MATLAB
Un prim pas spre cercetare

Cătălina Neghină
Alina Sultana
Mihai Neghină

Descrierea CIP a Bibliotecii Naţionale a României
NEGHINĂ, CĂTĂLINA
MATLAB : un prim pas spre cercetare / Cătălina Neghină, Alina Sultana, Mihai Neghină. - Sibiu : Editura Universităţii "Lucian Blaga" din Sibiu, 2016
Conţine bibliografie
ISBN 978-606-12-1213-2

I. Sultana, Alina
II. Neghină, Mihai

004.42 MATLAB

Copertă: Anna Enache

Cuvânt înainte

Această carte se dorește a fi un ghid util celor care vor să învețe să lucreze în MATLAB, un limbaj folosit intens în cercetare și în inginerie, ce permite implementarea cu ușurință a algoritmilor din diverse domenii precum: procesarea imaginilor, procesarea semnalelor audio, inteligență artificială, controlul sistemelor, statistică, finanțe etc.

Prin descrierile și exemplele prezentate, această carte reprezintă o introducere în utilizarea versiunii de MATLAB R2015a, fiind în același timp o deschidere de perspectivă despre ce se poate realiza în mediul de simulare MATLAB și o sistematizare a modului de a gândi matriceal, specific MATLAB-ului.

Cartea prezintă informațiile gradual, de la noțiuni de bază precum mediul de lucru, variabile, instrucțiuni etc, la lucrul cu matrice, interfață grafică, calcul parametric, ca în final să se ajungă la funcții utilizate în procesarea semnalelor (semnale audio, imagini, imagini medicale). Funcțiile prezentate sunt însoțite de exemple pentru o mai bună înțelegere. În cazul funcțiilor specifice procesării imaginilor, există un întreg capitol cu aplicații pentru imagistica medicală. Majoritatea funcțiilor descrise sunt disponibile și în versiunile anterioare ale MATLAB-ului.

Atașat acestei cărți se găsește un CD care include toate fișierele externe (imagini, fișiere audio, fișiere **.mat*) utilizate în aplicații. În acest mod, cititorii interesați pot testa rapid programele propuse.

Doresc pe această cale să mulțumesc Domnului Prof. Dr. Ing. MIHU P. Ioan pentru tot sprijinul acordat în activitatea mea didactică din cadrul *Facultății de Inginerie*, Universitatea *Lucian Blaga* din Sibiu, pentru încurajarea de a scrie această carte precum și pentru observațiile privitoare la conținutul cărții.

De asemenea, doresc să exprim sincere mulțumiri Doamnei Inginer Marina STOICA pentru observațiile și sugestiile făcute în legătură cu conținutul cărții precum și pentru testarea programelor incluse în această carte.

Cătălina NEGHINĂ

Cuprins

1. MATLAB. Vedere de ansamblu

Limbajul MATLAB este un limbaj de nivel înalt, folosit intens în cercetare și în inginerie, ce permite implementarea cu ușurință a algoritmilor din diverse domenii precum: procesarea imaginilor, procesarea semnalelor audio, inteligență artificială, controlul sistemelor, statistică, finanțe etc.

De ce să folosim limbajului MATLAB:

- MATLAB-ul este optimizat pentru calcul matriceal, operațiile cu matrice (așa cum se va vedea) fiind foarte ușor de realizat. Elementul de bază cu care lucrează MATLAB-ul este **matricea**, acest lucru sugerându-l chiar numele de MATLAB care vine de la **"matrix laboratory"**.
- Se poate lucra cu aproape orice tip de semnal și fișier; se pot importa date din Excel, se pot citi și salva imagini, se pot încărca și reda semnale audio și semnale video etc
- Reprezentarea grafică a funcțiilor este extrem de flexibilă: se pot face reprezentări 2D, 3D, histograme, reprezentări procentuale, reprezentări vectoriale etc.
- Este posibilă generarea unui raport automat (în format `html`, `pdf`, `Word`, `LaTeX`, `xml`) ce include codul MATLAB, comentariile și rezultatele obținute (inclusiv graficele).
- Se pot scrie programe în mod mult mai rapid decât în alte limbaje de programare deoarece nu este absolut necesar să se mai realizeze operații precum declararea variabilelor și specificarea tipului de date.
- În MATLAB, de multe ori poate fi evitată folosirea buclelor `for` (inevitabile în alte limbaje) datorită optimizării MATLAB-ului pentru lucrul cu matrice. Din acest motiv, codul scris în MATLAB este adesea mai compact, mai ușor de urmărit și mai puțin „stufos" decât codul scris în alte limbaje de programare.
- *Help*-ul este foarte bine dezvoltat, majoritatea funcțiilor având și exemple.
- Din MATLAB pot fi apelate direct funcții scrise în C, C++, Java și .NET. De asemenea, funcțiile MATLAB pot fi apelate din aplicații realizate în C sau C++.
- În MATLAB se pot realiza ușor interfețe grafice (GUI).
- MATLAB-ul dispune de o serie de biblioteci de funcții (*toolboxes*) foarte utile pentru simulările din domeniul inteligenței artificiale, procesarea imaginilor, procesarea semnalelor etc.

