru a atinge aceleași caracteristici.

Un filtru Butterworth poate fi proiectat în Matlab cu funcția `butter`:

```
>> [b a] = butter(N, Wn, type);
```

Parametrii sunt asemănători cu cele ale funcției `fir1`, astfel:

- N — ordinul filtrului, specifică numărul de poli și zerouri în funcția de transfer, atât vectorul de coeficienți b cât și vectorul de coeficienți a vor avea $N+1$ elemente
- *Wn* — frecvența de tăiere normalizată la frecvența maximă (jumătatea frecvenței de eșantionare), acest parametru poate fi și un vector cu două frecvențe de tăiere, caz în care filtrul generat va fi de tip trece bandă sau oprește bandă, nu se permite creare filtrelor cu mai multe benzi de trecere/tăiere
- *type* — opțional, un șir de caractere care specifică tipul filtrului, poate lua valorile `'low'` (filtru trece jos), `'high'` (filtru trece sus), `'bandpass'` (filtru trece bandă) sau `'stop'` (filtru oprește bandă); dacă acest parametru lipsește atunci se creează un filtru trece jos sau trece bandă (în funcție de conținutul parametrului *Wn*

De exemplu pentru a crea un filtru IIR trece jos de tip Butterworth de ordinul 5 și frecvența de tăiere la 25 % din jumătatea frecvenței de eșantioane putem folosi de următoarea comandă:

```
>> [b, a] = butter(5, 0.25);
```

Analizând răspunsul în frecvență cu comanda

```
>> freqz(b,a);
```

obținem figura 6.2.

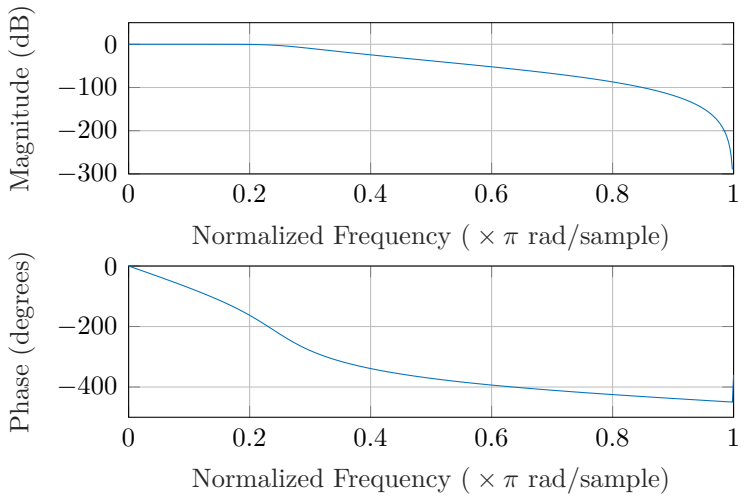


Figura 6.2. *Răspunsul în frecvență a unui filtru Butterworth*

Primul lucrul pe care putem observa la răspunsul filtrului Butterworth este că, spre deosebire de filtrele FIR, aceasta nu mai are un răspuns în fază liniară. Mai ales în zona de trecere dintre banda de tăiere și banda de trecere putem observa schimbări mari de liniaritate și în răspunsul în fază. Defapt, acest lucru este caracteristic la toate filtrele IIR, nici un filtru cu reacție nu va avea fază liniară. Condiția pentru a avea un filtru cu răspuns liniar este ca polii filtrului să fie așezați simetric față de cercul unitate, deci să avem poli și în interiorul cercului unitate și în afară. Pe de altă parte condiția de stabilitate a filtrului impune ca toți polii să se afle în interiorul cercului unitate, un pol în afara cercului invariabil conduce la un filtru instabil.

Răspunsul în magnitudine a filtrului Butterworth prezintă însă o liniaritate foarte bună în banda de trecere, și considerând o scară logaritmică și în banda de tăiere. Aceasta însă înseamnă o tranziție între banda de trecere și banda de tăiere destul de largă. Dacă analizăm răspunsul în magnitudine pe o scară liniară putem observa mai clar și această bandă de tranziție. Pe figura 6.3 se poate vedea răspunsul pe scară liniară a filtrului din exemplul precedent. Putem observa frecvența de tăiere, care prin convenție este considerată frecvența la care răspunsul ajunge la $-3\,\text{dB}$, echivalent cu $\frac{\sqrt{2}}{2}$ din amplitudinea sau jumătate din puterea semnalului.

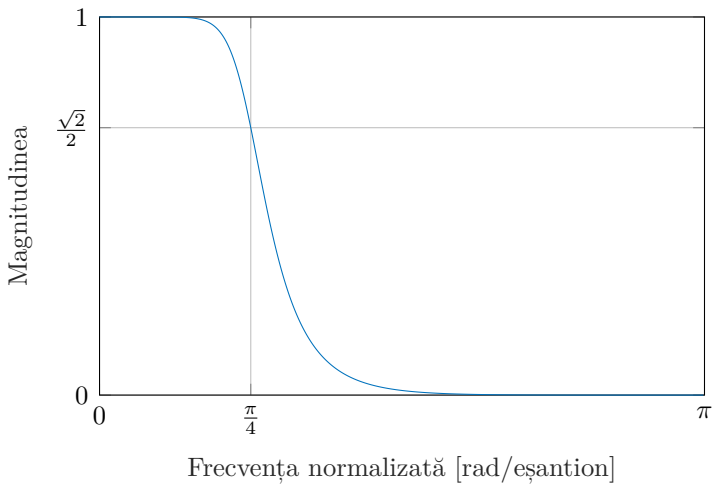


Figura 6.3. *Răspunsul în frecvență a unui filtru Butterworth pe scară liniară*

Filtrele Chebyshev

Spre deosebire de filtrele Butterworth, filtrele bazate pe polinoamele Chebyshev permit prezența unor ripluri, fie în banda de trecere fie în banda de tăiere. În funcție de banda în care sunt prezente aceste ripluri vorbim despre filtre Chebyshev de tipul I (cu ripluri în banda de trecere) sau de filtre Chebyshev de tipul II (cu ripluri în banda de tăiere).

Aceste filtre pot fi create în Matlab cu funcțiile `cheby1` și `cheby2` având următoarea sintaxă:

```
>> [b, a] = cheby1(N, Rp, Wn, type);
>> [b, a] = cheby2(N, Rs, Wn, type);
```

Parametri *N*, *Wn* și *type* sunt aceleași ca la funcția `butter`, reprezentând ordinul filtrului, frecvența de tăiere normalizată și tipul filtrului (asta din urmă fiind opțional). Diferența față de filtre Butterworth este dată de parametri:

- *Rp* — riplul maxim în banda de trecere, parametrul definește diferența vârf la vârf a riplurilor în banda de trecere, exprimat în dB; prin convenție pentru a putea considera bandă de trecere, riplurile în această bandă trebuie să fie mai mici de 3 dB
- *Rs* — riplul maxim în banda de tăiere, acest parametru, exprimat tot în dB, definește atenuarea riplului maxim față de vârful benzii de trecere; prin convenție, pentru a putea considera bandă de tăiere, această atenuare trebuie să fie de cel puțin 3 dB

Câte un exemplu pentru astfel de filtre poate fi văzut pe figura 6.4 creat cu comenzile:

```
>> [b, a] = cheby1(5, 1, 0.25);
>> figure; freqz(b,a);
>> [b, a] = cheby2(5, 20, 0.25);
>> figure; freqz(b,a);
```

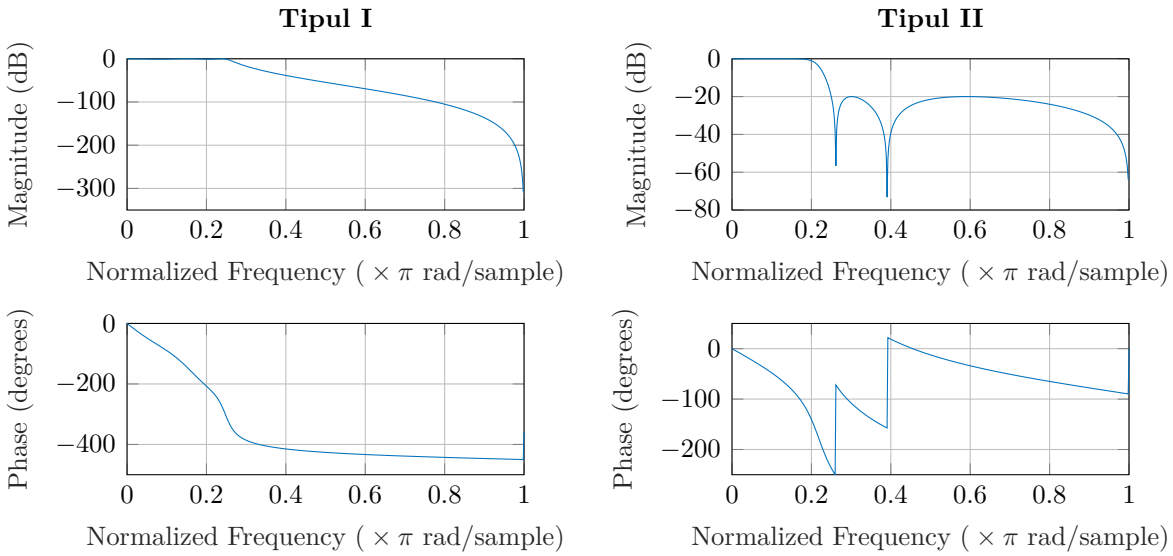


Figura 6.4. *Răspunsul în frecvență a filtrelor Chebyshev tipul I (a) și tipul II (b)*

Datorită rezoluției graficului cu răspunsul filtrului Chebyshev de tipul I, riplurile din banda de trecere sunt mai greu de observat, dar pe un grafic cu scară liniară sunt mai clar vizibile. Riplul de 1 dB specificat în exemplul precedent corespunde la aproximativ 89 % din semnal. Pe figura 6.5 este prezentat răspunsul acestui filtru pe o scară liniară.

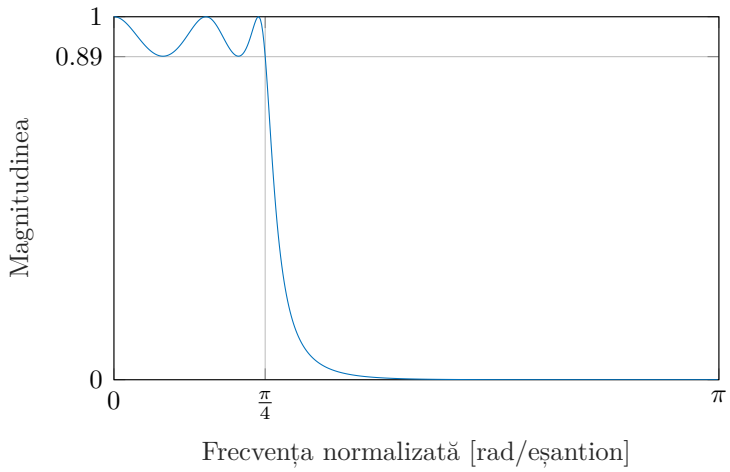


Figura 6.5. *Răspunsul în frecvență a unui filtru Chebyshev de tipul I pe scară liniară*

Se poate observa și o caracteristică specifică funcțiilor `cheby1` respectiv `cheby2` din Matlab, și anume faptul că parametrul *Wn* care precizează frecvența de tăiere a filtrului proiectat, de fapt precizează frecvența la care răspunsul filtrului părăsește riplul maxim precizat de parametrul *Rp* în cazul funcției `cheby1` respectiv când ajunge la riplul maxim precizat de parametrul *Rs* în cazul funcției `cheby2`. Această caracteristică indică o deviere față de convenția pentru frecvența de tăiere de −3 dB și trebuie avut grijă la alegerea corectă a frecvenței de tăiere pentru a obține răspunsul dorit în frecvență.

Filtrul eliptic

Filtrul eliptic este un filtru IIR cu ripluri atât în banda de trecere cât și în banda de tăiere. Datorită acestor ripluri acest filtru este caracterizat și de o bandă de tranziție foarte îngustă. Un filtru eliptic poate fi creat în Matlab cu funcția `ellip`, având următoarea sintaxă:

```
>> [b, a] = ellip(N, Rp, Rs, Wn, type);
```

Parametrii sunt aceeași ca la filtre Chebyshev de tipul I și II, dar putem observa prezența atât a parametrului *Rp* corespunzător riplului maxim în banda de trecere cât și *Rs* corespunzător riplului maxim în banda de tăiere. Acești doi parametri sunt precizați ca și la filtrele Chebyshev în decibeli față de valoarea de vârf din banda de trecere. Ceilalți parametri sunt cei obișnuiți, *N* ordinul filtrului care determină și numărul de coeficienți în vectorii rezultați, *Wn* frecvența de tăiere normalizată la jumătatea frecvenței de eșantionare și *type* tipul filtrului dorit.

De notat este faptul că, spre deosebire de filtrele Chebyshev, frecvența de tăiere specificată printre parametrii funcției `ellip`, precizează frecvența la care răspunsul ajunge la −3 dB, așa cum prevede convenția.

Un exemplu similar cu cele prezentate anterior—filtru trece jos de ordinul 5 cu frecvența de tăiere la 25 % din jumătatea frecvenței de eșantionare, riplul în banda de trecere de 1 dB și riplul în banda de tăiere de −20 dB, cu răspunsul în frecvență prezentat pe figura 6.6—poate fi obținut cu comanda:

```
>> [b a] = ellip(5, 1, 20, 0.25);
>> freqz(b,a)
```

Aspectul cel mai important ce putem nota analizând graficul cu răspunsul în frecvență este tranziția foarte abruptă între banda de trecere și banda de tranziție.

Pentru a vedea mai bine diferențele între diferite tipuri de filtre IIR, putem reprezenta răspunsul în frecvență pe aceeași grafic. Cele patru exemple anterioare sunt prezentate pe figura 6.7 pe o scară liniară. Se poate observa că filtrul Butterworth are banda de tranziție cea mai largă, filtrul eliptic are tranziția cea mai abruptă, iar filtrele Chebyshev se așează undeva între cele două. Se mai observă și faptul că filtrele Chebyshev au frecvența de tăiere convențională la −3 dB deplasate față de celelalte pentru

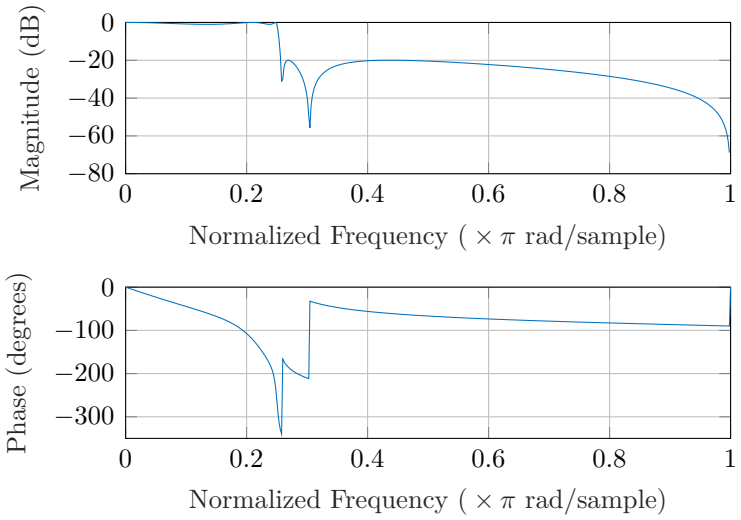


Figura 6.6. *Răspunsul în frecvență a unui filtru eliptic*

că funcția de proiectare a acestor filtre consideră valorile precizate pentru ripluri în loc de valoarea convențională.

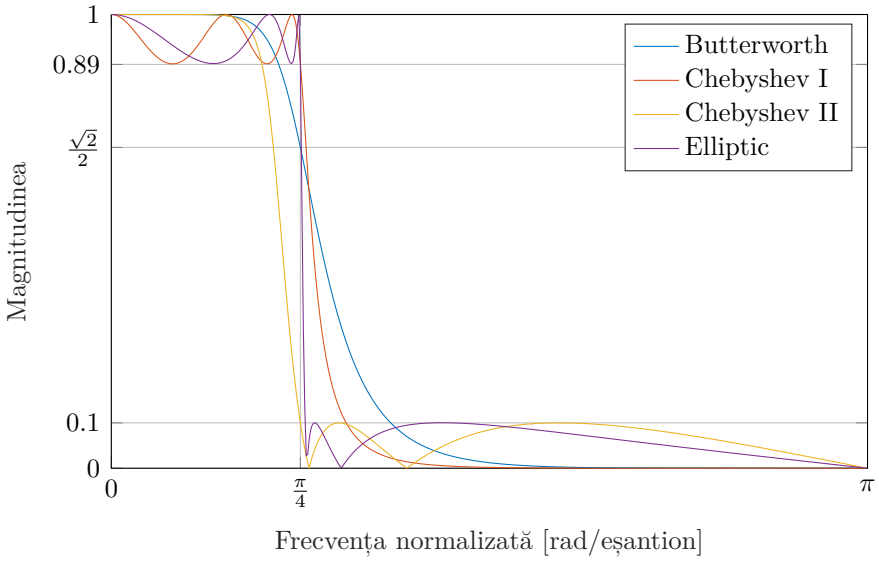


Figura 6.7. *Răspunsul în frecvență a unui filtrelor IIR*

6.2. Stabilitatea filtrelor IIR

O problemă semnificativă la proiectarea filtrelor IIR o reprezintă problemele de stabilitate. Datorită faptului că filtrele IIR sunt filtre cu reacție, trebuie să ne asigurăm că

reacția să fie negativă în orice situație pentru a avea un sistem stabil. Funcțiile matematice pentru proiectarea filtrelor Butterworth și Chebyshev în sine generează sisteme stabile. Din păcate, operațiile matematice discrete din calculator introduc o eroare de cuantizare care poate rezulta în filtre instabile.

Un exemplu elocvent pentru analiza efectelor de cuantizare asupra stabilității filtrelor obținem prin simulare numerică în Matlab a unei reprezentări pe virgulă fixă. Cu comenzile următoare putem simula o reprezentare a coeficienților unui filtru generat în exemplele anterioare pentru o arhitectură cu virgulă fixă pe 8.8 biți (8 biți partea întreagă, 8 biți partea fracțională):

```
>> [b a] = ellip(5, 1, 20, 0.25);
>> bx = round(b * 256) / 256;
>> ax = round(a * 256) / 256;
>> freqz(bx, ax);
```

Rezultatul acestor comenzi poate fi văzut pe figura 6.8. Răspunsul filtrului cu coeficienții cuantizați nu se mai încadrează în intervalul subunitar, ci anumite componente de frecvență vor fi amplificate semnificativ, ceea ce, la folosirea unui astfel de filtru pe un semnal, va conduce la saturația completă a semnalului la ieșire.

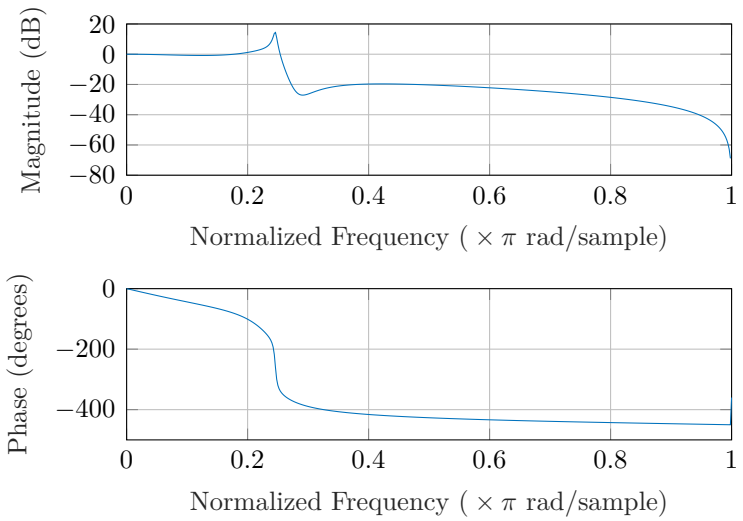


Figura 6.8. *Răspunsul în frecvență a unui filtru eliptic cu coeficienții cuantizați la 8.8 biți*

Problemele de stabilitate cauzate de erorile de cuantizare depind foarte mult și de gama dinamică a coeficienților. Cu cât e mare aceasta, cu atât avem nevoie de o rezoluție de cuantizare mai mare pentru a ține erorile sub limite acceptabile. În exemplul precedent coeficienții filtrului sunt aproximativ:

```
b =
    0.094328  -0.144354   0.099814   0.099814  -0.144354   0.094328
a =
    1.0000  -3.3561   5.3317  -4.6614   2.2611  -0.4758
```

Acești coeficienți, deși au o gamă dinamică mai mare decât ce poate fi reprezentat pe 8.8 biți, au totuși o gamă dinamică suficient de mică, pentru ca în cazul unei cuantizări de 16.16 biți, filtrul să rămână stabil. Dacă însă creștem ordinul filtrului, atunci obținem niște coeficienți a căror gamă dinamică să nu mai poată să fie reprezentată cu succes nici pe o cuantizare de 16.16 biți. Continuând exemplul precedent, un filtru eliptic cu parametri similari, dar de ordinul 15 va avea coeficienții aproximativ:

```
b =
   0.029304 -0.243243 1.015909 -2.731630 5.150885 -6.925690 6.253431
  -2.547461 -2.547461 6.253431 -6.925690 5.150885 -2.731630 1.015909
  -0.243243 0.029304
a =
   1.0000 -10.637 55.950 -191.26 472.82 -892.34 1325.2 -1574.7 1508.2
   -1163.9 717.86 -347.72 128.20 -34.029 5.8317 -0.48862
```

Pe figura 6.9 se vede rezultatul cuantizării la 16.16 biți a filtrului de ordinul 5 și de ordinul 15 suprapus cu răspunsul filtrului fără cuantizare.

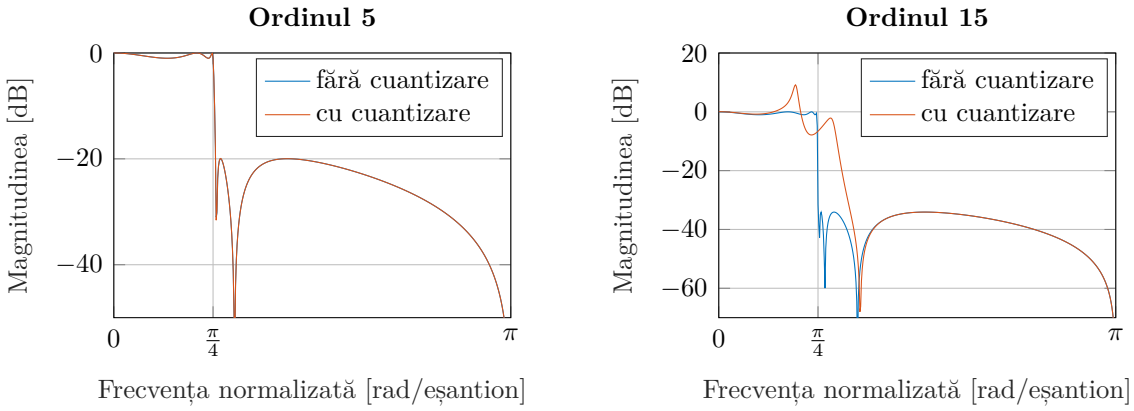


Figura 6.9. *Răspunsul în frecvență a unor filtre eliptice cu coeficienții cuantizați la 16.16 biți*

Având în vedere că și coeficienții cu care facem comparația sunt cuantizați, chiar dacă la virgulă mobilă cu dublă precizie, este clar că filtrele proiectate în Matlab în sine pot să fie instabile, dacă gama dinamică a coeficienților este una prea mare. Ordinul prea mare al filtrului, ripluri maxime mai mici sau banda de trecere prea îngustă poate rezulta într-un filtru instabil.

Un exemplu de filtrul eliptic de ordinul 15, ripluri la $\frac{1}{80}$ respectiv 80 dB și lățimea de bandă de 5 % din frecvența de eșantionare va avea un răspuns clar instabil și cu coeficienți reprezentați în virgulă mobilă, așa cum se vede pe figura 6.10.

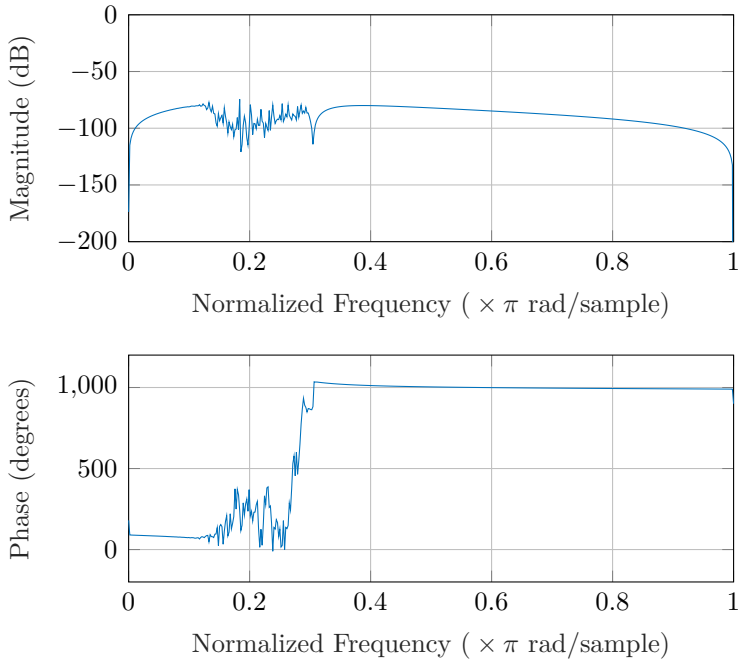


Figura 6.10. *Răspunsul în frecvență a unor filtre eliptice cu coeficienții cuantizați la 16.16 biți*

6.3. Exerciții

1. Studiați răspunsurile în frecvență a unor filtre IIR create cu ajutorul funcțiilor `butter`, `cheby1`, `cheby2` și `ellip`, cu diferite combinații de ordin a filtrului, lățimea de bandă și ripluri maxime (acolo unde e cazul), pentru toate formele de răspuns (trece jos, trece sus, trece bandă, oprește bandă). Urmăriți în special și stabilitatea filtrului pentru setul de parametri ales.

2. Separați semnalul generat la lucrarea 1 în cele două componente alcătuitoare, prin proiectarea și folosirea unui filtru IIR trece jos care păstrează componenta de 50 Hz și a unui filtru IIR trece sus care păstrează componenta de 220 Hz.

3. Separați semnalul de 300 Hz din semnalul oferit în exercițiul 3 de la lucrarea 5 folosind un filtru IIR. Vizualizați semnalul înainte și după implementarea operație de filtrare și comparați rezultatele cu cele ale folosirii unui filtru FIR.

Lucrarea 7

Schimbarea ratei de eșantionare

7.1. Importanța frecvenței de eșantionare

Conform teoremei eșantionării, un semnal poate fi refăcut corect și complet din eșantioanele sale dacă frecvența de eșantionare satisface criteriul Nyquist:

$$f_s \geq 2 \cdot f_m \tag{7.1}$$

Pentru a putea satisface acest criteriu, spectrul semnalului respectiv trebuie să fie mărginit de o frecvență maximă f_m ca în figura 7.1.

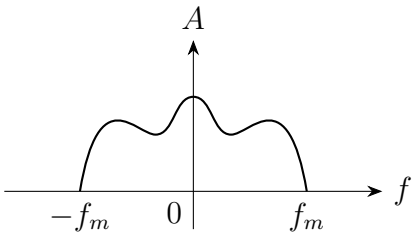


Figura 7.1. *Spectrul unui semnal mărginit*

Criteriul Nyquist este dat de faptul că, după eșantionare, spectrul semnalului va fi replicat centrat pe fiecare multiplu al frecvenței de eșantionare. Din figura 7.2 se poate

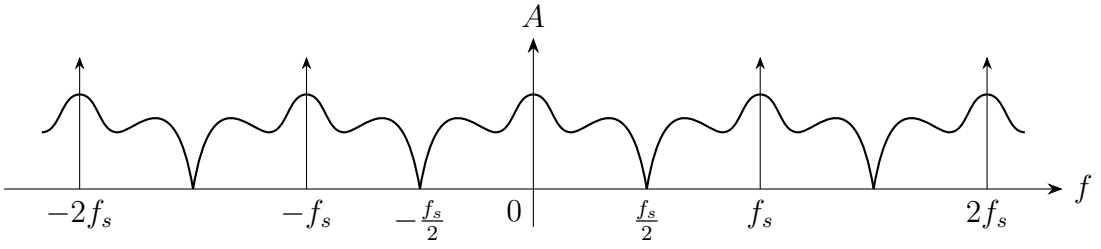


Figura 7.2. *Spectrul replicat al unui semnal eșantionat corect*

vedea clar că, pentru eșantionare corectă, adică pentru ca replicile spectrale să nu se suprapună, vom avea nevoie ca jumătatea frecvenței de eșantionare să fie cel puțin egală cu frecvența maximă din semnalul original.

Dacă criteriul Nyquist nu este respectat, atunci replicile spectrale se vor suprapune, fenomen ce poartă numele de aliere (vezi fig. 7.3). În acest caz, semnalul analogic original nu mai poate fi refăcut corect din eșantioanele sale.

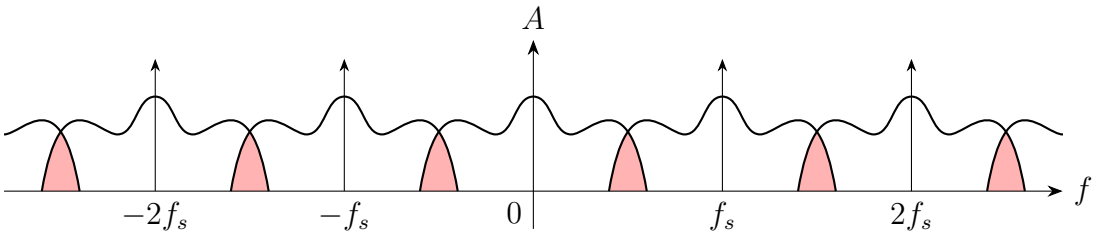


Figura 7.3. *Spectrul unui semnal subeșantionat, afectat de aliere*

Putem simula acest efect în Matlab, pornind de la un semnal eșantionat la o frecvență mai mare, pe care să aplicăm un efect eșantionare-memorare la o rată mai scăzută. De exemplu, pornind de la un semnal eșantionat la 8 kHz:

```
>> t = (0:1023)/8000;
>> s = 0.3*sin(2*pi*125*t) + 0.2*sin(2*pi*250*t+pi/4);
```

vom simula o eșantionare la frecvența de 1 kHz, prin selectarea a fiecărui al 8-lea eșantion și memorarea acestuia pe parcursul următoarelor 7 eșantioane:

```
>> x = s(floor((0:1023)/8)*8+1);
```

afișând aceasta cu:

```
>> w = 0:8000/1024:511*8000/1024;
>> stem(w,abs(fft(x))(1:512)/512);
```

obținem figura 7.4, pe care putem observa componentele originale de 125 și 250 Hz din semnal, cât și replicile acestuia centrate pe multiplii frecvenței de eșantionare.

Pentru refacerea semnalului original din eșantioane, practic va trebui să eliminăm replicile spectrale. Acest lucru putem realiza prin aplicarea unui filtru trece-jos ideal cu frecvența de tăiere la jumătatea frecvenței de eșantionare, ca în figura 7.5.

Așa cum s-a văzut în lucrările anterioare, filtrarea ideală nu este realizabilă în practică, filtrele reale vor avea un răspuns cu o tranziție mai lină între banda de trecere și banda de tăiere. Și trebuie să notăm și faptul că pentru refacerea semnalului analogic filtrarea trece jos va trebui realizată după ieșirea digitală cu filtre analogice. Acest filtru

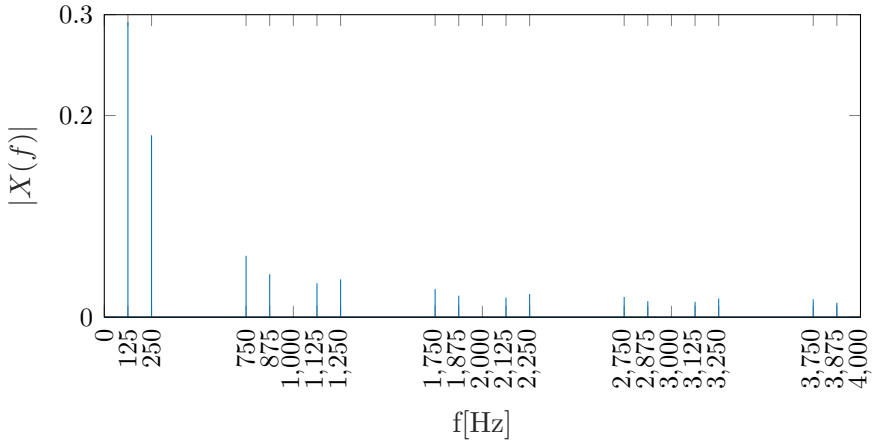


Figura 7.4. *Replicile spectrale a unui semnal eșantionat la 1kHz*

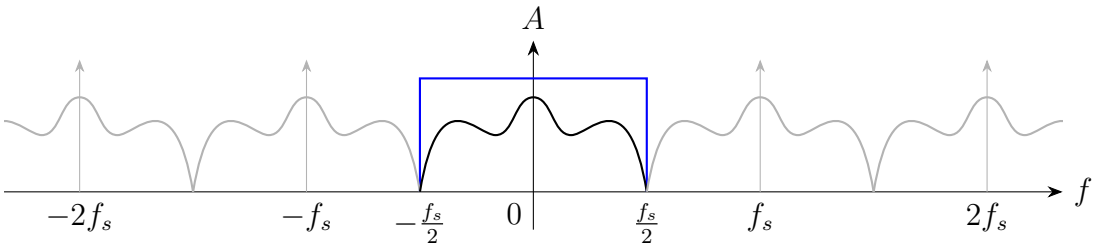


Figura 7.5. *Refacerea semnalului din eșantioane cu filtru trece jos ideal*

analogic probabil va avea și un ordin foarte mic, datorită costurilor, ceea ce înseamnă un interval de tranziție foarte mare în răspunsul filtrului.

Aplicând un astfel de filtru analogic pentru refacerea unui semnal eșantionat exact la frecvența Nyquist va rezulta în păstrarea a multor componente armonice la ieșire (vezi fig. 7.6).

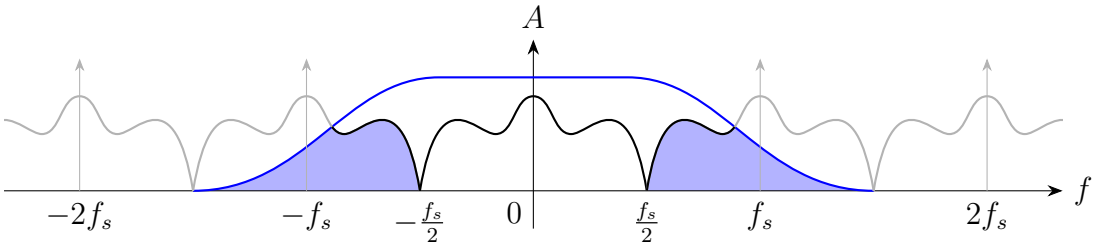


Figura 7.6. *Refacerea semnalului din eșantioane cu filtru trece jos real*

Pentru a reduce distorsiunea armonică în semnalul analogic refăcut din eșantionare, considerând doar posibilitatea folosirii unor filtre de ordin redus, trebuie să avem o frecvență de eșantionare mult mai mare decât frecvența Nyquist (fig. 7.7).

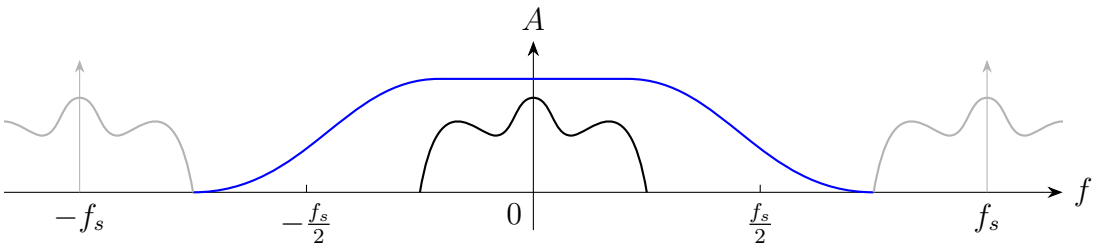


Figura 7.7. *Reducerea distorsiunilor armonice prin folosirea unei frecvențe de eșantionare mare*

Un alt aspect important al alegerii frecvenței de eșantionare este efectul indirect asupra zgomotului de cuantizare din semnalul discretizat. Așa cum s-a văzut în lucrarea 1, nivelul zgomotului de cuantizare depinde exclusiv de numărul de biți folosiți pentru reprezentarea eșantioanelor. Puterea zgomotului de cuantizare, datorat circuitului de conversie analog-digitală, dar și de efectul lungimii de cuvânt finit la fiecare operație matematică asupra semnalului, va fi identică indiferent de frecvența de eșantionare. Pe de altă parte, zgomotul acesta este asimilat unui zgomot distribuit uniform, ceea ce înseamnă că nu are un spectru mărginit, și implicit va fi supus fenomenului de aliere la eșantionare. După cum se vede și pe figura 7.8, dacă alegem o frecvență de eșantionare mai mare, o parte mai mică din zgomotul de cuantizare va fi direct suprapus peste semnalul util. Astfel, dacă la finalul lanțului de prelucrare aplicăm un filtru trece jos, putem să înlăturăm o parte din zgomotul introdus de lanțul de prelucrare în sine și să îmbunătățim raportul semnal-zgomot în semnalul rezultat.

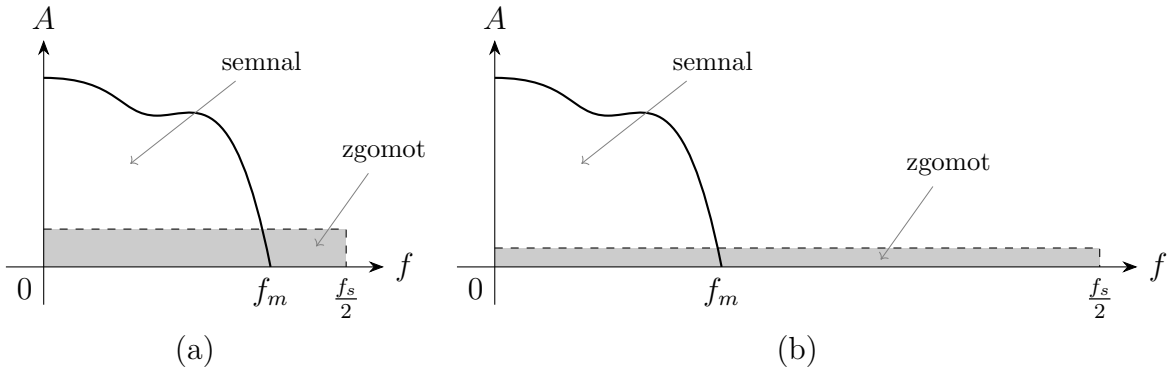


Figura 7.8. *Distribuția zgomotului de cuantizare la frecvența de eșantionare mică (a) respectiv mare (b)*

Bineînțeles, creșterea prea mare a frecvenței de eșantionare are și dezavantajele proprii. Circuitele de eșantionare-cuantizare rapide ce permit o frecvență de eșantionare mai mare sunt mai scumpe. O frecvență de eșantionare mai mare rezultă și într-un număr mai mare de eșantioane, ceea ce înseamnă costuri suplimentare pentru stocare și pentru transmisia la distanță a datelor.

De aceea se practică schimbarea ratei de eșantionare în interiorul blocului de procesare digitală. Datele se stochează la o frecvență de eșantionare mai mică, înainte de a începe prelucrarea se schimbă frecvența de eșantionare la una mai mare, iar după terminarea prelucrărilor se schimbă frecvența de eșantionare înapoi la o valoare mai mică. Aceasta include și avantajul că filtrele trece jos digitale pot fi implementate în lanțul de prelucrare digitală la un ordin superior cu un cost relativ redus.

7.2. Creșterea ratei de eșantionare

Creșterea ratei de eșantionare, numită interpolare, este realizată prin introducerea unor eșantioane suplimentare între eșantioanele curente ale semnalului. În Figura 7.9 puteți observa semnalul original, reprezentat de pătrățele albastre, și eșantioanele ce trebuie adăugate reprezentate cu ×.

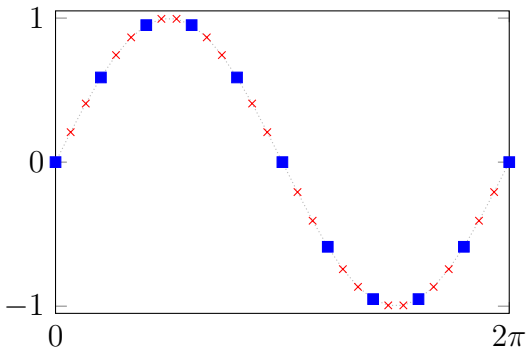


Figura 7.9. *Exemplu de interpolare*

Pentru a calcula valorile eșantioanelor ce trebuie adăugate putem folosi o metodă de interpolare, de exemplu interpolare liniară sau interpolare de tip spline. Interpolarea liniară este poate cea mai simplă metodă, care însă nu prezintă continuitate în jurul eșantioanelor vechi, introducând astfel o distorsiune semnificativă în semnal. Pentru a putea reproduce curbura originală, avem nevoie de o interpolare care ține cont de mai multe eșantioane (nu numai vecinii imediați, cum se întâmplă în cazul interpolării liniare), de exemplu interpolarea cubică folosește polinoame de gradul 3 pentru calcularea noilor valori dintre două eșantioane existente, polinoame ce sunt determinate din cel puțin 4 eșantioane vecine.