Această carte a fost scrisă pentru a exemplifica utilizarea versiunii de MATLAB R2015a. Majoritatea funcțiilor descrise aici sunt însă disponibile și în versiunile anterioare ala MATLAB-ului.

Pentru versiunea de MATLAB R2015a există o varietate de instrumente pentru dezvoltarea eficientă a algoritmilor:

1) Fereastra *Command Window* permite executarea instrucțiunilor linie cu linie.

2) Fereastra *Workspace* pemite vizualizarea tuturor variabilelor utilizate și stocate în memorie în timpul unei sesiuni MATLAB.

3) Fereastra *Command History* permite vizualizarea ultimelor instrucțiuni scrise în *Command Window*. Se poate accesa o instrucțiune din *Command History* prin dublu-click pe acea instrucțiune. Apăsând tasta ↑ (săgeată în sus) în fereastra *Command Window*, se vor afișa linie cu linie instrucțiunile din *Command History* în ordinea inversă în care au fost scrise.

4) *Editorul MATLAB*, în care se pot scrie fișiere script și funcții. Aici există posibilitatea de a realiza *debug* pentru codul scris.

Dacă se dorește pornirea aplicației în modul *Default*, se accesează din *Toolstrip* opțiunea *Home* și apoi *Layout* varianta *Default*.