Câteva interpolări simple, disponibile în Matlab prin intermediul funcției `interp1` sunt prezentate pe figura 7.10, generată cu secvența de cod:

```
>> x = 0:0.2:1;
>> y = sin(2 * pi * x);
>> xi = 0:0.002:1;
>> ynear = interp1(x, y, xi, 'nearest');
```

```
>> ylin = interp1(x, y, xi, 'linear');
>> ypchip = interp1(x, y, xi, 'pchip');
>> yspline = interp1(x, y, xi, 'spline');
>> yi = sin(2 * pi * xi);
>> plot(xi, yi, ':k', xi, ynear, xi, ylin, xi, ypchip, xi, yspline,
   ↪  x, y, '*b');
```

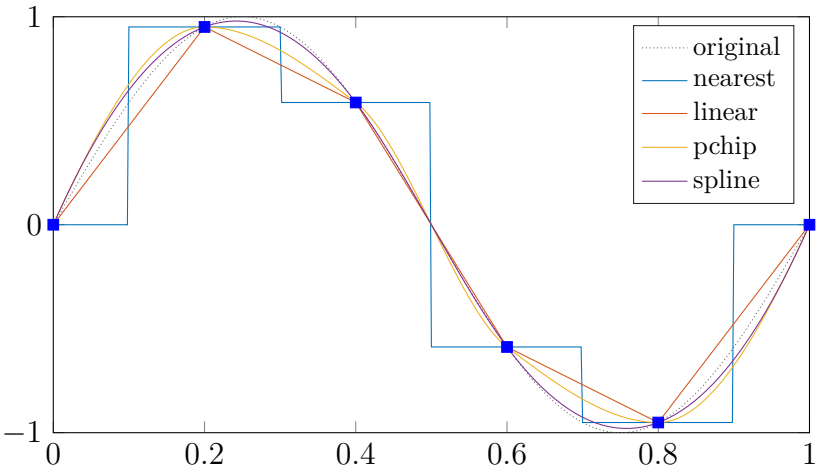


Figura 7.10. *Exemple de interpolare*

După cum se poate observa, polinoamele de interpolare cu gradul mai mare aproximează mai bine semnalul original. Interpolarea Lagrange, care folosește polinoame de gradul n (egal cu numărul total de eșantioane), poate oferi o aproximare și mai bună, dar costurile computaționale vor fi foarte mari.

O altă abordare, mai avantajoasă din punct de vedere computațional, este realizarea interpolării prin doi pași. Mai întâi creștem numărul de eșantioane prin introducerea unor eșantioane noi de valoare 0 între eșantioanele vechi. Această operație va avea ca efect apariția în spectru a unor imagini ale semnalului centrat pe multiplii frecvenței de eșantionare originale. Imaginile pot fi eliminate cu un filtru anti-imagine, un filtru trece jos de ordine superioară având frecvența de tăiere egal cu inversul factorului de interpolare, adică $\frac{f_{s1}}{f_{s2}}$. Astfel operația de interpolare poate fi reprezentată pe blocuri ca în figura 7.11.

Efectul acestei abordări asupra unui semnal eșantionat poate fi observat pe Fig. 7.12, care prezintă semnalul original, semnalul interpolat, în care între fiecare eșantion original s-a inserat câte un eșantion nou de valoarea 0, respectiv rezultatul filtrării trece jos. Pe graficul cu spectru se poate observa apariția imaginii de frecvență

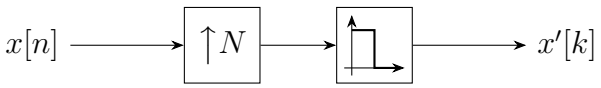


Figura 7.11. *Interpolare cu factor întreg N*

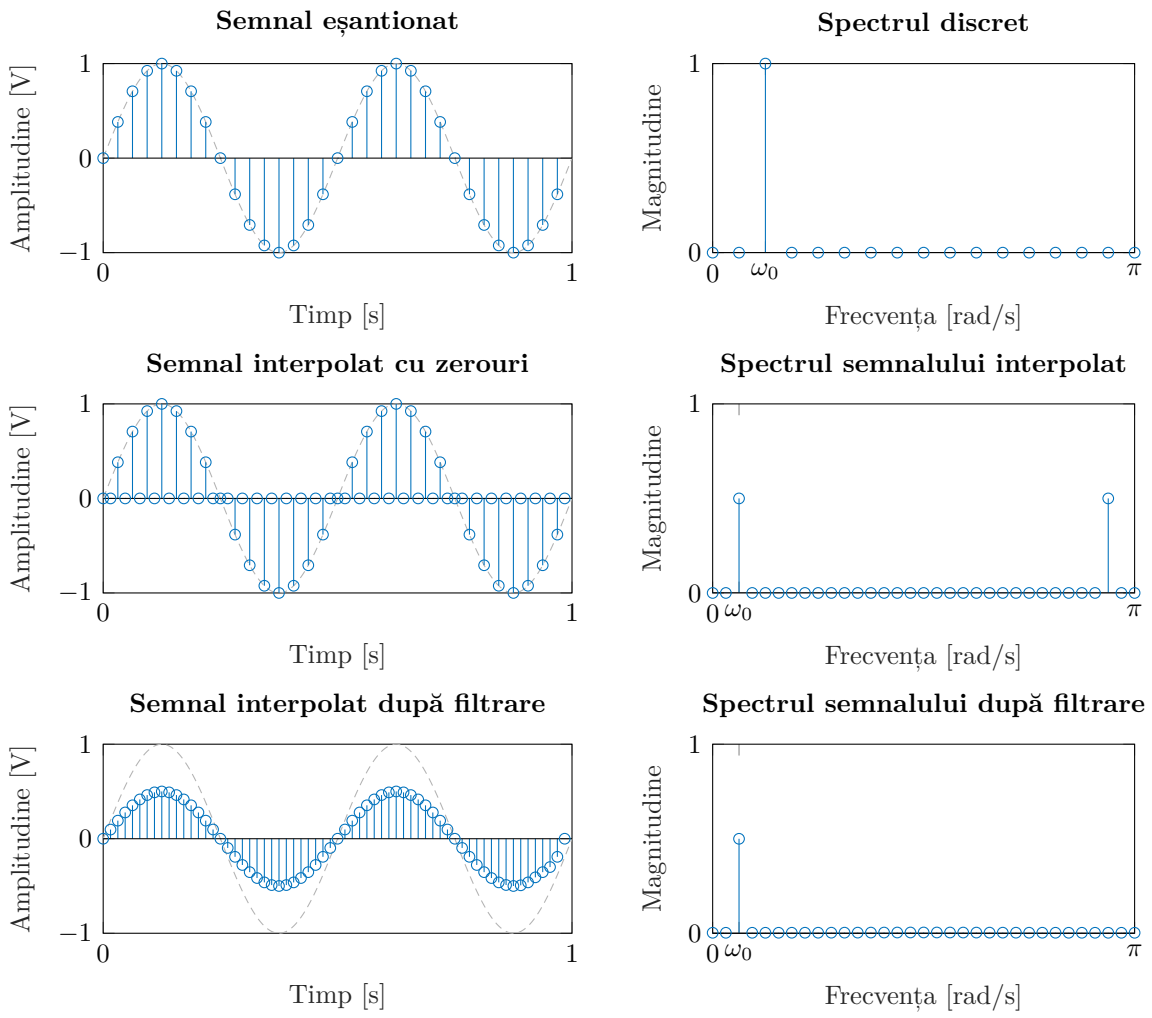


Figura 7.12. *Interpolarea unui semnal prin introducere de zerouri și filtrare trece jos.*

înaltă, care este eliminată cu filtrul trece jos, rezultând în semnal identic cu semnalul original, dar cu numărul de eșantioane dublate. În același timp se poate observa că amplitudinea semnalului rezultat scade datorită scăderii puterii totale a semnalului la inserarea de eșantioane de valoare 0, ceea ce poate fi contracarat cu o amplificare cu un factor egal cu factorul de interpolare.

Această operație de creștere a ratei de eșantionare prin introducere de zero-uri și filtrare trece jos este implementată în Matlab de funcția `resample`. Folosind semnalele de pe figura 7.10 putem compara și rezultatul acestei funcții cu cel al interpolării polinomiale, cu următoarea secvență de cod:

```
>> yr = resample(y, 100, 1);
>> plot(xi, yi, ':k', xi, yspline, xi, yr(1:length(xi), x, y, '*b');
```

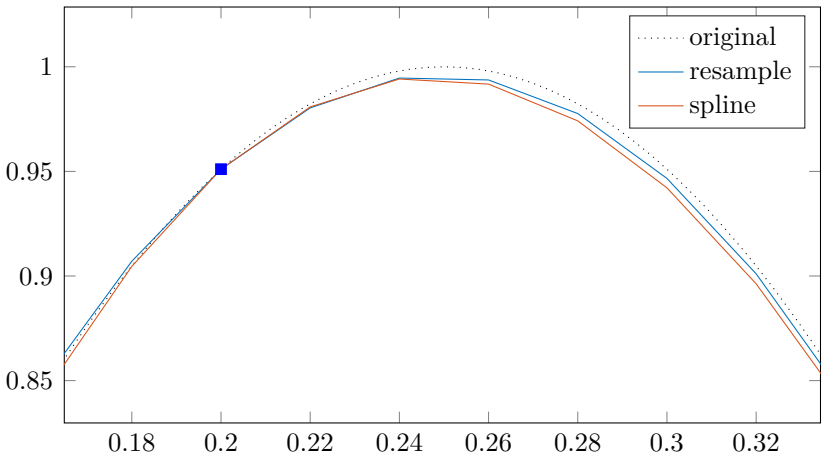


Figura 7.13. *Comparație între resample și interpolare polinomială*

Datorită modului de adăugare de zerouri, pentru o comparare corectă, lungimea secvenței interpolate trebuie ajustată să corespundă cu cea realizată prin interpolare polinomială. De asemenea, semnalul original poate fi extins pentru a acomoda calarea filtrului trece jos. Rezultatul operației este prezentat pe figura 7.13 folosind o mărire a imaginii într-o zonă pentru a vedea mai bine diferențele față de semnalul original.

În condițiile în care factorul de interpolare (raportul dintre frecvențele de eșantionare) este unul relativ mic, atunci această abordare poate fi mai avantajoasă, pentru că filtrarea trece jos ajută și la îmbunătățirea raportului semnal-zgomot, așa cum s-a văzut mai înainte, și deseori face parte din lanțul de prelucrare.

7.3. Scăderea ratei de eșantionare

Scăderea ratei de eșantionare, numită decimare, este realizată prin renunțarea la anumite eșantioane. Este o metodă complementară cu interpolarea, de exemplu, pentru cazul din Figura 7.9, eliminăm eșantioanele marcate cu $\times$, rămânând doar eșantioanele marcate cu pătrățele.

Trebuie avut însă grijă la apariția fenomenului de aliere: dacă există componente cu frecvențe mai mari decât jumătatea noii frecvențe de eșantionare, acestea se vor suprapune componentelor semnalului util. Pentru a evita acest lucru înainte de decimare, semnalul original trebuie trecut printr-un filtru anti-aliere, un filtru trece-jos cu frecvența la jumătatea frecvenței de eșantionare noi (vezi figura 7.14).

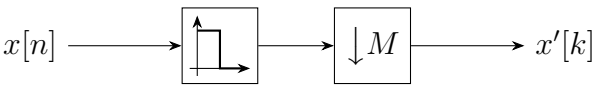


Figura 7.14. *Decimare cu un factor întreg M*

Dacă nu aplicăm filtrul de antialiere, atunci eventualele distorsiuni armonice datorită operațiilor de prelucrare pot fi suprapuse peste semnalul original, distorsionând

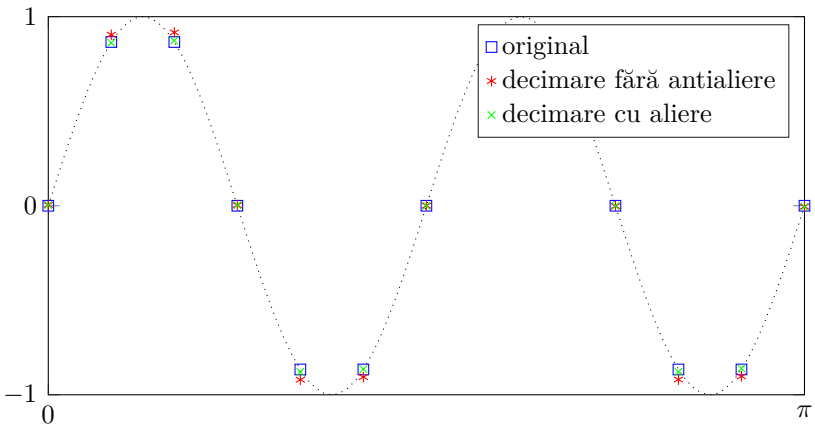


Figura 7.15. *Efecte de aliere asupra semnalului după interpolare și decimare*

acesta (prin amplitudine sau fază), așa cum este exemplificat pe figura 7.15. Aici eșantioanele semnalului original (marcate cu □) au fost interpolate, apoi pe semnalul interpolat s-au adăugat niște armonici superioare și s-a decimat înapoi la rata de eșantionare original într-o variantă fără filtru antialiere (marcată cu ∗) și într-o variantă cu filtru antialiere (marcată cu ×). Se poate observa cum rezultatul decimării folosind filtrul antialiere se apropie mai bine de semnalul original.

7.4. Schimbarea ratei de eșantionare cu un factor rațional

Operațiile de interpolare și decimare prin introducerea/eliminarea eșantioanelor, prezentate anterior, permit schimbarea ratei de eșantionare doar cu un factor întreg.

În multe situații însă este de dorit schimbarea ratei de eșantionare cu un factor rațional. Un exemplu clasic este conversia între o rată de eșantionare a semnalelor audio de la 44 100 Hz (audio CD) la 48 kHz (digital audio).

Pentru realizarea unei asemenea schimbări, semnalul original trebuie interpolat la o frecvență care reprezintă cel mai mic multiplu comun al celor două frecvențe de eșantionare, după care trebuie decimat la noua frecvență de eșantionare. Cele două filtre trece-jos (filtrul anti-imagine de la interpolare și filtrul anti-aliere de la decimare) se pot combina într-unul singur (vezi Figura 7.16).

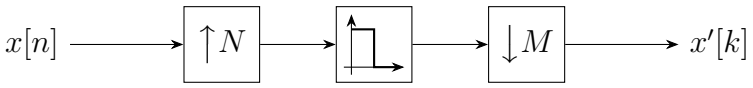


Figura 7.16. *Schimbarea ratei de eșantionare cu un factor rațional*

7.5. Exerciții

1. Realizați în Matlab interpolarea semnalului de la lucrarea 1 la o frecvență de eșantionare de 4 ori mai mare prin introducerea de 0-uri și filtrarea trece jos. Observați semnalul în fiecare etapă.

2. Observați efectul de aliere prin afișarea pe același grafic a unui semnal eșantionat corespunzător și a aceluiași semnal decimat la o frecvență mai mică decât cea corespunzătoare criteriului Nyquist.

3. Convertiți un semnal eșantionat la 44100 Hz la o rată de eșantionare de 48000 Hz. Găsiți calea optimă pentru a realiza conversia. Pentru a găsi calea optimă, porniți de la ideea că interpolarea cu un factor foarte mare va necesita foarte multe calcule.

   Pentru a calcula cel mai mic multiplu comun a două numere, folosiți funcția `lcm`, iar pentru factorizarea unui număr funcția `factor`.

Lucrarea 8

Detecția semnalelor DTMF

Semnalele DTMF (Dual-Tone Multiple Frequency) sunt foarte des folosite în sistemele de telefonie, control interactiv prin telefon și aplicații digitale prin linii telefonice. Acestea au înlocuit formarea numărului de apel cu codificare prin pulsuri (care suferea de pierderi de linie la distanțe mari), tonurile sinusoidale pure nefiind afectate de astfel de pierderi. Și chiar dacă în telefonia digitală modernă nu se mai folosește pentru apelarea numerelor, semnalizarea DTMF continuă să fie folosită pentru diferite aplicații de roboți telefonici.

8.1. Codarea semnalelor DTMF

Codificarea DTMF se bazează pe construirea a două tonuri sinusoidale cu frecvențe luate din două grupuri mutual exclusive de frecvențe prestabilite. Fiecare pereche de tonuri reprezintă o cifră sau un simbol unic. Combinația de frecvențe pentru fiecare simbol este prezentată în tabela 8.1, Fiecare simbol va fi codat cu o sinusoidă de frecvența din linia corespunzătoare însumat cu o sinusoidă de frecvența din coloana corespunzătoare simbolului.

Tabela 8.1. *Combinația de frecvențe DTMF*

	1209 Hz	1336 Hz	1477 Hz	1633Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Frecvențele au fost alese astfel încât să nu existe raport armonic între ele, adică nu sunt multipli unui celuilalt, nu sunt sume sau diferențe între alte două frecvențe și nu au un divizor comun în banda vocală. Astfel se va evita identificarea falsă datorită interferențelor și distorsiunilor de linie.

Pentru a genera un semnal DTMF se poate folosi următoarea funcție

dtmf_encode_symbol.m

```
% DTMF_ENCODE_SYMBOL - codifică un simbol cu două tonuri de
%                      de frecvență distincte
% symbol - este simbolul codificat, trebuie să fie un caracter
%          din gama '0'-'9', 'A'-'D', '*', '#'
% len - numărul de eșantioane generate
% fs - frecvența de eșantionare
function [X] = dtmf_encode_symbol(symbol, len, fs)
  symbols = ['1', '2', '3', 'A';
             '4', '5', '6', 'B';
             '7', '8', '9', 'C';
             '*', '0', '#', 'D'];
  freq1 = [697, 770, 852, 941];
  freq2 = [1209, 1336, 1477, 1633];
  % căutăm mai întâi simbolul din tabela de simboluri, pentru
  % a identifica frecvențele corespunzătoare
  [r,c] = find(symbols == symbol, 1);
  % dacă simbolul nu este găsit se emite un mesaj de eroare
  if isempty(r)
    error("Unknown symbol");
  end
  % altfel se generează semnalul cu suma a două sinusoide
  t = (0:len-1)/fs;
  X = sin(2 * pi * freq1(r) * t) + sin(2 * pi * freq2(c) * t);
end
```

Fiecare simbol va avea câte două componente spectrale ce pot fi foarte ușor vizualizate cu transformata Fourier rapidă (fig. 8.1) folosind de exemplu o secvență:

```
>> x = dtmf_encode_symbol('0', 320, 8000);
>> X = abs(fft(x)(1:160));
>> w = 0:8000/320:159*8000/320;
>> stem(w,X);
>> axis([500 2000 0 160]);
```

Este important de observat că numărul de eșantioane ales pentru semnalul DTMF afectează profund spectrul rezultat după transformata Fourier rapidă datorită scurgerii spectrale a acestuia. Astfel, în funcție de lungimea aleasă a transformatei, putem avea unele tonuri foarte precis așezate pe un singur eșantion spectral, altele însă împărțite între două eșantioane vecine, care poate să îngreuneze detecția lor. Exemplul de spectru prezentat pe figura 8.1 folosește niște lungimi de ton alese astfel încât să ofere cea mai bună reprezentare pentru fiecare număr în parte.

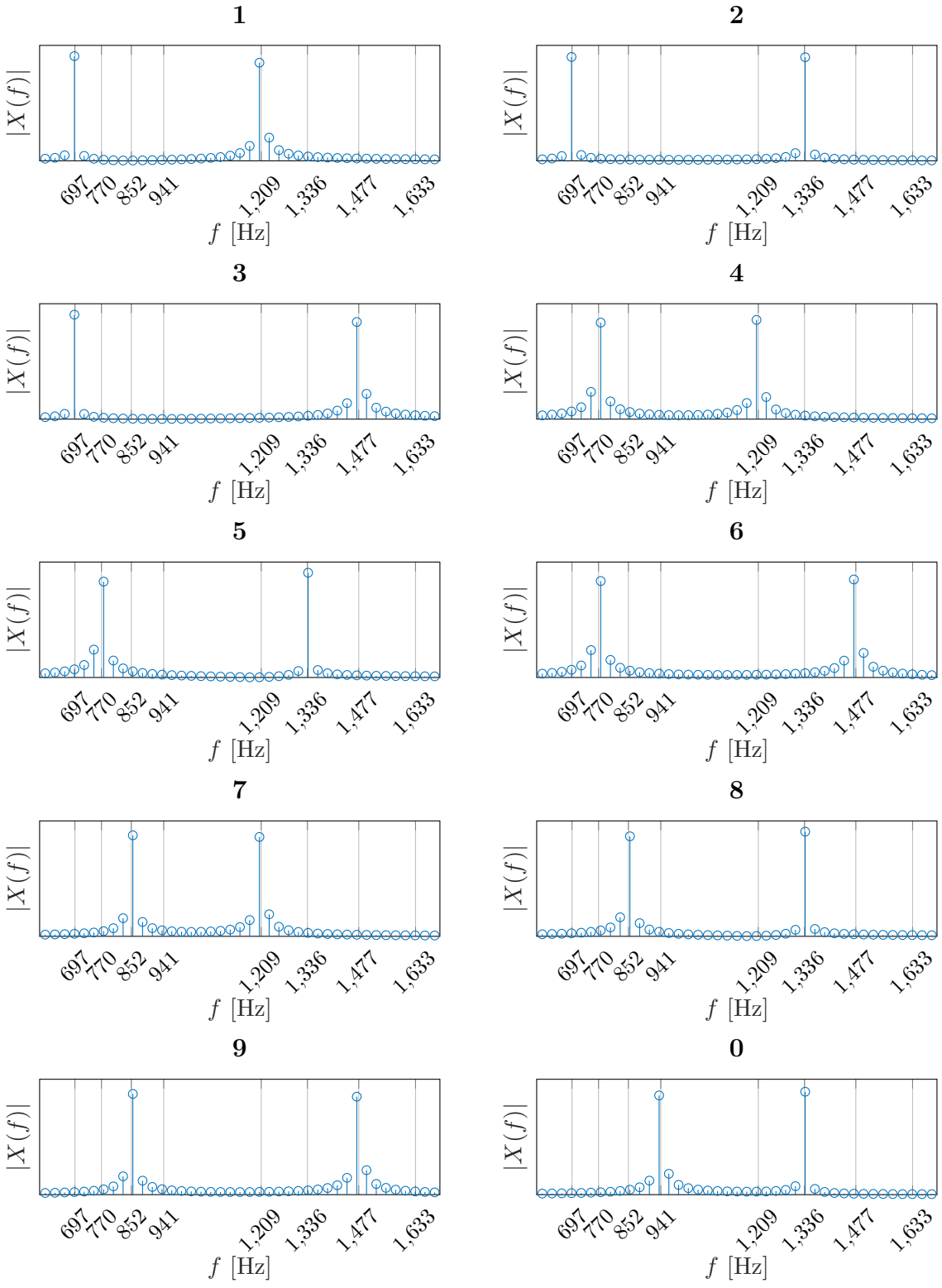


Figura 8.1. *Spectrul semnalelor DTMF corespunzătoare diferitelor cifre*

8.2. Decodarea semnalelor DTMF

Pentru decodarea semnalelor DTMF, sistemele analogice foloseau de un banc de filtre trece bandă acordate pentru cele 8 frecvențe componente posibile. Evident, această abordare poate fi folosită și în digital. Trebuie însă avut în vedere că vom avea nevoie de filtre trece bandă performante cu benzi de trecere foarte înguste, ceea ce înseamnă filtre de ordin superior și frecvențe de tăiere selectate cu grijă pentru selectivitate optimă între tonurile de frecvență adiacente.

O abordare mult mai facilă este folosirea analizei spectrale prin intermediul transformatei Fourier rapide.

Semnalul recepționat introducem într-o fereastră suficient de largă încât să avem o rezoluție ce permite identificarea separată a tuturor frecvențelor folosite în semnalele DTMF, după care verificăm punctele corespunzătoare acelor frecvențe pentru a determina prezența componentei respective în semnalul de la intrare. După ce găsim cele două componente posibile, putem determina simbolul corespunzător dintr-un lookup-table.

O implementare banală pentru aceasta ar fi:

dtmf_decode_symbol.m

```
% DTMF_DECODE_SYMBOL - decodifică un simbol cu două tonuri de
%                      frecvențe
% x - semnalul conținând simbolul DTMF
% fs - frecvența de eșantionare folosit la semnal
% funcția returnează un caracter conținând simbolul detectat
% un caracter gol dacă nu detectează nici un simbol respectiv
% un caracter '!' dacă detectează prea multe tonuri
function [s] = dtmf_decode_symbol(x, fs)
  symbols = ['1', '2', '3', 'A';
             '4', '5', '6', 'B';
             '7', '8', '9', 'C';
             '*', '0', '#', 'D'];
  freq1 = [697, 770, 852, 941];
  freq2 = [1209, 1336, 1477, 1633];
  X = abs(fft(x));
  f1 = X(round(freq1 * length(x) / fs) + 1);
  f2 = X(round(freq2 * length(x) / fs) + 1);
  r = find(f1 > length(x) / 4);
  c = find(f2 > length(x) / 4);
  if length(r) == 1 && length(c) == 1
    s = symbols(r,c);
  else if length(r) > 1 || length(c) > 1
    s = '!';
  else
    s = '';
  end
end
```

Din păcate, această abordare are de suferit în prezența zgomotului și alegerea ferestrei corespunzătoare ar putea fi problematică. Pentru adaptarea mai bună a analizei spectrale, dar și pentru a îmbunătăți performanțele se poate opta pentru folosirea filtrului Goertzel.

Algoritmul Goertzel

Filtrul Goertzel este o metodă mai avantajoasă pentru a calcula transformata Fourier corespunzătoare a unui număr redus de eșantioane spectrale. Aceasta realizează calculul transformatei printr-un algoritm recursiv, asemănător unui filtru digital, cu o complexitate a algoritmului comparabil sau chiar puțin mai mare decât transformata Fourier discretă tradițională, deci mult mai lent decât transformata rapidă, dar făcând calculele pentru un număr mic de eșantioane din totalul spectrului, puterea computațională totală necesară este mai mică.

Transformata Fourier discretă corespunzătoare unui eșantion spectral k este definită ca:

$$X[k] = \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{kn}{N}} \tag{8.1}$$

Având în vedere că:

$$e^{j2\pi \frac{kN}{N}} = 1 \tag{8.2}$$

putem extinde ecuația 8.1 ca:

$$X[k] = \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{kn}{N}} e^{j2\pi \frac{kN}{N}} = \sum_{n=0}^{N-1} x(n) e^{j2\pi \frac{k}{N}(N-n)} \tag{8.3}$$

Această ecuație este similară cu o convoluție scrisă sub forma:

$$y_k[l] = \sum_{n=0}^{l} x_e[n] e^{j2\pi \frac{k}{N}(l-n)} \tag{8.4}$$

Aplicând transformata Z asupra acestei convoluții, obținem:

$$Y_k(z) = \frac{X_e(z)}{1 - e^{j2\pi \frac{k}{N}} z^{-1}} \tag{8.5}$$

Considerând ieșirea unui sistem LIT generic:

$$Y(z) = H(z) \cdot X(z) \tag{8.6}$$

putem considera ecuația 8.5, ca fiind rezultatul în frecvență a aplicării unui sistem LIT cu răspunsul în frecvență:

$$H_k(z) = \frac{1}{1 - e^{j2\pi \frac{k}{N}} z^{-1}} \tag{8.7}$$

Aceasta corespunde unui filtru IIR prezentat pe figura 8.2, având următorii coeficienți:

$$b_1 = 1 \quad a_2 = e^{j2\pi\frac{k}{N}} \tag{8.8}$$

Dacă aplicăm acest filtru pe un semnal $x_e[n]$ egal cu $x[n]$ pentru $N = \overline{0, N-1}$ respectiv 0 în rest, adică $x[N] = 0$, și considerăm starea inițială a filtrului $y[-1] = 0$, atunci ieșirea filtrului pentru al N-lea element $y_k[N]$ va fi exact $X[k]$, adică eșantionul k din transformata Fourier a semnalului de la intrare.

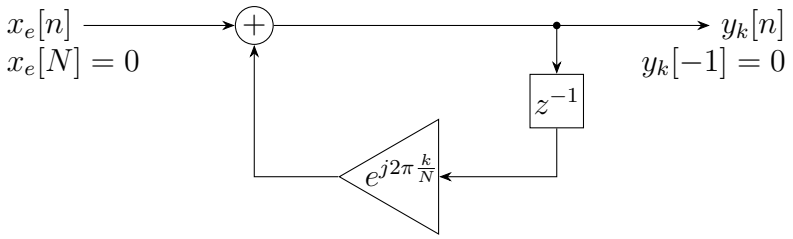


Figura 8.2. *Filtru Goertzel de ordinul 1*

Complexitatea acestui calcul (care implică și niște înmulțiri complexe) este mai mare decât complexitatea calculului transformatei Fourier directe, dar putem reduce complexitatea de calcul dacă creștem ordinul filtrului prin introducerea unui termen suplimentar $1 - e^{-j2\pi\frac{k}{N}}z^{-1}$, astfel:

$$\begin{aligned} H_k(z) &= \frac{1}{1 - e^{j2\pi\frac{k}{N}}z^{-1}} \cdot \frac{1 - e^{-j2\pi\frac{k}{N}}z^{-1}}{1 - e^{-j2\pi\frac{k}{N}}z^{-1}} = \frac{1 - e^{-j2\pi\frac{k}{N}}z^{-1}}{(1 - e^{j2\pi\frac{k}{N}}z^{-1})(1 - e^{-j2\pi\frac{k}{N}}z^{-1})} \\ &= \frac{1 - e^{-j2\pi\frac{k}{N}}z^{-1}}{1 - (e^{-j2\pi\frac{k}{N}} + e^{j2\pi\frac{k}{N}})z^{-1} + (e^{-j2\pi\frac{k}{N}+j2\pi\frac{k}{N}})z^{-2}} \\ &= \frac{1 - e^{-j2\pi\frac{k}{N}}z^{-1}}{1 - 2\cos(2\pi\frac{k}{N})z^{-1} + z^{-2}} \end{aligned} \tag{8.9}$$

Prin această operație coeficienții recursivi vor fi reali și doar un coeficient direct va fi număr complex:

$$\begin{aligned} b_1 &= 1 & b_2 &= e^{-j2\pi\frac{k}{N}} \\ a_2 &= 2\cos\left(2\pi\tfrac{k}{N}\right) & a_3 &= -1 \end{aligned} \tag{8.10}$$

Acest lucru presupune în continuare o complexitate de calcul similară ca în forma precedentă, dar dacă studiem structura filtrului rezultat, așa cum se vede pe figura 8.3, putem observa că e suficient să facem doar calculele reale, nu e nevoie să facem și toate calculele complexe. Având în vedere că ne interesează doar elementul $y_k[N]$, este suficient să parcurgem variabila internă $u[n]$ prin reacția filtrului pentru fiecare element de la intrare, iar numai în ultima iterație să adăugăm și rezultatul înmulțirii cu coeficientul complex $e^{-j2\pi\frac{k}{N}}$.

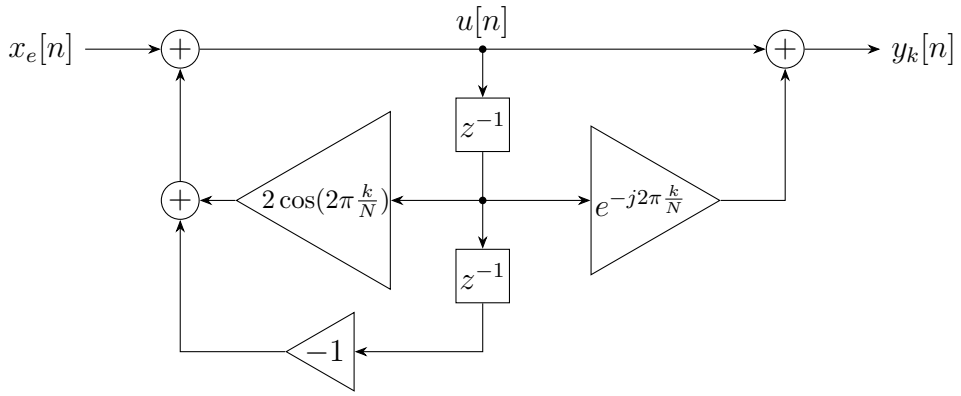


Figura 8.3. *Filtru Goertzel de ordinul 2*

Prin această optimizare putem avea o buclă doar cu adunări/înmulțiri de numere reale și doar după terminarea buclei o singură înmulțire de numere complexe, ceea ce se va executa mai rapid decât o buclă din transformata Fourier discretă.

Astfel, o implementare simplă pentru a calcula al k-lea element al transformatei Fourier discrete (X_k) a unui semnal stocat într-un vector x este prezentată de următorul pseudocod:

$N \leftarrow len_x$ // lungimea DFT
$s_1 \leftarrow 0$ // stările interne a filtrului
$s_2 \leftarrow 0$
$x_e \leftarrow [x\ 0]$ // pasul adiţional pentru a obţine $X[k]$
for $n \leftarrow 0$ **to** N **do**
    $u \leftarrow x_e[n] + 2\cos(2\pi\frac{k}{N}) \cdot s_1 - s_2$
    $s_2 \leftarrow s_1$
    $s_1 \leftarrow u$
$X_k \leftarrow s_1 - e^{-j2\pi\frac{k}{n}} \cdot s_2$

Având în vedere că în majoritatea cazurilor algoritmul Goertzel este folosit pentru a determina puterea unei anume purtătoare din semnalul de intrare, putem să optimizăm și mai mult algoritmul. În ultimul pas al buclei putem calcula pătratul magnitudinii spectrale, ceea ce va rezulta în operații aritmetice pe numere reale. Știind că pentru un semnal real la intrare, toate variabilele intermediare u vor fi și ele reale, putem aplica:

$$\begin{aligned} y_k[N] &= u[N] - e^{-j2\pi\frac{k}{N}}u[N-1] \\ &= x[N] + 2\cos(2\pi\tfrac{k}{N})u[N-1] - u[N-2] - e^{-j2\pi\frac{k}{N}}u[N-1] \\ &= 0 + \cos(2\pi\tfrac{k}{N})u[N-1] - u[N-2] + j\sin(2\pi\tfrac{k}{N})u[N-1] \end{aligned} \tag{8.11}$$

$$\begin{aligned} |y_k[N]|^2 &= \big(\cos(2\pi\tfrac{k}{N})u[N-1] - u[N-2]\big)^2 + \big(\sin(2\pi\tfrac{k}{N})u[N-1]\big)^2 \\ &= u^2[N-1] + u^2[N-2] - 2\cos(2\pi\tfrac{k}{N})u[N-1]u[N-2] \end{aligned} \tag{8.12}$$

Algoritmul modificat va arăta astfel:

$N \leftarrow len_x$
$s_1 \leftarrow 0$
$s_2 \leftarrow 0$
for $n \leftarrow 0$ **to** $N - 1$ **do**
    $u \leftarrow x[n] + 2\cos(2\pi\frac{k}{N}) \cdot s_1 - s_2$
    $s_2 \leftarrow s_1$
    $s_1 \leftarrow u$
$|X_k|^2 \leftarrow s_1^2 + s_2^2 - 2\cos(2\pi\frac{k}{N}) \cdot s_1 \cdot s_2$

Având numai calcule pe numere reale, timpul total de calcul va fi mai mic decât pentru un DFT complex, iar dacă numărul de eșantioane pe care vrem să calculăm este comparabil mai mic decât $\log_2 N$ (ca în cazul tonurilor DTMF), atunci calculele pot fi făcute mai eficient față de FFT.

Folosirea a 8 filtre Goertzel corespunzătoare frecvențelor DTMF mai are și avantajul că fiecare dintre ele poate fi adaptat separat la o lungime de fereastră care să minimizeze erorile de detecție, și astfel să fie mai fiabil și în prezența zgomotului, decât folosirea unei singure transformate Fourier rapide.

8.3. Exerciții

1. Determinați dimensiunea optimă a ferestrei pentru transformata Fourier capabilă să decidă corect frecvențele optime știind faptul că durata minimă a unui cod DTMF conform standardului este de 40ms.

2. Studiați fezabilitatea folosirii metodei bazate pe FFT pentru detecția semnalelor DTMF înecate în zgomot.

3. Implementați algoritmul Goertzel și verificați fezabilitatea folosirii unui banc de filtre Goertzel adaptate separat pentru fiecare ton specific DTFM pentru detecția semnalelor DTMF înecate în zgomot.

Lucrarea 9

Prelucrarea semnalelor audio în Matlab/Octave

O parte aplicativă de interes general în procesarea semnalelor reprezintă prelucrarea semnalelor audio, ceea ce poate fi realizat foarte ușor și folosind MATLAB sau Octave prin bibliotecile încorporate pentru a interacționa cu placa de sunet din calculator.

9.1. Achiziția semnalelor audio

Pentru achiziția semnalelor audio în Matlab, trebuie să urmărim doi pași: trebuie selectat și configurat dispozitivul de captare a semnalului audio (în general microfonul încorporat), după care trebuie înregistrat o secvență audio folosind acel dispozitiv, care să rezulte în vectorul de eșantioane pe care putem prelucra ulterior.

Selectarea dispozitivului audio

Dispozitivele de înregistrare audio din sistem pot fi accesate în Matlab folosind un obiect din clasa `audiorecorder`. Obiectul se creează în felul următor:

```
>> r = audiorecorder();
```

ceea ce creează obiectul `r` folosind intrarea implicită și modul de eșantionare implicit setat în sistem. Aceasta din urmă poate varia de la sistem la sistem, putem avea una sau două canale de achiziție (mono sau stereo), un nivel de cuantizare de la 8 până la 24 de biți, și o frecvență de eșantionare de la 8 kHz și până la 192 kHz. În practică cel mai des se întâlnește combinația de 16 biți și 44.1 kHz. Setările concrete pot fi afișate prin parametri obiectului:

```
>> r
```

Dacă vrem să controlăm manual parametrii de eșantionare, se poate opta la specificarea acestora în parametrii funcției `audiorecorder` precizând în ordine frecvența de eșantionare, numărul de biți folosiți la cuantizare și numărul de canale. Pentru a înregistra un semnal vocal de telefonie clasic, eșantionat la 8 kHz, cu un singur canal cuantizat la 8 biți, vom folosi:

```
>> r = audiorecorder(8000, 8, 1);
```

De remarcat însă este faptul că în telefonie de obicei se folosesc semnale compandate cu legea μ sau legea A, care rezultă într-o cuantizare logaritmică. Plăcile audio din calculatoare folosesc o cuantizare liniară de aceea pentru a acoperi oarecum o bandă dinamică similară, adică o calitate vocală suficientă, ar trebui să achiziționăm semnalul la un nivel de cuantizare de 16 biți, după care putem companda eșantioanele rezultate la 8 biți cuantizați logaritmic.

```
>> r = audiorecorder(8000, 16, 1);
```

Dacă vrem să obținem o calitate muzicală suficient de bună, atunci trebuie să ne orientăm către frecvențe de eșantionare peste 40 kHz, care să acopere întreaga plajă de percepție auditivă umană (considerat a fi între 20 Hz și 20 kHz). De exemplu, un sunet de calitate corespunzătoare cu CD audio obținem cu

```
>> r = audiorecorder(44100, 16, 1);
```

Dacă sistemul permite mai multe intrări în sistem (de exemplu microfon încorporat, Line-In etc) atunci se poate preciza și dispozitivul folosit într-un al patrulea parametru:

```
>> r = audiorecorder(8000, 8, 1, id);
```

unde `id` este un număr ce identifică dispozitivele din sistem. Pentru a afla identificatorii prezenți în sistem se folosește obiectul `audiodevinfo`. Aceasta are doi membri, `input` și `output` (primul oferind informații despre dispozitive de înregistrare, iar al doilea despre dispozitive de redare), fiecare la rândul său având trei membri, `Name`, `DriverVersion` și `ID`. Verificând acești membri cu:

```
>> audiodevinfo.input.Name
>> audiodevinfo.input.ID
```

se va afișa toate dispozitivele disponibile, cu numele configurat în sistem, respectiv ID-ul asociat fiecăreia.

Pentru o afișare mai comodă a combinațiilor de identificator și nume putem aplica o funcție anonimă de afișare pe vectorul de dispozitive:

```
>> arrayfun(@(c) disp(sprintf('%d: %s', c.ID, c.Name)),
   ↪  audiodevinfo.input)
```

care poate rezulta de exemplu într-o listă de felul următor:

```
0:   HDA Intel PCH: ALC236 Analog (hw:0,0) (ALSA)
4:   sysdefault (ALSA)
10:  lavrate (ALSA)
11:  samplerate (ALSA)
12:  speexrate (ALSA)
13:  pulse (ALSA)
14:  speex (ALSA)
15:  upmix (ALSA)
16:  vdownmix (ALSA)
18:  default (ALSA)
```

Dintre acestea, dispozitivul de tipul ALC236 este unicul dispozitiv de captare audio, restul sunt pseudo-dispozitive asociate diferitelor pluginuri ALSA. Deci, pentru a înregistra audio de la microfonul nativ al sistemului, putem folosi ID-ul 0:

```
>> r = audiorecorder(8000, 16, 1, 0);
```

Înregistrarea semnalului audio

După ce a fost creat obiectul `audiorecorder`, acesta poate fi folosit pentru a înregistra un semnal audio prin intermediul funcțiilor `record` respectiv `recordblocking`. Primul permite înregistrare fără să blocheze accesul la linia de comandă (deci se poate continua execuția altor funcții în timp ce se decurge înregistrarea), iar al doilea se blochează până la terminarea înregistrării. Durata înregistrării se specifică în al doilea parametru al acestor funcții, primul parametru fiind obiectul de înregistrare folosit.

Astfel, pe obiectul `audiorecorder` creat anterior (cu 8 kHz frecvența de eșantionare și 16 biți de cuantizare) putem să înregistrăm vocea noastră prin microfon pe o durată de exemplu de 3 secunde fie cu comanda:

```
>> record(r, 3);
```

fie cu comanda:

```
>> recordblocking(r, 3);
```

Diferența este că prima variantă va da înapoi controlul la linia de comandă și putem să introducem altceva în timp ce vorbim, a doua variantă însă va aștepta parcurgerea celor 3 secunde și numai după aceea va da controlul înapoi.

În mod normal, varianta `recordblocking` este preferată, mai ales în cazul în care folosim funcția într-un script, pentru că funcțiile imediat următoare deja pot să accese datele înregistrate. Altfel, va trebui să ne asigurăm că nu apelăm operații care accesează eșantioanele înainte de expirarea timpului de înregistrare.