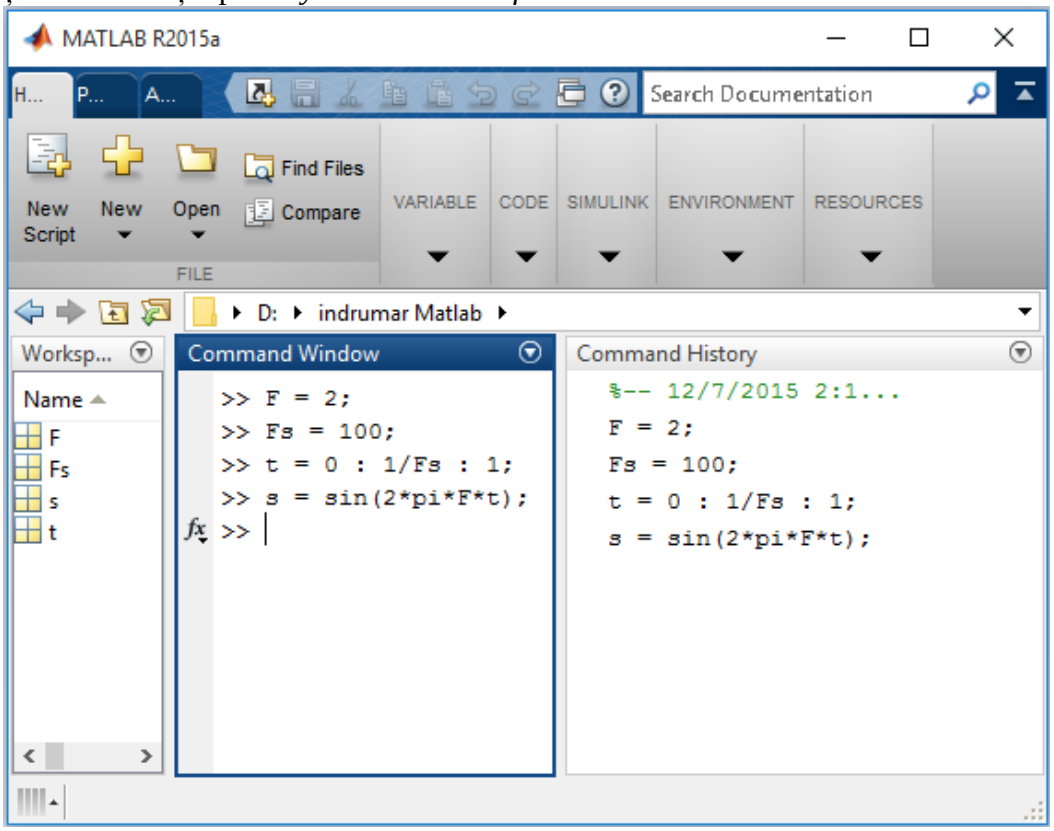


Figura 1.1. Interfață MATLAB versiunea (R2015a)

Fereastra *Command Window*

Fereastra *Command Window* permite executarea instrucțiunilor linie cu linie. Instrucțiunea se execută doar la apăsarea tastei ENTER. În această fereastră este recomandat să se scrie instrucțiuni scurte, să se vizualizeze valorile variabilelor, să se efectueze calcule etc. Însă dacă se dorește să se scrie un program sau mai mult de 3-4 linii de cod, atunci este indicat să se folosească *Editorul* MATLAB. Dacă se dorește să se scrie în *Command Window* mai multe linii cu instrucțiuni iar execuția să se realizeze abia după ultima instrucțiune, în acest caz se va apăsa SHIFT + ENTER după fiecare linie.

- Dacă se dorește să se calculeze $(\frac{162}{54}+2)\cdot 2$, se va scrie în *Command Window*:

```
>> (162/54+2)*2
   ans =
       10
```

Observații:

- atunci când utilizatorul nu specifică salvarea rezultatului într-o variabilă, rezultatul se va salva automat în variabila `ans`.
- dacă nu se dorește afișarea în *Command Window* a rezultatului unei instrucțiuni, atunci se va termina acea instrucțiune cu *punct și virgulă*(;). Spre deosebire de alte limbaje de programare (de exemplu C++), operatorul punct și virgulă nu este obligatoriu în MATLAB la sfârșitul unei instrucțiuni; prezența lui suprimă doar afișarea rezultatelor în Command Window
- Pentru a șterge toate liniile din *Command Window* se folosește comanda *clc.*

- Fie `a = 5` și `b = 205`. Să se calculeze $c = \frac{b-a}{2}$.

```
>> a = 5;
>> b = 205;
>> c = (b-a)/2
c =
      100
```

- Să se genereze o matrice de 3 x 3 cu următoarea proprietate: suma pe linii să fie egală cu suma pe coloane și cu suma pe diagonale. *Observație*: o astfel de matrice formează un pătrat magic care se poate genera în MATLAB cu funcția `magic`.

```
>> M = magic(3)
M =
     8     1     6
     3     5     7
     4     9     2
```

Fereastra *Workspace*

Fereastra *Workspace* permite vizualizarea tuturor variabilelor utilizate și stocate în memorie în timpul unei sesiuni MATLAB (pentru a vedea variabilele folosite într-o funcție trebuie ca MATLAB-ul să fie în modul *Debug*).
Pentru fiecare variabilă există mai multe proprietăți ce pot fi vizualizate:

- `Value`: afișează valorile variabilei în cazul în care variabila este o matrice de mici dimensiuni; dacă variabila este o matrice de mari dimensiuni atunci se afișează dimensiunea matricei.
- `Size`: dimensiunea matricei
- `Bytes`: numărul de octeți ocupați de variabilă

Mai sunt de asemenea și alte proprietăți precum: `class`, `min`, `max`, `range` etc. Pentru a include o proprietate în *Workspace* se dă click-dreapta pe bara de sub *Workspace*, acolo unde apare scris `Name`.