După terminarea înregistrării, inspectând obiectul:

```
>> r
```

vom vedea proprietățile acestuia:

```
BitsPerSample    = 16
CurrentSample    = 0
DeviceID         = -1
NumberOfChannels = 1
Running          = off
SampleRate       = 8000
TotalSamples     = 24000
Tag              =
Type             = audiorecorder
UserData         = [](0x0)
```

Putem observa că obiectul acum conține 24000 de eșantioane (*TotalSamples*), corespunzătoare celor 3 secunde de înregistrare cu 8000 de eșantioane pe secundă.

Pentru a accesa aceste eșantioane trebuie folosit funcția `getaudiodata`. Aceasta citește eșantioanele stocate în obiectul `r` și returnează un vector (sau o matrice, dacă înregistrarea s-a realizat pe mai multe canale) cu eșantioanele prelevate, folosind tipul de date precizat.

```
>> x = getaudiodata(r);
```

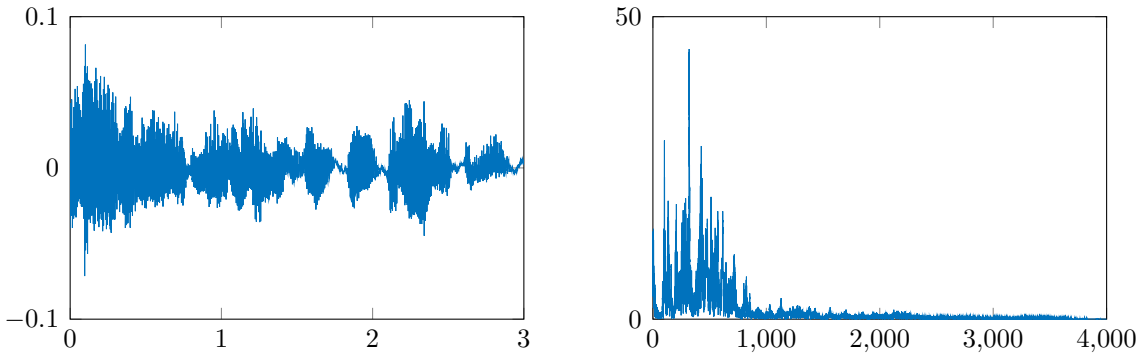
Implicit eșantioanele sunt reprezentate cu un tip `double`, având valorile între -1 și 1, dar se poate cere și eșantioanele reprezentate pe întregi de 8 (valori între -128 și 127) sau 16 biți (valori între -32768 și 32767).

Vectorul respectiv este un vector obișnuit folosit și la alte prelucrări de semnale (cu observația că eșantioanele sunt aranjate pe coloană).

Semnalul din acest vector poate fi afișat grafic în domeniul timp sau în domeniul spectral cu funcțiile deja obișnuite:

```
>> figure, plot((0:r.TotalSamples-1)/r.SampleRate, x);
>> figure, plot((0:r.TotalSamples/2-1)*r.SampleRate/r.TotalSamples,
   ↪  abs(fft(x))(0:length(x)/2));
```

rezultând în graficele de pe figura 9.1.



(a) Timp *(b) Spectru*

Figura 9.1. *Semnal vocal în timp și în spectru*

9.2. Generarea semnalelor audio

Pentru redarea semnalelor audio, de asemenea trebuie să urmărim doi pași: construirea unui obiect de interfațare cu dispozitivul audio și operația de redare a semnalului pe dispozitivul selectat.

Interfața de ieșire cu dispozitivul audio

Pentru ieșirea audio se va folosi în Matlab un obiect de tip `audioplayer`. Acest obiect trebuie construit asemănător cu `audiorecorder` prin precizarea frecvenței de eșantionare, nivelul de cuantizare și identificatorul dispozitivului audio din sistem. Principala diferență însă față de `audiorecorder` este că obiectul `audioplayer` trebuie umplut cu eșantioane încă de la instanțiere.

Astfel pentru a crea un `audioplayer` în cea mai simplă structură vom folosi comanda

```
>> a = audioplayer(x, fs);
```

unde `x` este un semnal pe care vrem să redăm și `fs` este frecvența de eșantionare dorită.

Precizarea nivelului de cuantizare este opțional, acesta poate fi dedus din tipul de date al vectorului de semnal, iar dacă vectorul conține valori de tip `double`, atunci se va folosi cuantizarea cea mai bună oferită de placa audio. Dacă vrem să reducem acest nivel, de exemplu pentru clasicul 8 biți la 8 kHz, putem folosi:

```
>> a = audioplayer(x, 8000, 8);
```

Evident putem alege și dispozitivul audio dorit, precizând identificatorul în al patrulea parametru. Așadar, dacă executând comanda:

```
>> arrayfun(@(c) disp(sprintf('%d: %s', c.ID, c.Name)),
   ↪  audiodevinfo.input)
```

obținem de exemplu o listă de felul următor:

```
0  = HDA Intel PCH: ALC236 Analog (hw:0,0) (ALSA)
1  = HDA Intel PCH: HDMI 0 (hw:0,3) (ALSA)
2  = HDA Intel PCH: HDMI 1 (hw:0,7) (ALSA)
3  = HDA Intel PCH: HDMI 2 (hw:0,8) (ALSA)
4  = sysdefault (ALSA)
5  = front (ALSA)
6  = surround40 (ALSA)
7  = surround51 (ALSA)
8  = surround71 (ALSA)
9  = hdmi (ALSA)
10 = lavrate (ALSA)
11 = samplerate (ALSA)
12 = speexrate (ALSA)
13 = pulse (ALSA)
14 = speex (ALSA)
15 = upmix (ALSA)
16 = vdownmix (ALSA)
17 = dmix (ALSA)
18 = default (ALSA)
```

atunci putem selecta placa audio analogică, de exemplu, pentru redarea unui semnal vocal prin comanda:

```
>> a = audioplayer(x, 8000, 8, 0);
```

iar pentru redarea unei melodii pe un televizor conectat la ieșirea a doua de HDMI:

```
>> a = audioplayer(x, 44100, 16, 2);
```

Redarea semnalelor audio

Pentru a reda semnalul audio din obiectul `audioplayer`, se folosește funcția `play` ce se execută în felul următor.

```
>> play(a)
```

Execuția funcției este asincronă, controlul revine imediat în linia de comandă în timp ce redarea sunetului continuă până la terminarea tuturor eșantioanelor. Comanda poate fi executată de mai multe ori pe același obiect `audioplayer`, redarea sunetului începând din nou de la primul eșantion.

Pentru generarea unui sunet putem folosi, de exemplu, o secvență de cod, care generează un ton modulat de tip A1A:

```
>> t = (0:16999)/8000;
>> s = 0.25*sin(2*pi*800*t);
>> s(([3,5,9,10,11,15,17,21,23,24,25,27,29,31,32,33]*500+(0:499)')(:)+1)=
   ↪  0;
>> a = audioplayer(s, 8000);
>> play(a);
```

Se poate opta și pentru redarea unui sunet înregistrat într-un obiect `audiorecorder`. Funcția `play` acceptă ca parametru și un astfel de obiect, caz în care se va crea automat un `audioplayer` având exact aceleași setări ca `audiorecorder` și conținând exact eșantioanele stocate în acesta.

```
>> play(r);
```

De cele mai multe ori însă redarea simplă a sunetului înregistrat nu e suficientă. De obicei vrem să aplicăm o anumită prelucrare înainte de redare, caz în care trebuie să construim tot lanțul de înregistrare-redare:

```
>> r = audiorecorder(8000, 8, 1);
>> recordblocking(r, 5);
>> x = getaudiodata(r);
>> y = prelucrează(x);
>> a = audioplayer(y, 8000);
>> play(a);
```

9.3. Salvarea și încărcarea fișierelor audio

Pentru a lucra cu fișiere audio în Matlab se pot folosi funcțiile `audioread` și `audiowrite`. Acestea suportă principalele formate audio, de exemplu WAV, OGG sau MP3. Lista completă a formatelor suportate se poate obține cu comanda `audioformats`.

Salvarea unui semnal audio se face cu comanda

```
>> audiowrite(filename, s, fs);
```

Semnalul este precizat în parametrul `s`, frecvența de eșantionare în `fs`, iar numele fișierului sub forma unui șir de caractere în `filename`. Formatul salvat se decide pe baza extensiei din `filename`. Astfel, dacă vrem să salvăm semnalul generat anterior în formatul MP3 într-un fișier `'ton.mp3'` vom folosi comanda:

```
>> audiowrite('ton.mp3', s, 8000);
```

Pentru a încărca fișierul salvat, vom folosi comanda:

```
>> [x, fs] = audioread('ton.mp3');
```

Informațiile despre frecvența de eșantionare de data aceasta se găsesc în fișier și trebuie încărcate de acolo, de aceea, funcția întoarce două variabile, primul un vector ce conține eșantioanele și al doilea valoarea frecvenței de eșantionare.

9.4. Efecte folosite în prelucrarea audio

Bineînțeles, semnalele audio pot să fie supuse prelucrărilor obișnuite de semnale, cum ar fi schimbarea ratei de eșantionare sau filtrarea, dar există și anumite efecte special gândite pentru aceste semnale pornind de la modul cum este perceput sunetul de către urechea umană.

Aceste efecte sunt folosite pentru a îmbunătăți percepția și a simula un mod aparte de percepție audio. Efectele audio pot fi încadrate în trei grupe în funcție de domeniul în care este aplicat. Astfel putem avea efecte de amplitudine, de întârziere, respectiv spectrale. Efectele de amplitudine se aplică asupra valorilor eșantioanelor atenuând sau amplificând acestea. Un exemplu pentru astfel de efect este compandarea. Efectele de întârziere adaugă la semnal diferite versiuni întârziate în timp. Exemple pentru acestea includ ecoul, reverberația, efectul de cor sau tremolo. Efectele spectrale se aplică în mod diferențiat asupra diferitelor componente spectrale. Putem avea de exemplu efectul de fazor, care modifică fazele unor componente sau efectul pitch-shifting care modifică poziția componentelor spectrale.

Ecou

Cel mai simplu efect audio, aceasta practic implementează efectul sunetului reflectat de pe un obiect aflat la distanță. Implementarea efectului se bazează pe adăugarea unei copii atenuate și întârziate în timp, așa cum se poate observa pe figura

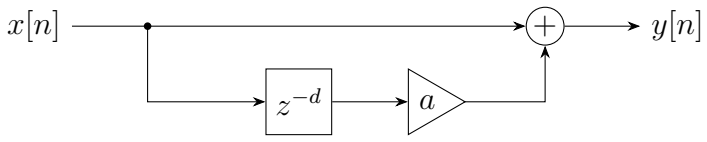


Figura 9.2. *Ecou simplu*

Pentru a realiza acest efect, trebuie suprapus peste semnalul original o copie atenuată a semnalului care este întârziată câteva zeci de milisecunde. De exemplu, următorul cod adaugă un ecou întârziat cu 160 de eșantioane (care, considerând o frecvență de eșantionare de 8 kHz, corespunde cu 20 ms) și atenuat cu 90 %:

```
>> d = 160;
>> a = 0.1;
>> y = [x(1:d); x(d + 1:end) + x(1:length(x)-d) * a];
```

Prin alterarea întârzierii și/sau a atenuării putem obține un ecou mai pronunțat sau mai subtil. O întârziere mare rezultă într-un ecou pronunțat, care este ușor de recunoscut ca și ecou. Întârzierile mai scurte (cum e și în exemplul dat) rezultă mai degrabă într-un efect care modifică timbrul sunetului sau modul de percepție, ceea ce poate fi folosit în scopuri artistice.

Pentru astfel de prelucrări, o implementare de tip IIR poate avea un efect mai confortabil pentru percepție decât implementarea FIR prezentată anterior. Pentru a avea o implementare IIR, întârzierea și atenuarea se pun într-o buclă de reacție (figura 9.3). Aceasta rezultă în ecouri multiple care dispar treptat.

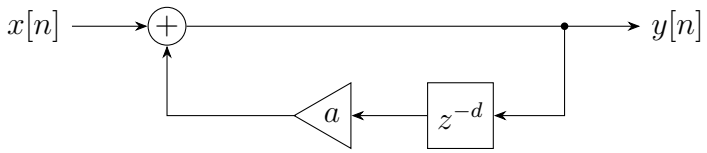


Figura 9.3. *Ecou multiplu prin feedback*

Implementarea unui ecou cu reacție necesită o funcție de felul următor:

ecou.m

```
function y = ecou(d,a,x)
  y = zeros(1,length(x));
  for n = 1:length(y)
    if (n > d)
      y(n) = x(n) + a*y(n-d);
    else
      y(n) = x(n);
    endif
  endfor
endfunction
```

Reverberație

Este o variantă mai specială a ecoului, care încearcă să simuleze ecourile multiple auzite într-o încăpere, unde sunetul ajunge la ascultător pe mai multe căi, reflectându-se de pe pereții aflați la distanțe diferite față de ascultător.

Pentru a crea un astfel de efect se folosește mai multe ecouri având diferite întârzieri și atenuări, așa cum se poate vedea pe figura 9.4.

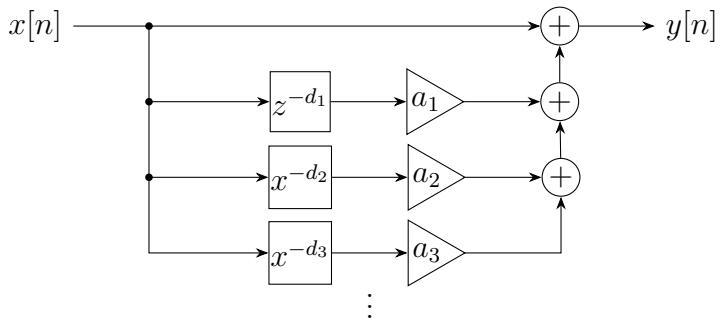


Figura 9.4. *Reverberație*

Cor

Acest efect simulează cântatul în cor pornind de la înregistrarea unui singur cântec. Corul este caracterizat prin faptul că deși se cântă aceeași melodie, între cântatul membrilor corului există o mică diferență în viteză respectiv amplitudinea vocii, astfel practic apar niște ecouri cu întârzierea variind între anumite limite pe tot parcursul înregistrării. O reprezentare schematică se poate vedea pe figura 9.5.

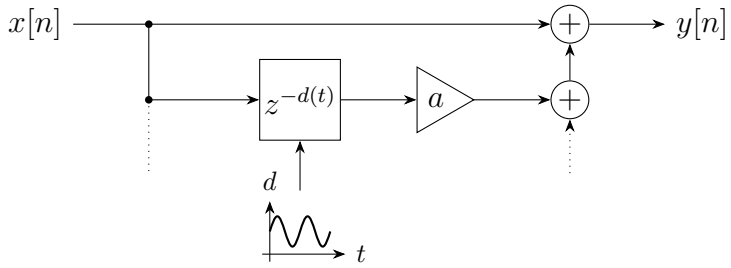


Figura 9.5. *Cor*

9.5. Exerciții

1. Înregistrați un semnal vocal de scurtă durată, pe care să implementați efectele audio prezentate anterior.

2. Folosiți diferite filtre pentru a prelucra semnalul, eliminați zgomotul de fond, observați separat basii și înaltele etc.

3. Încercați o prelucrare în frecvență a vocii, prin care transformați semnalul în spectru, mutați componentele spectrale mai la stânga sau la dreapta și transformați înapoi semnalul.

Lucrarea 10

Spectrograma semnalelor

De foarte multe ori se întâmplă ca analiza spectrală a semnalelor să nu fie suficientă pentru a caracteriza complet semnalul. Foarte multe tipuri de semnale sunt caracterizate prin faptul că spectrul lor variază în timp, iar informația principală nu este dată de spectrul la un moment definit de timp, ci de toată variația spectrului pe parcursul desfășurării semnalului. Pentru a analiza un astfel de semnal se folosește spectrograma, adică o reprezentare atât în timp cât și în frecvență a semnalului.

Analiza cu spectrograme este folosită foarte mult în prelucrarea de voce și de sunete, pentru a identifica elementele din semnalul audio, dar și în prelucrări de radiofrecvență pentru vizualizarea semnalelor de spectru larg. Un exemplu de semnale FT8 în banda de 10.136MHz este prezentat pe figura 10.1. Pe această figură semnalul este reprezentat cu spectrul desfășurându-se pe orizontală și timpul derulându-se pe verticală. Prin derularea timpului imaginea creată seamănă cu o cascadă, de aceea spectrograma se mai numește și imagine de tip waterfall.

Pe o astfel de spectrogramă, din moment ce axele Ox și Oy sunt folosite pentru reprezentarea frecvenței, respectiv a timpului, puterea semnalului în sine este codificată printr-o culoare. În exemplul dat, culoarea albastră reprezintă zgomotul obișnuit în bandă, iar culorile galben-portocaliu-roșu reprezintă vârfuri de putere a semnalului util. Astfel se poate vizualiza ușor existența unor transmisii în banda de interes și identifica

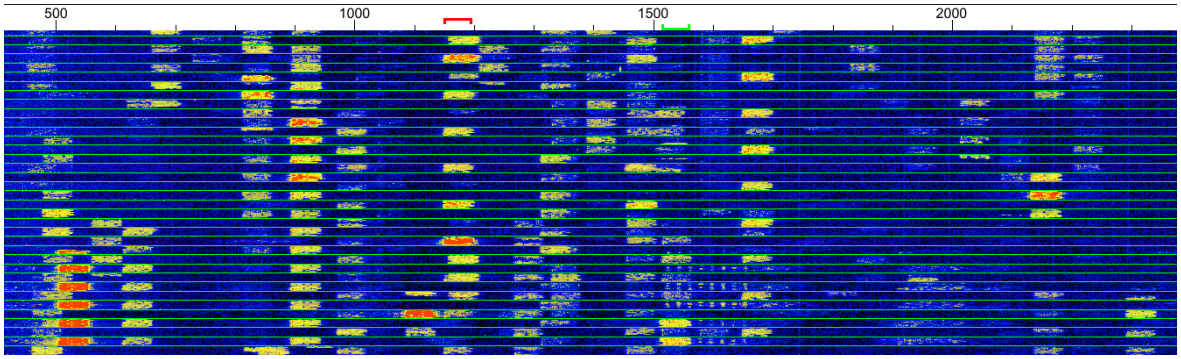


Figura 10.1. *Spectrograma unor semnale FT8 capturată cu WSJT-X*

ușor frecvența purtătoare, respectiv intervalul de timp în care transmisia este activă.

Pentru a calcula spectrograma unui semnal se folosește metoda unei ferestre glisante prin care se obțin bucăți din semnal căruia separat se calculează spectrul prin transformata Fourier. Fereastra folosită este caracterizată prin faptul că tinde spre 0 pe cele două capete iar ferestrele se vor suprapune parțial în timp, astfel putem reproduce evoluția spectrului în timp. Lungimea ferestrei alese stabilește atât rezoluția spectrală (dată de caracteristicile transformatei Fourier discrete) cât și rezoluția în timp a variației spectrului. Cu cât fereastra este mai lungă, cu atât rezoluția spectrului este mai bună, însă variațiile rapide vor fi suprapuse.

10.1. Calculul spectrogramei în Octave

Pentru a calcula o spectrogramă se poate folosi funcția `specgram` disponibil în pachetul `signal`. Sintaxa este următoare:

```
>> [s, f, t] = specgram(x, n, fs, window, overlap);
```

unde

- `s` este spectrograma calculată
- `f` este vectorul de frecvență pentru spectrogramă (dependent de rezoluția spectrală și de frecvența de eșantionare)
- `t` este vectorul de timp pentru spectrogramă (momentele de timp pentru fiecare bucată de spectru calculat)
- `x` este semnalul analizat
- `n` lungimea segmentului pentru care se calculează spectrul, dacă parametrul lipsește se consideră implicit valoare de 256
- `fs` este frecvența de eșantionare, dacă este precizat se va folosi pentru a raporta vectorul de frecvențe la această frecvență, altfel vectorul de frecvență întors este un vector normalizat la jumătatea frecvenței de eșantionare (oricare ar fi asta)
- `window` este fereastra folosită, implicit se va folosi o fereastră Hanning de aceeași lungime ca și segmentul, dar se poate crea o altă fereastră în funcție de necesități
- `overlap` este lungimea superpoziției între segmentele calculate, implicit se folosește jumătatea ferestrei.

Spectrograma poate fi afișată pe toate variantele de grafice tridimensionale suportate de Matlab, cu vectorul de timp `s` și vectorul de frecvență `f` pe două axe și semnalul pe a treia. De notat însă că matricea de semnal este calculată în eșantioane spectrale

complexe (așa cum e obișnuit la FFT), deci va trebui prelucrată înainte de a putea fi reprezentată pe grafic. În mod uzual se reprezintă magnitudinea spectrală calculând modulul eșantioanelor.

Totuși, cea mai uzuală reprezentare a spectrogramei este printr-o hartă colorată (`colormap`) pe un grafic bidimensional, în loc de folosirea unei grafice tridimensionale. Pe această hartă colorată, fiecare eșantion spectral este reprezentat de un pixel cu o nuanță de culoare sau cu intensitatea luminoasă dependentă de valoarea cuantizată.

Există mai multe posibilități de a codifica valorile în culori. Cel mai frecvent se folosește un model în care amplitudinilor mici corespund culorile reci (de la albastru la verde) iar amplitudinilor mari corespund culorile calde (de la galben la roșu). Figura 10.2 prezintă câteva hărți de culoare existente în Octave. Fiecare bară conține tranziția de culoare de la amplitudinea minimă la amplitudinea maximă corespunzătoare. Harta poate fi selectată prin comanda precizată lângă bara de culoare.

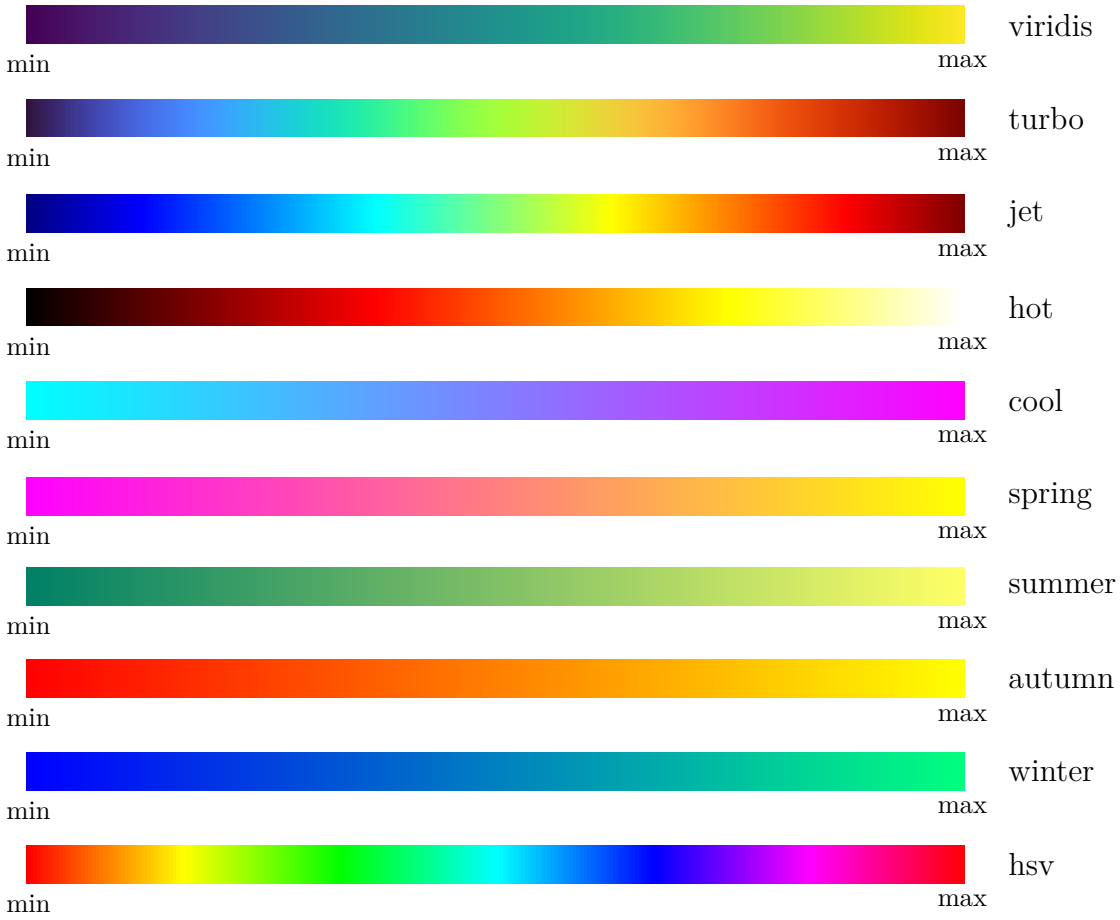


Figura 10.2. *Modele de culoare întâlnite în Octave*

Dacă nu se precizează parametrii de ieșire la funcția `specgram`, aceasta va afișa spectrograma folosind această abordare de afișare bidimensională colorată.

10.2. Exemple de spectrograme

Semnalul chirp

Cea mai simplă funcție care poate fi analizată în detaliu numai pe spectrogramă este funcția `chirp`, care creează un semnal sinusoidal cu o frecvență liniar crescătoare (fig. 10.3a). Astfel de semnale sunt folosite, de exemplu, pentru a baleia intrarea într-un vobuloscop.

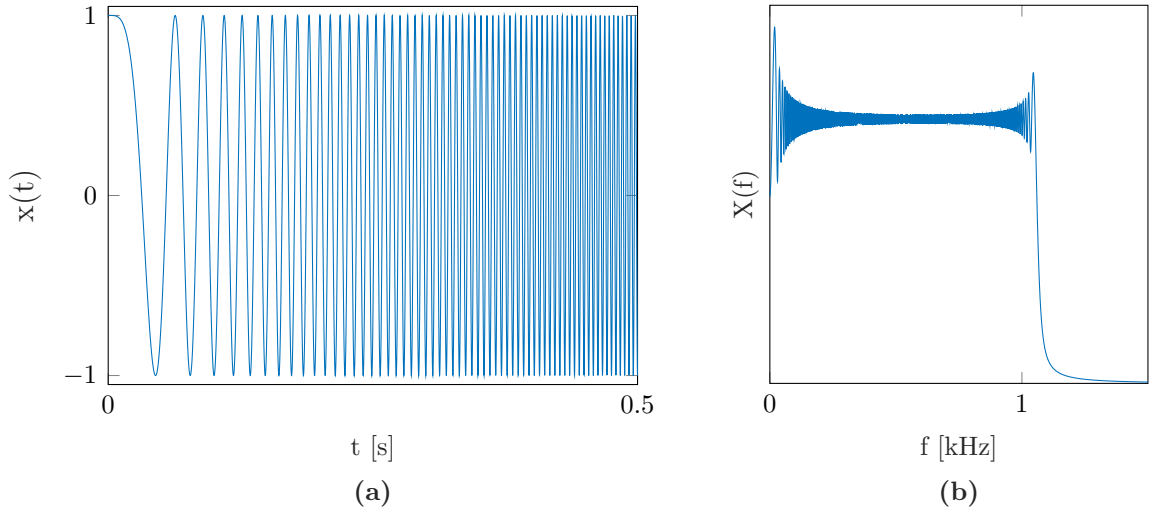


Figura 10.3. *Semnal chirp reprezentat în timp a și în spectru b*

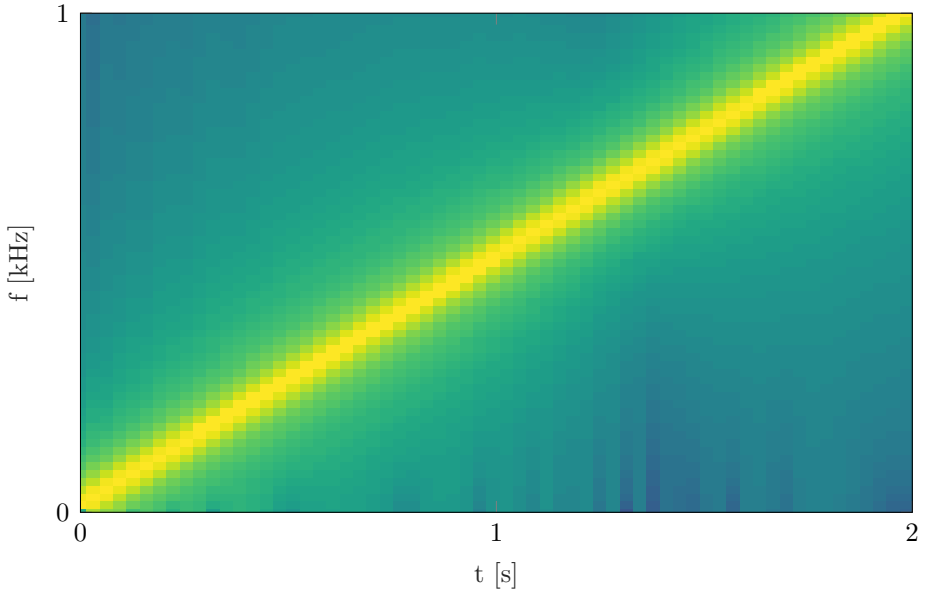


Figura 10.4. *Spectrograma unui semnal chirp*

La o analiză spectrală cu transformata Fourier pe întreg semnalul, vom obține un spectru uniform distribuit (fig. 10.3b), din care este foarte greu de determinat ce tip de semnal este. Pe de altă parte, semnalul `chirp` poate fi foarte ușor de recunoscut pe o spectrogramă care se va prezenta ca o dungă liniară corespunzătoare creșterii frecvenței în funcție de timp. Pe figura 10.4 se poate vedea componenta principală în frecvență de culoare galbenă cum se mută de la 0 la 1000Hz pe parcursul a 2s.

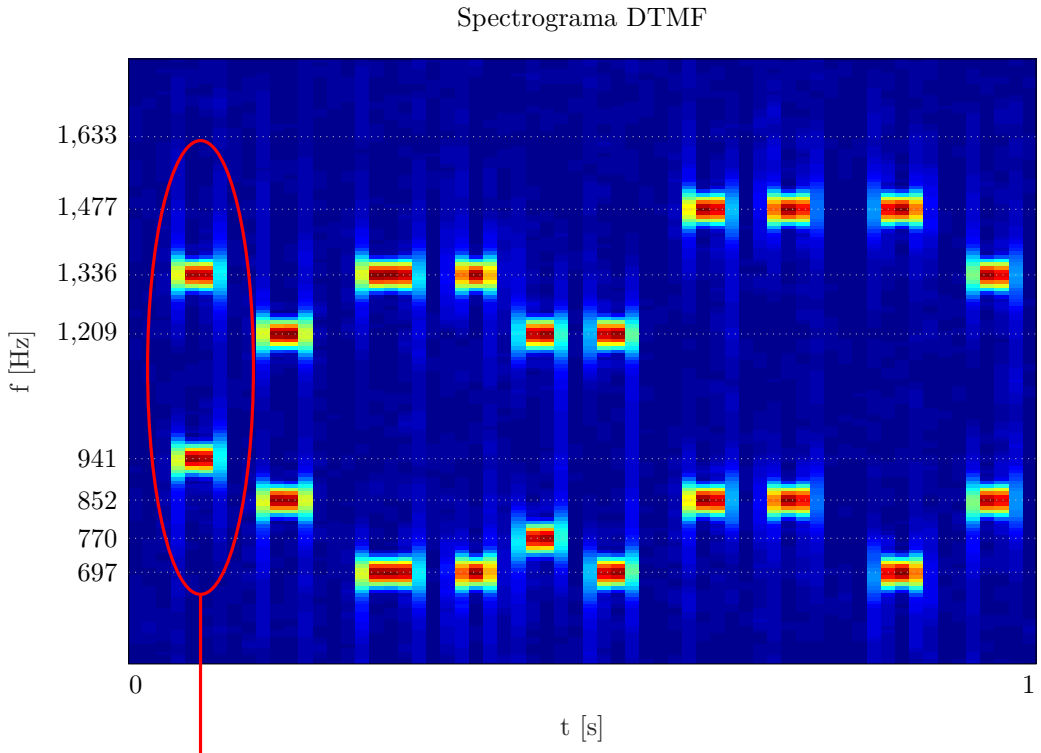
Semnalul și figurile sunt generate cu următorul cod Octave:

chirpgram.m

```
% generare semnal chirp cu frecvenţa între 0-1000Hz, durata 2s
% frecvenţa de eșantionare 8kHz
fs = 8000;
dt = 2;
n = dt*fs;
t = 0:1/fs:dt+511/fs;
c = chirp(t,0,2,1000);

% afișăm semnalul în timp
fig1 = figure;
plot(t,c);
% limităm afișarea la primele 500ms, pentru că la frecvenţe mai
% mari imaginea devine prea compactă
axis([0, 0.5, -1.05, 1.05]);
xlabel('t [s]');
ylabel('x(t)');
xticks([0,0.5]);
yticks([-1,0,1]);

% afișăm spectrul semnalului
w = 0:1/dt:fs/2-1/dt;
fig2 = figure;
plot(w,abs(fft(c))(1:n/2)/(n));
% și în frecvenţă ne limităm pentru zona de interes
axis([0, 1500, 0, 0.016]);
xlabel('f [Hz]');
ylabel('X(f)');
xticks([0, 1000]);
yticks([]);

% și la final să vedem și spectrograma
fig3 = figure;
specgram(c, 512, fs);
% și aici ne limităm la gama de frecvenţe de interes
axis([0,2,0,1000]);
xlabel('t [s]');
ylabel('f [Hz]');
```

Semnale DTMF

Un alt exemplu foarte elocvent pentru a demonstra utilitatea spectrogramei este analiza semnalelor DTMF studiate în lucrarea 8. Un șir de mai multe cifre (de exemplu, un număr de telefon) va avea mai multe perechi de componente spectrale desfășurându-se în timp (figura 10.5).



	1209 Hz	1336 Hz	1477 Hz	1633Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	**0**	#	D

Figura 10.5. *Identificare cifrelor de pe spectrograma unui semnal DTMF*

Fiecare cifră alcătuită de o pereche de tonuri poate fi ușor distinsă pe spectrogramă (folosind o rezoluție suficient de bună, așa cum a fost stabilit în lucrarea 8). Din perechea de tonuri, a căror frecvență putem citi de ordonată, folosind tabelul frecvențelor, identificăm linia și coloana, și la final simbolul transmis. De exemplu, din perechea marcată cu roșu pe figură putem identifica simbolul **0**.

Semnale vocale

Folosirea spectrogramelor este larg răspândită și în prelucrare audio. Analiza vocală folosește preponderent această tehnică, unde informația auditivă practic se regăsește în variația componentelor spectrale în timp.

Un exemplu de spectrogramă vocală este prezentat pe figura 10.6. S-a folosit utilitarul `audiorecorder` studiat la lucrarea 9 pentru a capta un semnal vocal:

voice.m

```
% Captare semnal vocal (3 secunde, setări implicite)
r = audiorecorder(44100,16,1);
recordblocking(r, 3);
x = getaudiodata(r);
fs = get(r, "SampleRate");

% calcularea spectrogramă cu următoare setări:
% fereastră de 40ms luate pe fiecare 5ms
% lungimea FFT rotunjită la putere a lui 2
window = round(40e-3 * fs);
overlap = round(35e-3 * fs);
n = 2^nextpow2(window);

[S, f, t] = specgram(x, n, fs, window, overlap);

% spectrograma se poate limita la un interval de frecvență mai
% restrâns, de exemplu 40 - 4000Hz
% iar magnitudinea normalizată pe o scală logaritmică
i1 = round(40*n/fs); i2 = round(4000*n/fs);
S = abs(S(i1:i2,:));
S = S/max(S(:));

% folosind imagesc se afișează spectrograma
imagesc (t, f(i1:i2), log(S));
```

Deoarece semnalul vocal are niște caracteristici mai specifice, nu s-a folosit afișarea implicită de la `specgram` ci mai întâi s-au operat anumite modificări asupra rezultatului. În primul rând s-a ales o fereastră mai lungă care se suprapune într-o proporție mai mare (în exemplul dat, fereastra e de lungime de 40 ms cu o suprapunere de 35 ms, adică vom întocmi spectrograma din 5 în 5 ms, dar luând o lungime mai mare o să avem rezoluție spectrală suficientă pentru FFT. De asemenea, deși captarea s-a realizat la o frecvență de eșantionare de 44 100 Hz (obișnuită în prelucrare audio), un semnal vocal este de obicei restrâns la un interval de frecvență mai joasă, așa că nu merită să afișăm întregul spectru captat. În acest sens s-a limitat vectorul de frecvențe și semnalul la un interval de 40 – 4000 Hz. Iar la final s-a folosit o scară logaritmică, oportună pentru reprezentarea vocii umane.

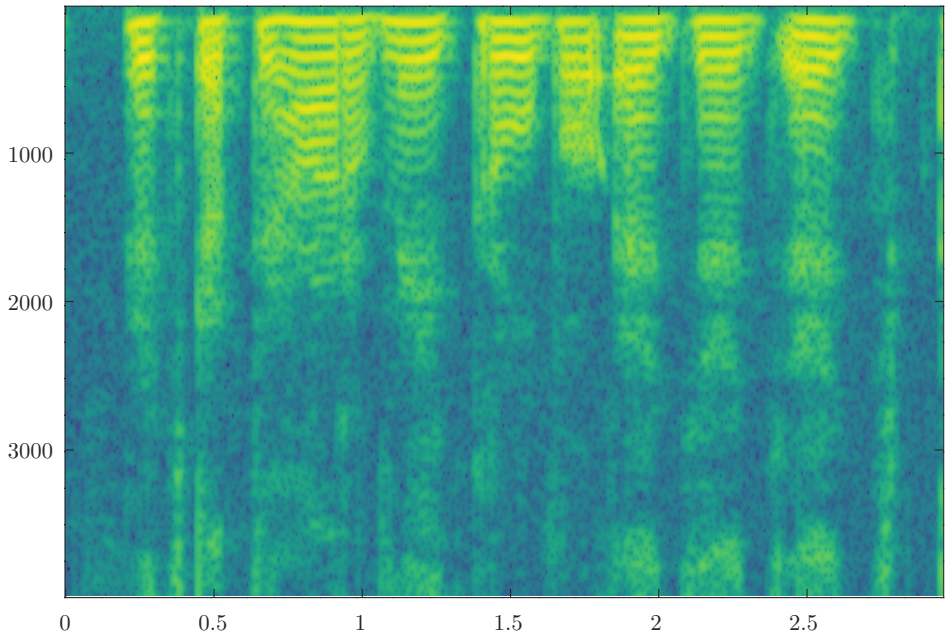


Figura 10.6. *Spectrograma unui semnal vocal*

10.3. Exerciții

1. Vizualizați spectrograma propriului voce preluat cu modulul `audiorecorder`.
2. Vizualizați spectrograma semnalului DTMF atașat și identificați cifrele transmise.

Anexa A

Noțiuni de bază Matlab / Octave

Limbajul Matlab este un limbaj de nivel înalt ce permite efectuarea calculelor matematice fără a fi nevoie de implementarea algoritmilor de calcul într-un limbaj de nivel jos. Acesta a fost creat cu scopul de a permite un acces ușor la bibliotecile de calcul matricial realizate în FORTRAN, printr-o formă simplă, apropiată de scrisul matematic obișnuit [7]. Denumirea limbajului duce și el spre orientare către calculul matricial: *MAT*rix *LAB*oratory.

Matlab este un limbaj interpretat, având nevoie de un interpretor pentru efectuarea propriu-zisă a calculelor. Interpretorul este oferit în suita de programe **MATLAB**, oferit de compania MathWorks[1], o suită comercială, care pe lângă interpretor oferă și o sumedenie de biblioteci (numite toolboxuri) pentru o gamă largă de domenii, printre care și prelucrarea semnalelor (**Signal Processing Toolbox**).

Pe lângă soluția comercială există și o variantă open-source pentru interpretarea limbajului Matlab, numit **Octave**[2], oferit prin intermediul proiectului GNU. Și pentru acest program există un număr de biblioteci, deși un număr mai mic și cu facilități mai limitate decât în MATLAB. Pentru prelucrare de semnale există pachetul **signal**[3].

Deși cele două programe nu sunt 100% compatibile (vezi anexa C), marea majoritate a programelor create într-unul din aceste programe pot fi rulate fără probleme și în celălalt. Diferențele importante, cum ar fi disponibilitatea a mai multor toolbox-uri în MATLAB, se pot sesiza numai în cazul problemelor specializate. Pentru realizarea lucrărilor de laborator din acest îndrumar sunt suficiente toolbox-urile de procesare a semnalelor oferite de ambele variante de interpretor Matlab.

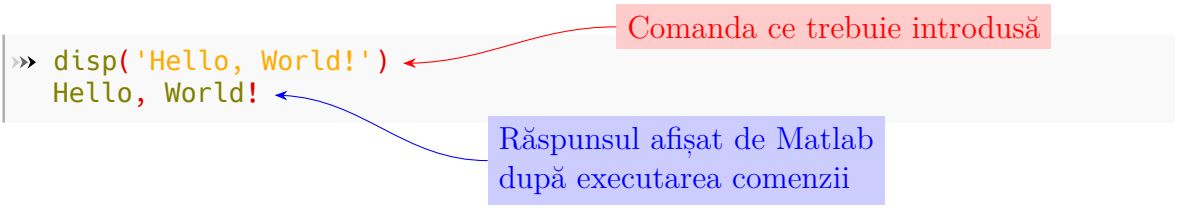
Atât MATLAB cât și Octave oferă un mediu integrat, care conține ca și element principal un *Command Window*, o linie de comandă prin care pot fi executate comenzi folosind limbajul Matlab (vezi figura). Liniile de cod Matlab pot fi introduse pe această linie de comandă, iar ele vor fi interpretate și executate pe loc la apăsare tastei Enter. Interpretarea se face linie cu linie, și datele intermediare sunt disponibile imediat.