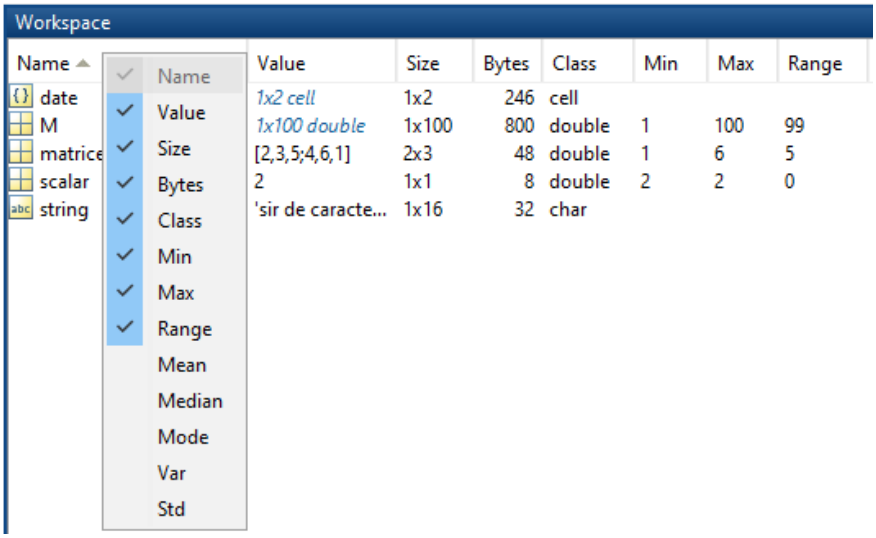


Figura 1.2. Fereastra *Workspace* din MATLAB

Observații:

- Dacă se dorește ștergerea unei variabile din *Workspace* se va scrie în *Command Window* sau în Editorul MATLAB comanda `clear nume_variabilă`. De exemplu, dacă se dorește ștergerea matricei `M` din exemplul de mai sus, se va scrie `clear M`.
- Pentru a șterge toate variabilele din *Workspace* se folosește comanda `clear all`.

Pentru a vizualiza conținutul unei variabile se poate da dublu-click pe numele variabilei din *Workspace*. De asemenea se poate tasta numele variabilei în *Command Window*, fără a pune punct și virgulă la sfârșit.

Editorul MATLAB

Pentru a edita un fișier MATLAB se selectează *Home* și apoi *New Script*. De asemenea se poate da comanda `edit` în *Command Window*. Numele sub care se salvează un fișier trebuie să respecte următoarea regulă: să înceapă cu literă și să conțină numai litere, cifre sau *underline* (_). Un fișier scris în *Editorul* MATLAB este un fișier *m*-file (se salvează cu extensia **.m*).
Atenție: numele unui fișier nu poate conține spațiu.

Pentru a rula un program, se poate folosi tasta **F5** sau se poate alege opțiunea *Editor* și apoi *Run* (săgeata verde). Este bine ca primele linii ale programului să fie:

```
clc
clear all
close all
```

deoarece la rularea programului:

- `clc` șterge tot ce era în *Command Window*. În acest fel ne asigurăm că orice rezultat sau eroare afișată în *Command Window* se referă la rularea curentă.
- `clear all` șterge toate variabilele din *Workspace*.
- `close all` închide toate ferestrele cu grafice.

Comentarii

Pentru a comenta o linie de cod se folosește semnul procent (`%`). Linia comentată va fi scrisă cu verde.

```
>> % m = 1 : 5; variabila m nu exista deoarece este comentat
>> n = 1 : 5; % n contine numerele de la 1 la 5
```

Observații:

- Pentru a comenta mai multe linii de cod în același timp, se pot selecta liniile și folosi combinația de taste CTRL + R.
- Pentru a decomenta mai multe linii de cod, se pot selecta liniile și folosi combinația de taste CTRL + T.

Pentru a separa fișierul script în secțiuni se folosește semnul `%%`

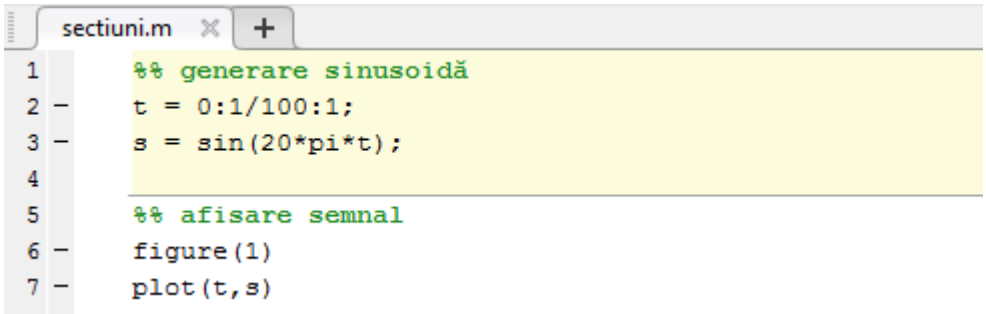


Figura 1.3. Împărțirea codului pe secțiuni

Corectarea erorilor (Debug)