[1] https://www.mathworks.com
[2] https://octave.org
[3] https://packages.octave.org

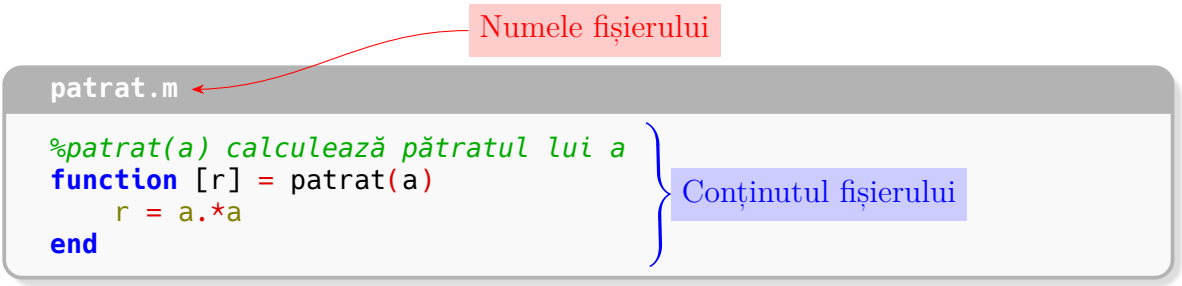
Codul Matlab poate fi introdus și în fișiere sursă (atât MATLAB cât și Octave dispunând și de un editor încorporat), care după aceea poate fi lansat în execuție din linia de comandă și va fi interpretat ca și cum fiecare linie ar fi fost introdusă una după cealaltă în linia de comandă.

Pe parcursul acestui îndrumar s-a folosit următoarea notație:

- pentru comenzi ce se introduc direct în linia de comandă se folosește un chenar deschis, în care liniile care trebuie introduse sunt marcate cu simbolul »» , iar liniile care nu conțin acest simbol conțin mesajul afișat de Matlab la executarea comenzii respective. De exemplu următoarea secvență poate fi folosit pentru a afișa celebrul mesaj *Hello, World!* în Matlab:



- pentru exemple mai lungi, scripturi și funcții se folosește un chenar închis, în care este marcat numele fișierului și codul ce trebuie introdus în fișier. Codul trebuie introdus în întregime în fișier (se poate folosi editorul încorporat prin meniul *New Script* sau *New Function*), salvat sub numele indicat după care poate fi lansat în execuție din linia de comandă folosind numele fișierului fără extensie. În următorul exemplu se creează un fișier numit **patrat.m** conținând o funcție ce poate fi folosită pentru a calcula pătratul unui număr:



Pentru a calcula pătratul unui număr folosind funcția creată, se apelează din linia de comandă funcția sub forma:

```
»» patrat(5)
ans =
      25
```

La apelarea unei funcții, Matlab va căuta în directorul curent de lucru un fișier cu numele funcției și extensia **.m** și va executa automat acel fișier. Acestea au întotdeauna prioritate față de nume de variabile sau de funcții interne, de aceea trebuie avut grijă ca să nu se folosească nume de fișiere care sunt identice cu numele unor funcții interne

sau variabile din Matlab, acestea din urmă devenind inaccesibile în prezența unui fișier `.m` în directorul curent.

Tipuri de date

Elementul de bază al operațiilor Matlab este matricea, toate variabilele sunt considerate ca fiind de tipul matrice. Astfel, toate operațiile matematice disponibile în Matlab sunt definite ca operații matriciale, iar toate funcțiile interne sunt capabile să opereze pe matrici.

Ca și tipuri de date, Matlab recunoaște șiruri de caractere (recunoscute ca un vector unidimensional de caractere individuale), matrici numerice și obiecte (vectori de elemente care pot fi de diferite tipuri). Numerele în mod implicit sunt stocate în virgulă mobilă, dublă precizie (`double`). Pot fi definite și alte tipuri (de exemplu întregi pe 8, 16, 32 de biți sau virgulă mobilă simplă precizie) în mod explicit. Acestea sunt mai degrabă folosite pentru optimizarea stocării datelor și la comunicare prin API-uri externe. În cazul în care însă ele sunt supuse unor operații numerice, vor fi automat convertite la `double`. De asemenea, dacă operații presupun lucrul cu numere complexe, atunci reprezentarea va fi automat convertită în numere complexe (partea reală și partea imaginară pe câte un `double`).

Valorile numerice automat sunt organizate în matrici. Matlab permite folosirea matricilor multidimensionale, limitarea fiind dată doar de dimensiunea memoriei disponibile.

Un caz particular este când matricea respectivă conține un singur element (aceasta este folosită pentru reprezentarea valorilor scalare) sau toate dimensiunile unei matrici cu excepția uneia sunt 1 (reprezentând un vector). Aceste cazuri sunt tratate în mod special în anumite situații:

- operațiile pe un scalar sunt automat definite ca operații matematice scalare
- dacă operația necesită o matrice, o valoare scalară va fi automat extinsă prin duplicarea valorii pe toate elementele matricii de dimensiunea necesară
- un vector poate fi adresat cu aceiași indici indiferent dacă e vorba de vector linie sau vector coloană

Variabile

Variabilele în Matlab sunt definite automat la momentul atribuirii cu operatorul `=`. Numele de variabilă poate fi format din caracterele alfanumerice și _, pe prima poziție nefiind permise cifre. Tipul variabilei este asociat automat în funcție de rezultatul operației. Atribuirea valorii poate fi directă:

```
>> x = 10
   x =
       10
```

sau poate fi rezultatul unei operații:

```
>> y = x + 1
   y =
       11
```

După fiecare linie de cod, Matlab afișează rezultatul operației, care în cazul atribuirii unei variabile este variabila însăși. Dacă rezultatul operației nu este stocat într-o variabilă, atunci implicit se va folosi o variabilă denumită `ans`:

```
>> 1 + 2
   ans =
       3
```

Pentru a preveni afișarea rezultatului la fiecare linie de cod executată, se folosește caracterul `;` introdus la sfârșitul liniei;

```
>> z = 2;
```

Pentru definirea unor matrici se folosește operatorul `[]`. Aceasta permite concatenarea elementelor (ce pot fi valori scalare, alte matrici sau rezultatul unor operații) pe linie și/sau pe coloană. Elementele pe linie vor fi separate cu spațiu sau cu virgulă, iar liniile între ele cu punct și virgulă.

Astfel, pentru a crea un vector de tip linie se folosește comanda:

```
>> v1 = [1 2 3]
   v1 =
       1 2 3
```

un vector coloană cu comanda:

```
>> v2 = [1;1;2]
   v2 =
       1
       1
       2
```

iar o matrice 2×2 cu comanda:

```
>> m = [1 2; 2 3]
   m =
       1 2
       2 3
```

Pe lângă concatenarea directă a elementelor, un vector conținând o progresie aritmetică poate fi creat și cu operatorul `:`, având următoarea sintaxă:

```
>>  valoare de început : increment : valoare de sfârșit
```

Incrementul este opțional, dacă nu e prezent, atunci se consideră implicit un increment de 1.

Exemple pentru crearea unor progresii aritmetice:

```
>> i = 1:2:9
   i =
       1 3 5 7 9
>> j = 2:4
   j =
       2 3 4
```

Accesarea unui element dintr-o matrice se realizează cu operatorul `( )`. Acest operator poate fi folosit atât pe partea din dreapta a ecuației (pentru a extrage un element) cât și pe partea din stânga a ecuației (pentru a modifica un element). Elementul modificat poate fi și o submatrice (în orice fel de ordine).

```
>> m(1,2)
   ans =
       2
>> m(1,1:2)
   ans =
       1   2
>> m([2 1], [2 1])
   ans =
       3   2
       2   1
>> m(1,1:2) = [2 3]
   m =
       2   3
       2   3
```

Operatori

Matlab cunoaște operațiile matematice de bază reprezentate cu simbolurile:

`+`	adunare
`-`	scădere
`*`	multiplicare
`/`	împărțire din dreapta
`\`	împărțire din stânga
`^`	ridicare la putere

Spațiile goale (spațiu sau tabulator) nu au influență asupra modul de interpretare a operatorilor. Exemplu de folosire:

```
>> 1+2
ans =
     3
>> a = 2.5 * 2 - 1
a =
     4
>> b = a / 2;
>> c = b^5
c =
    32
```

Operațiile aritmetice prezentate anterior sunt considerate a fi operații matriciale atunci când sunt executate asupra unor matrici. Operatorii `*`, `\`, `/` și `^` toți sunt considerați ca operatori matriciali când operează asupra unor matrici. Din acest motiv avem și două variante diferite de operatori de împărțire. Din moment ce înmulțirea matricială nu este comutativă, operația inversă trebuie să aibă și o anume direcție în care se va efectua: poate fi făcută dinspre stânga cu `a\b` (`a` împarte pe `b`) sau din dreapta cu `a/b` (`a` este împărțit cu `b`).

De multe ori însă se dorește a realiza o operație scalară între elementele de pe poziții corespunzătoare din două matrici. Pentru a executa o astfel de operație element cu element, operatorul trebuie precedat de `.`, astfel vom avea operatorii `.*`, `.\`, `./` și `.^`.

De exemplu, două matrici pot fi înmulțiți matricial respectiv element cu element în felul următor:

```
>> a=[1 2; 0 1]; b=[2 1;4 3];
>> a*b
ans =
    10     7
     4     3
>> a.*b
ans =
     2     2
     0     3
```

Când sunt realizate operații asupra unor matrici, fie că e vorba de operație matricială sau element cu element, matricile trebuie să aibă dimensiuni compatibile:

- pentru operații element cu element, cele două matrici trebuie să aibă exact aceeași dimensiune
- pentru înmulțire matricială, numărul de coloane din primul operand trebuie să fie egal cu numărul de linii din al doilea operand
- împărțirea matricială poate fi executată numai dacă divizorul este o matrice inversabila
- ridicarea la putere poate fi realizat numai între o matrice și un scalar

Un alt operator care este definit în contextul operațiilor cu matrici este transpunerea. Aceasta poate fi realizată prin operatorul `'` (transpusa complex conjugată) sau `.'` (transpusa fără conjugare):

```
>> m = [1 2; 3 4];
>> m'
   ans =
       1   3
       2   4
```

Restul operațiilor, care sunt uzual definite pe matrici, sunt realizate prin intermediul unor funcții dedicate, de exemplu `inv` pentru calcularea inversei unei matrici sau `det` pentru calcularea determinantului:

```
>> inv(m)
   ans =
      -2.0000   1.0000
       1.5000  -0.5000
>> det(m)
   ans =
      -2
```

Funcții

Comenzile Matlab pot fi grupate împreună în funcții. Acestea pot primi un set de parametri și la final returnează un set de rezultate. O funcție este declarată folosind cuvântul cheie **`function`** în felul următor:

```
>> function [r_1, r_2, ... r_n] = nume_funcție(p_1, p_2, ... p_m)
       ...
>> end
```

unde r_1 … r_n sunt variabilele în care se va stoca rezultatele funcției, iar p_1 … p_m sunt parametrii funcției. În corpul funcției, înainte de **`end`**, fiecare variabilă specificată ca rezultat va trebui să fie asignată.

Definirea funcției poate fi realizată în linia de comandă, caz în care interpretorul este suspendat până la introducerea cuvântului cheie **end**. Astfel pot fi introduse mai multe linii de cod. Totul va fi interpretat doar după ce funcția a fost finalizată.

După ce funcția a fost definită, aceasta poate fi apelată în linia de comandă sub forma:

```
>> [r1 r2 ... rn] = nume_funcție(p1, p2, ... pm)
```

$r1$ ... r_n sunt nume de variabile în care se vor plasa valorile returnate de funcție, $p1$ ... p_m sunt parametri. Dacă funcția returnează o singură valoare, atunci parantezele drepte nu sunt necesare. Dacă nu sunt specificate variabilele pentru rezultate, atunci prima valoare rezultată din funcție se stochează în variabila specială `ans`, iar restul se pierd. La fel, dacă nu se specifică suficiente variabile pentru toate rezultatele (în cazul în care funcția are mai multe valori returnate) atunci valorile rezultate în plus se vor pierde.

Deși funcțiile pot fi definite și în linia de comandă, de cele mai multe ori este preferată salvarea lor într-un fișier sursă. Sursele în Matlab au extensia **.m** și trebuie să aibă exact aceeași denumire ca și funcția implementată în interior.

Fișiere **.m** care definesc funcții de asemenea trebuie să înceapă cu comanda **function** care definește funcția respectivă. Fișierul poate conține și alte funcții, dar acelea vor rămâne funcții interne, nefiind accesibile din linia de comandă. Înainte de cuvântul cheie **function** nu pot fi alte comenzi Matlab, poate fi adăugat însă un comentariu, care în acest caz va fi considerat documentația funcției respective, accesibilă prin comanda `help`.

La apelarea unei funcții, Matlab va efectua o căutare în directorul curent după un fișier cu numele funcției și extensia **.m**. Dacă găsește acel fișier, atunci va încărca fișierul și va executa funcția din fișier. Dacă nu există un fișier cu numele căutat în directorul curent, atunci se va căuta funcția printre funcțiile interne și din toolbox-urile instalate. De aceea, trebuie avut foarte mare grijă de a nu crea fișier **.m** cu nume identic cu o funcție internă, căci acesta va acoperi funcția internă, care astfel devine inutilizabilă.

Structuri de control

Pentru implementarea funcțiilor, pe lângă operații matematice simple, poate fi nevoie și de structuri de control. Matlab permite implementarea buclelor și a ramurilor de execuție condiționată. Pentru a crea o buclă se pot folosi structurile **for** sau **while**, iar pentru ramuri condiționate **if** – **else**.

Bucla **for** va parcurge un vector atribuind unei variabile *contor* valoarea fiecărui element din *vector* pe rând și executând corpul buclei câte o dată pentru fiecare valoare în parte. Sintaxa acestei comenzi este următoarea:

```
for contor = vector
    ...
end
```

Pentru a crea o variabilă care se incrementează după fiecare execuție a corpului buclei (similar cu alte limbaje de programare), se poate folosi operatorul `:` pentru a crea o progresie aritmetică ce va fi parcursă de contor:

```
for contor = început : pas : sfârșit
    ...
end
```

Bucla **while** este o structură repetitivă condiționată, corpul buclei se repetă atât timp cât condiția logică asociată buclei este adevărată.

```
while condiție
    ...
end
```

Structura **if** – **else** permite execuția condiționată a unor secvențe de cod în funcție de valoarea de adevăr a unei condiții. Sintaxa acestuia este:

```
if condiție
    ... % ramura executată când condiţia este adevărată
else
    ... % ramura executată când condiţia este falsă
end
```

Ramura **else** este opțională, poate să lipsească de tot, sau poate introduce și condiții adiționale prin mai multe ramuri **else if**.

Condițiile logice precizate la **while** respectiv **if** trebuie să fie variabile logice sau expresii ce rezultă într-o valoare logică. Acestea pot fi obținute în Matlab doar prin folosirea unor sau mai multe operatori logici:

==	egal
~=	diferit
<	mai mic
<=	mai mic sau egal
>	mai mare
>=	mai mare sau egal
&	ȘI logic
|	SAU logic
~	negare

Trebuie notat faptul că Matlab, fiind orientat asupra operațiilor cu matrici, eficiența operațiilor intrinseci pe matrici este mult mai bună decât executarea operațiilor pe fiecare element în parte. Folosirea buclelor pentru parcurgerea matricilor ar trebui evitată pe cât posibil, dacă poate fi înlocuită cu o operație matricială, folosirea lor este oportună numai în cazul operațiilor care neapărat trebuie executate secvențial.

De exemplu, pentru a calcula o sumă ponderată a elementelor unui vector `v` cu ponderile aflate în vectorul `p` se va folosi instrucțiunea:

```
>> s = v * p';
```

în loc de o buclă de forma:

```
>> for i = 1 : length(v)
>>     s = s + v(i) * p(i);
>> end
```

Anexa B

Grafice în Matlab

Pe lângă interpretarea și executarea codurilor sursă în mod text, Matlab este capabil de generare unor figuri. O figură este o fereastră separată, care poate conține una sau mai multe grafice. Un grafic, în contextul Matlab, este reprezentarea vizuală a unui set de puncte conectate cu segmente de linie între ele. Punctele pot fi exprimate în coordonate carteziene sau coordonate polare pe un sistem de axe liniare sau logaritmice.

Un exemplu simplu este prezentat pe figura B.1, care afișează două segmente de linie care conectează punctele cu coordonatele $(0, 0)$, $(2, 3)$ și $(4, 2)$.

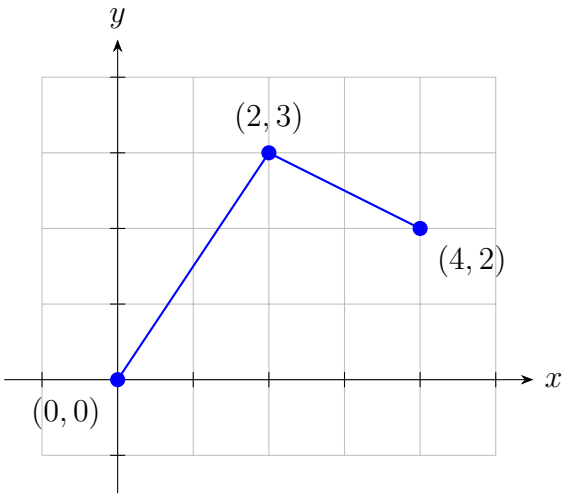


Figura B.1. *Exemplu de grafic*

În Matlab, un astfel de grafic în coordonate carteziene se creează cu comanda `plot`, care primește doi parametri x și y sub forma `plot(x,y)`, unde x este un vector care conține coordonatele de pe abscisă (axa Ox), iar y este un vector care conține coordonatele de pe ordonată (axa Oy). Cei doi vectori trebuie să fie de lungimi egale, și perechile de elemente de pe aceeași poziție a vectorului definesc coordonatele punctelor de pe grafic, iar segmentele de linie le conectează în ordinea în care ele se regăsesc în vectori.

Astfel pentru a crea un grafic similar cu cel din figura B.1, conținând punctele cu coordonatele $(0, 0)$, $(2, 3)$ și $(4, 2)$, vom avea nevoie de vectorul x conținând abscisele din coordonate, adică $[0, 2, 4]$ și vectorul y conținând ordonatele $[0, 3, 2]$. Executând comanda:

```
>> plot([0, 2, 4], [0, 3, 2]);
```

vom obține graficul din figura B.2.

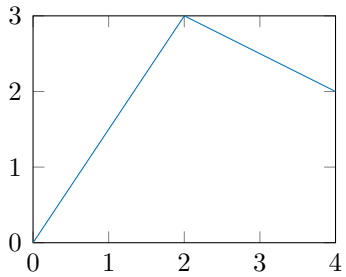


Figura B.2. *Plot simplu*

La crearea graficului, Matlab va determina automat axele necesare în așa fel încât să cuprindă toate coordonatele. Utilizatorul poate specifica și în mod arbitrar limitele axelor folosind funcția `axis`. Aceasta trebuie executată după funcția `plot` și primește ca parametru un vector ce conține limitele axelor Ox și Oy. De exemplu, pentru a extinde axele din exemplul precedent, se poate executa comanda:

```
>> axis([-1, 5, -1, 4]);
```

Aceasta va rezulta în graficul de pe figura B.3.

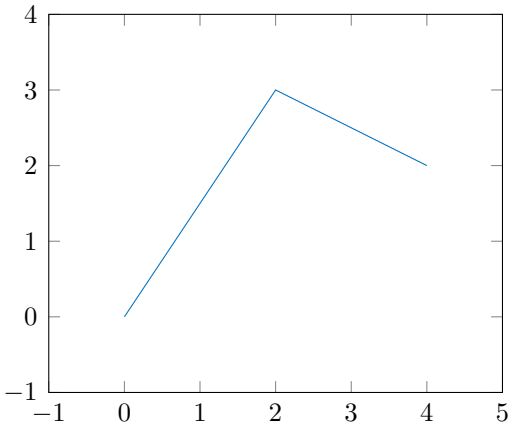


Figura B.3. *Plot cu axe extinse*

După crearea figurii cu `plot` se poate activa o grilă în fundal pentru a ajuta la identificarea mai bună a coordonatelor de pe grafic folosind comanda:

```
>> grid on
```

Graficul rezultat după aceasta se poate vedea pe figura B.4.

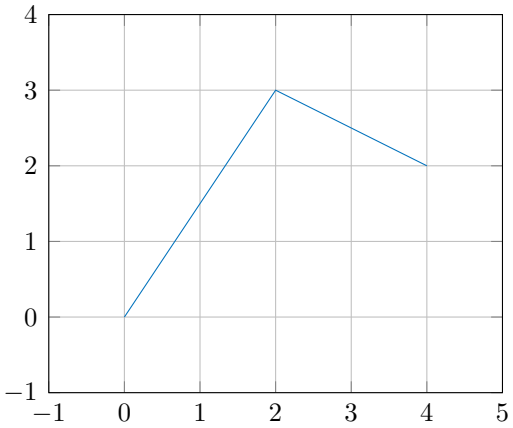


Figura B.4. *Plot cu grila activată*

Grila și marcajul de pe axe este poziționat implicit de Matlab cu o distribuție uniformă și valori cât mai rotunde. Un control mai precis asupra marcajelor pe axe se poate obține cu funcțiile `xticks` și `yticks`. De exemplu, aplicând comenzile:

```
>> xticks([0,2,4]);
>> yticks([0,2,3]);
```

pe figura anterioară va rezulta în figura B.5

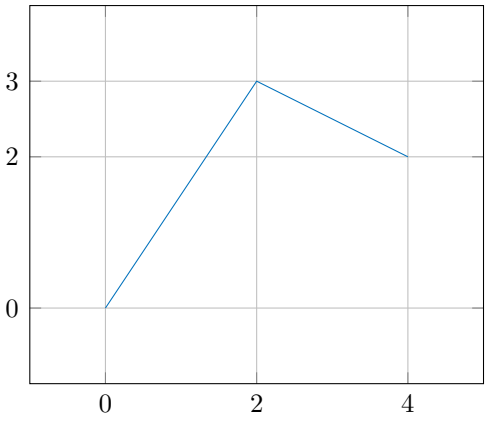


Figura B.5. *Plot cu marcaje pe axe specifice*

Funcția `plot` permite și specificarea unor opțiuni de afișare pentru liniile desenate, cum ar fi culoarea, tipul liniei și marcajul coordonatelor. Acestea pot fi codificate în al treilea parametru al funcției sub forma unui șir de caractere conținând simboluri din tabela B.1.

Tabela B.1. *Opțiunile pentru liniile din grafice*

Culoare		Tipul liniei		Tipul marcajului	
`r`	roșu	`-`	linie continuă	`.`	punct
`g`	verde	`--`	linie întreruptă	`+`	cruce
`b`	albastru	`:`	linie punctată	`o`	cerc
`y`	galben	`-.`	linie punct	`*`	stea
`m`	magenta			`x`	X
`c`	cian			`|`	liniuță verticală
`w`	alb			`-`	liniuță orizontală
`k`	negru			`s`	pătrățel
				`d`	romb
				`^`	triunghi (vârful spre sus)
				`v`	triunghi (vârful spre jos)
				`>`	triunghi (vârful spre dreapta
				`<`	triunghi (vârful spre stânga)
				`p`	pentagramă
				`h`	hexagramă

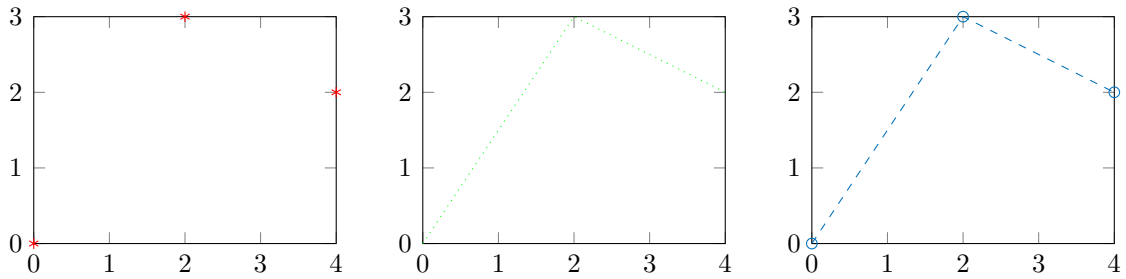
Se poate înșirui câte un simbol din fiecare coloană. Dacă specificarea culorii lipsește, atunci Matlab va aloca automat o culoare din lista implicită începând cu albastru. Dacă tipul liniei și tipul marcajului lipsesc atunci implicit se desenează o linie continuă fără marcaj. Dacă tipul liniei este prezent și tipul marcajului lipsește atunci coordonatele nu vor fi marcate. Dacă lipsește tipul liniei și e prezent tipul marcajului atunci doar marcajul coordonatelor va apare, fără a fi legate între ele cu linie.

De exemplu următoarele linii de cod vor genera graficele de pe figura B.6. Vom avea o figură cu doar marcajele coordonatelor de tip stea de culoare roșie, fără linie între ele, o figură cu linie punctată de culoare verde, fără marcaje, și o figură cu linie întreruptă de culoare implicită (albastru) cu coordonatele marcate cu cerculețe.

```
»» figure; plot([0 2 4], [0 3 2], 'r*');
»» figure; plot([0 2 4], [0 3 2], 'g:');
»» figure; plot([0 2 4], [0 3 2], '--o');
```

Comanda `figure` înaintea fiecărui `plot` asigură crearea unei ferestre noi, astfel comenzile `plot` nu vor suprascrie cele precedente.

Opțiuni adiționale, cum ar fi dimensiunile liniilor, a marcajelor, combinații adiționale de culoare etc. pot fi specificate cu perechi adiționale de parametri sub forma unor



(a) `r*` *(b)* `g:` *(c)* `-o`

Figura B.6. *Diferite formatări de plot*

șiruri de caractere ce specifică proprietatea respectiv valoarea dorită. Cele mai uzuale proprietăți sunt `"linestyle"`, `"linewidth"`, `"color"`, `"marker"`, `"markersize"`, `"markeredgecolor"` și `"markerfacecolor"`.

Un exemplu complet pentru a reproduce desenul din figura B.1, cu marcajul solid pe coordonate este dat de codul următor rezultând în figura B.7. Adițional, s-a adăugat un titlu deasupra graficului cu `title` și etichete pe axe cu `xlabel` și `ylabel`.

```
>> plot([0 2 4],[0 3 2],'b-o','markersize',10,'markerfacecolor','b');
>> axis([-1,5,-1,4]);
>> xticks([0,2,4]);
>> yticks([0,2,3]);
>> grid on;
>> xlabel('x');
>> ylabel('y');
>> title('Exemplu grafic');
```

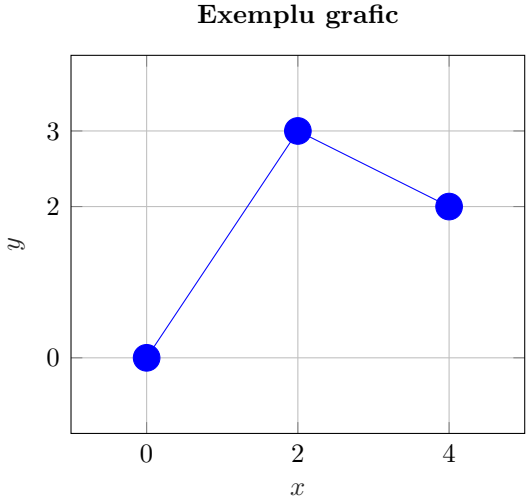


Figura B.7. *Plot complet*

Funcția `plot` este capabilă să afișeze și mai multe grafice pe aceeași figură. Este posibil afișarea pe aceleași axe a mai multor seturi de coordonate diferite, adăugând fiecare sub forma unei perechi de vectori x și y, și opțional formatul dorit, ca parametri adiționali la funcția `plot`.

Astfel dacă vrem să afișăm funcțiile $sin(x)$ și $cos(x)$ în intervalul $(0, 2\pi)$ pe aceeași grafic, va trebui să construim două perechi de vectori x,y corespunzător celor două funcții. Cum graficele din Matlab nu pot reprezenta direct curbe ci doar o aproximare prin segmente de linie, vectorul pentru coordonatele de pe abscisă trebuie să aibă o rezoluție suficient de mare pentru ca aproximarea să aibă o eroare maximă acceptabilă. În cazul afișării pe ecran putem considera eroarea acceptabilă dacă este sub dimensiunea unui pixel. De exemplu, dacă vrem să afișăm intervalul $(0, 2\pi)$ pe un ecran cu rezoluția de 1920x1080 pixel, atunci ar trebui să considerăm un pas de cel mult $\frac{2\pi}{1920} \approx 0.003272$. Următorul exemplu de cod afișează câte o perioadă completă de sinus și cosinus suprapuse unul peste celălalt:

```
>> x = 0:pi/1000:2*pi;
>> s = sin(x);
>> c = cos(x);
>> plot(x,s,';sin(x);',x,c,';cos(x);');
>> xlabel('x'); ylabel('F(x)');
>> xticks([0, pi, 2*pi]); yticks([-1,0,1]);
>> axis([0,2*pi,-1,1]);
>> xticklabels({'0','\pi','2\pi'});
```

Graficul rezultat se vede pe figura B.8.

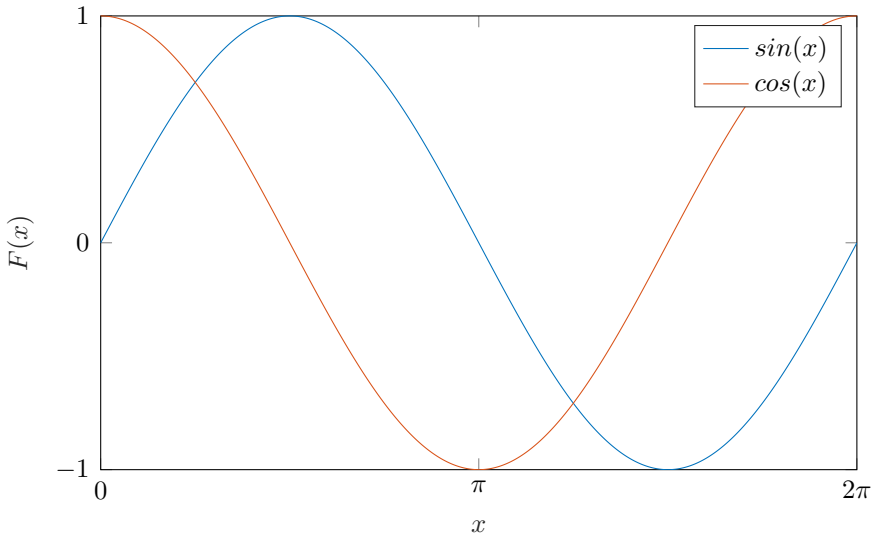


Figura B.8. *Două grafice suprapuse*

S-a folosit de sintaxa `';nume;'` pentru a asocia și o legendă care să identifice care linie corespunde cu care funcție (în cazul nostru albastru corespunde cu sinus și roșu cu

cosinus). Legenda poate fi specificată și cu comanda `legend` care permite și configurări mai avansate (poziție, dimensiune etc). De asemenea, datorită faptului că Matlab preferă marcajele pe axe să fie valori rotunde, am specificat manual că vrem să vedem π și 2π pe axe. Pentru aceasta trebuie specificat atât valorile cu `xticks` cât și textul asociat cu `xticklabels`, altfel am avea doar valorile aproximative 3.1416 și 6.2832.

Pentru a afișa mai multe grafice pe axe separate se folosește comanda `subplot(r,c,p)`. Aceasta va împărți figura completă într-o serie de grafice aranjate într-o grilă cu r rânduri și c coloane și selectează poziția p (numerotarea fiind de la stânga la dreapta și de sus în jos). Aranjarea pe grilă cu 3 rânduri și 2 coloane, respectiv pozițiile aferente, este prezentată pe figura B.9.

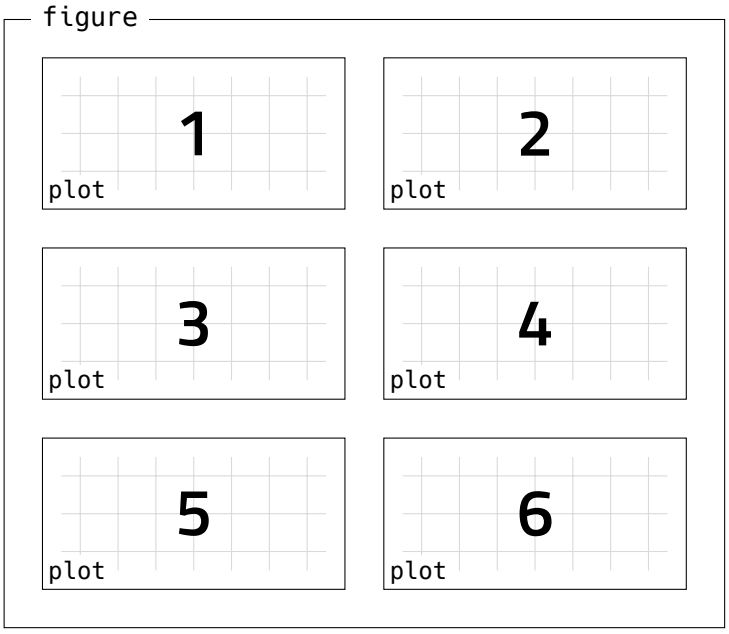


Figura B.9. *Aranjarea sub-graficelor*

Cât timp fereastra rămâne pe ecran, apelul la `plot` va desena graficul în celula selectată ultima dată.

De exemplu, dacă vrem să afișăm sinusoida și cosinusoida din exemplul precedent pe două grafice aranjate unul sub celălalt, putem apela următoarea secvență de cod:

```
>> x = 0:pi/1000:2*pi;
>> s = sin(x);
>> c = cos(x);
>> subplot(2,1,1), plot(x, s);
>> xlabel('x'); ylabel('sin(x)');
>> xticks([0, pi, 2*pi]); yticks([-1,0,1]);
>> axis([0,2*pi,-1,1]);
>> xticklabels({'0','\pi','2\pi'});
```

```
>> subplot(2,1,2), plot(x, c);
>> xlabel('x'); ylabel('cos(x)');
>> xticks([0, pi, 2*pi]); yticks([-1,0,1]);
>> xticklabels({'0','\pi','2\pi'});
>> axis([0,2*pi,-1,1]);
```

Rezultatul codului este prezentat pe figura B.10. Trebuie notat faptul că fiecare `plot` trebuie configurat separat, chiar dacă vrem să obținem setări identice.

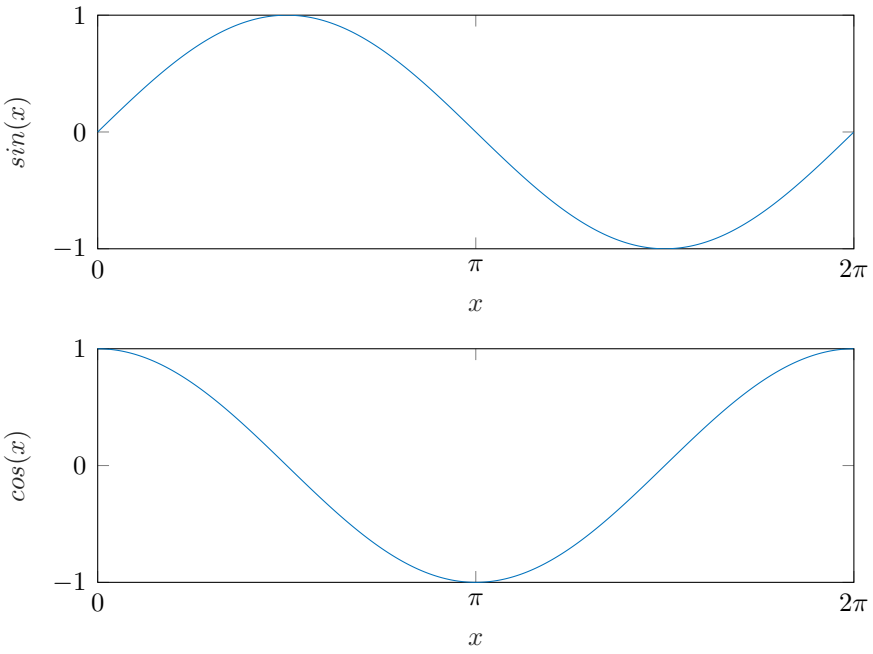


Figura B.10. *Subplot cu două grafice*

Anexa C

Diferențe între MATLAB și Octave

Cele două suite de interpretor Matlab sunt aproape compatibile, în sensul că scripturile scrise într-una din soluții vor rula și în cealaltă fără modificare, sau cu foarte mici modificări. Există totuși câteva mici diferențe, de care trebuie ținut cont la portarea unui script dintr-un mediu în altul. Exemplele de cod din acest îndrumar, în mare parte, au fost testate cu Octave, dacă vreunul nu funcționează în MATLAB atunci trebuie aplicată una din modificările detaliate în continuare.

Blocuri de control

În MATLAB blocurile de cod începute cu **function**, **if**, **for**, **while** se termină fiecare cu **end**. Octave recunoaște **end** ca terminator pentru aceste blocuri, dar permite și existența unor terminatori dedicați **endfunction**, **endif**, **endfor**, **endwhile**, pentru a oferi o claritate sporită a codului în cazul unor structuri încorporate.

Pentru claritatea sporită și în acest îndrumar unele funcții folosesc notația din Octave. Pentru a rula acestea în MATLAB, ele trebuie modificate prin înlocuirea terminatorului de bloc cu **end**.

De exemplu secvența de cod:

dit.m

```
while N/2 < lenx
  for k = 0:N/2-1
    w = wn(k * lenx / N+1);
    for n = k+1:N:lenx
      a = X(n); b = w*X(n+N/2);
      X(n) = a + b; X(n+N/2) = a - b;
    endfor
  endfor
  N = N*2;
endwhile
```

va trebui rescrisă în:

dit.m

```
while N/2 < lenx
  for k = 0:N/2-1
    w = wn(k * lenx / N+1);
    for n = k+1:N:lenx
      a = X(n); b = w*X(n+N/2);
      X(n) = a + b; X(n+N/2) = a - b;
    end
  end
  N = N*2;
end
```

pentru a putea rula în MATLAB.

Operatori compacte

Octave permite folosirea operatorilor compacți de asignare (`+=`, `-=`, `*=`, `/=`, `^=` etc.) și cele de incrementare/decrementare (`++`, `--`) similar cu limbajul C++. MATLAB nu recunoaște acești operatori.

În Octave pot fi scrise liniile de cod:

```
>> i++;
>> a += 2;
>> c *= 5;
```

În MATLAB aceste linii de cod trebuie scrise explicit:

```
>> i = i + 1;
>> a = a + 2;
>> c = c * 5;
```

Indexare compactă

Octave permite folosirea operatorului de indexare peste orice obiect ce se evaluează într-o matrice, spre deosebire de MATLAB, care permite indexarea numai pe variabile. Astfel, codul scris în Octave poate fi mai compact, rezultatul unei operații putând fi indexat direct, nu trebuie salvat într-o variabilă temporară.

De exemplu, în Octave putem obține spectrul real al unui semnal într-o singură linie de cod:

```
>> X = abs(fft(x))(1:length(x)/2);
```

Aceasta în MATLAB trebuie despărțit în două linii de cod:

```
>> X = abs(fft(x));
>> X = X(1:length(x)/2);
```

Comentarii

MATLAB folosește semnul `%` pentru a indica comentariile. Tot ce este după acest semn până la sfârșitul liniei este ignorat de interpretor. Octave tratează semnul `%` la fel, dar pe lângă aceasta și semnul `#` poate fi folosit pentru a introduce comentarii (similar cu alte limbaje scriptice obișnuite, ca Python, Perl, bash etc.).

Acest lucru permite ca scripturile Octave să fie integrate cu interpretorul de comenzi din Linux (bash) prin posibilitatea rulării scripturilor care încep cu un shebang `#!/usr/bin/octave` direct din linia de comandă, fără a fi nevoie de pornirea interpretorului.

De exemplu următorul fișier:

lcm

```
#!/usr/bin/octave -qf
args = str2double(argv());
if length(args) != 2 || any(isnan(args))
    error ("Expected 2 numeric arguments");
endif

disp(lcm(args(1),args(2)));
```

poate fi executat direct din linia de comandă (bash), ca orice alt executabil (presupunând că flagul de execuție a fost setat), sub forma:

```
> ./lcm 44100 48000
```

Acest fișier însă eșuează dacă este rulat în MATLAB.

Șiruri de caractere

Matlab folosește semnul `'` pentru a delimita șiruri de caractere și semnul `"` pentru a crea vectori de șiruri de caractere. Octave nu recunoaște aceasta din urmă și folosește atât `'` cât și `"` pentru delimitarea șirurilor de caractere, singura diferență fiind la interpretarea codurilor speciale (ca în Perl).

Astfel secvența:

```
>> a = 'abc';
>> b = "def";
>> [a b]
```

va rezulta într-un șir de caractere `'abcdef'` în Octave și într-un vector cu două elemente de tip șiruri de caractere `{"abc" "def"}` în Matlab.

Altele

Există și o multitudine de alte diferențe subtile, care sunt mai rar întâlnite, dar care pot cauza comportament neașteptat la trecerea dintr-un mediu în celălalt. Lista completă a diferențelor și a implicațiilor acestora este descrisă în wiki-ul Octave: `https://wiki.octave.org/Differences_between_Octave_and_Matlab`

Bibliografie

[1] Coliban R.M., Ivanovici L.M., *Prelucrarea semnalelor: îndrumar de laborator*, Editura Universității Transilvania, Brașov, 2018

[2] Gâlmeanu H., *Bazele procesării și transmiterii semnalelor II – Îndrumar de laborator*, Editura Universității Transilvania, Brașov, 2002

[3] Kalechman M., *Practical Matlab Applications for Engineers*, CRC Press, Boca Raton, FL, 2008

[4] Kalechman M., *Practical Matlab Basics for Engineers*, CRC Press, Boca Raton, FL, 2008

[5] Kertész Cs.Z., Ivanovici L.M., *Procesarea digitală a semnalelor – Îndrumar de laborator*, Editura Universității Transilvania, Brașov, 2009

[6] Mitra S.K., *Digital Signal Processing: A Computer-Based Approach*, ed. 2., McGraw-Hill, New York, 2001

[7] Smith J.O., *Introduction to Matlab and Octave*, Center for Computer Research in Music and Acoustics, Stanford University, Stanford, CA, februarie 2009

[8] Smith S.W., *The Scientist and Engineer's Guide to Digital Signal Processing*, ed. 2., California Technical Publishing, San Diego, 1999, URL `http://www.dspguide.com/pdfbook.htm`