Primul pas înainte de a face *debug* este de a salva programul. Pentru a vedea valori intermediare, oprirea temporară a rulării poate fi făcută cu ajutorul *punctelor de întrerupere* (*breakpoints*). Adăugarea de breakpoints se poate face prin click pe liniuța (-) din dreptul liniei unde se dorește oprirea temporară a programului; linia 8 în exemplul de mai jos.

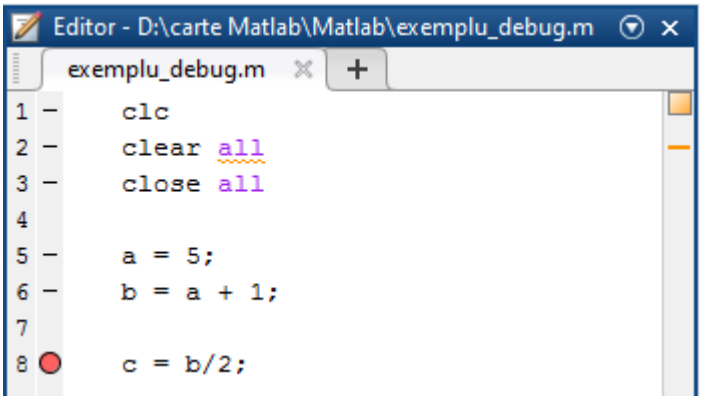


Figura 1.4. Inserarea unui breakpoint

Oprirea se va face înainte de rularea comenzii de pe linia cu breakpoint; în exemplul de mai sus, înainte de înjumătățirea lui `b`. In timp ce programul este oprit într-un breakpoint, în *Command Window* apare indicația `K>>`, care informează utilizatorul că MATLAB-ul poate accepta o valoare de la tastatură (keyboard).

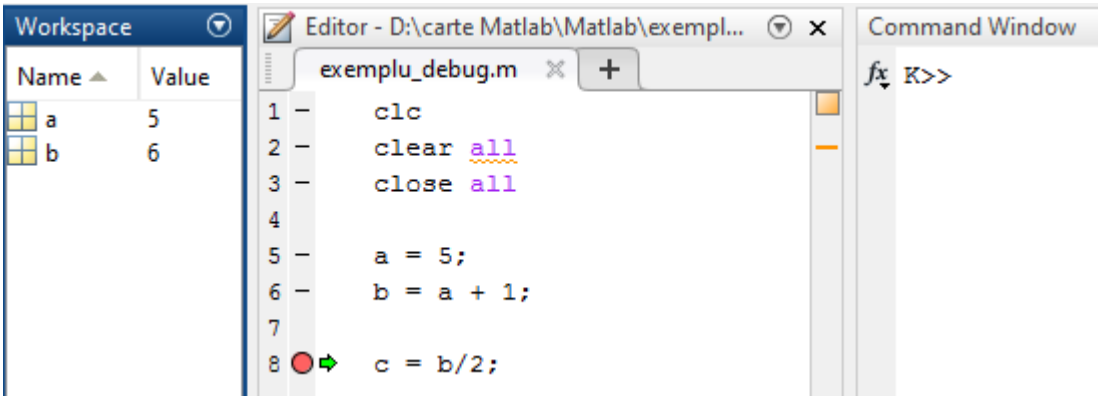


Figura 1.5. Rezultatul rulării unui program în care s-a inserat un breakpoint

Variabilele din *Workspace* sunt accesibile și modificabile în timpul rulării, și chiar pot fi introduse variabile noi, de care rularea ulterioară a programului va ține cont.

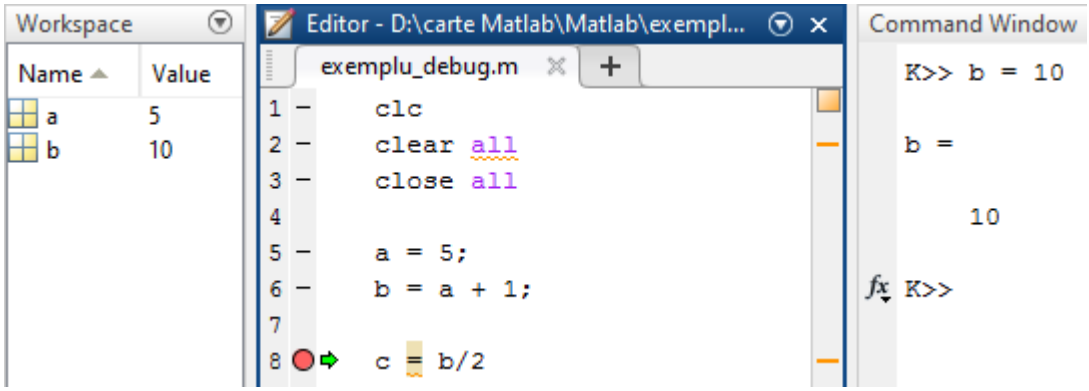


Figura 1.6. Modificarea unei variabile în timpul rulării programului

Pentru a ieși din breakpoint, este suficient un click pe butonul *Continue* 

. În acest caz, programul va continua cu valoarea lui `b = 10` iar rezultatul afișat în *Command Window* va fi `c = 5`.

Butonul *Quit Debugging* 

oprește complet rularea programului, însă păstrează în continuare valorile actuale (intermediare) ale variabilelor în *Workspace*.

Punctele de întrerupere pot fi adăugate și din cod, cu comanda `keyboard`. Efectul este același ca un breakpoint adăugat cu mouse-ul. Un mod alternativ de a continua un program oprit cu breakpoint este comanda `return` scrisă în *Command Window*.

- În exemplul de mai jos, atunci când se ajunge la linia de cod 8, se va realiza o oprire temporară a programului iar în *Command Window* o sa apară `K>>`. Dacă se va scrie `K>> return`, se va continua rularea programului și se va afișa în *Command Window* `c = 3`.

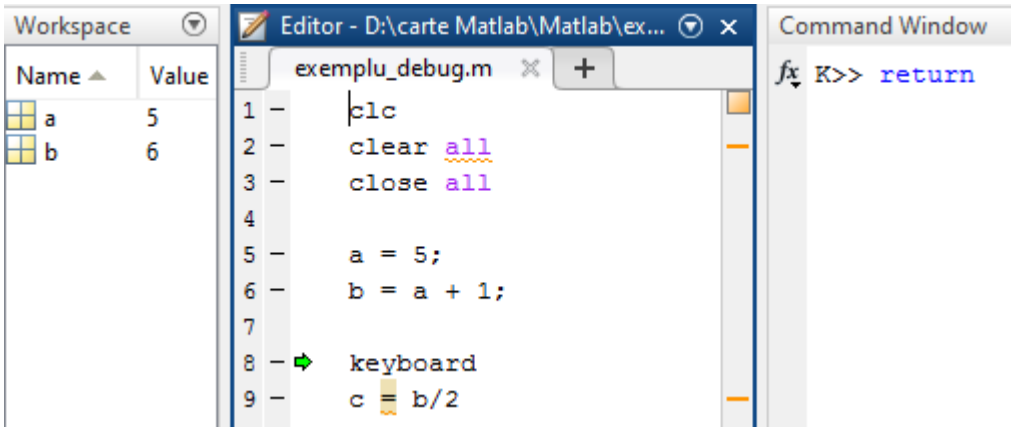


Figura 1.7. Exemplu de utilizare a comenzilor `keyboard` și `return`

Comanda `pause`

Comanda `pause(t)`, unde `t` este un număr pozitiv, oprește temporar rularea programului pentru `t` secunde. Comanda `pause` fără parametri oprește temporar rularea programului până când utilizatorul apasă o tastă (orice tastă). Această comandă poate fi utilă de exemplu atunci când programul suprascrie o imagine de mai multe ori, iar utilizatorul este interesat să aibă timp să vadă imaginile intermediare.

- Să se afișeze la interval de 0.5s, toate imaginile **.jpg* dintr-un folder (pentru a înțelege mai bine codul de mai jos, a se citi capitolul *3.4. Citirea tuturor fișierelor dintr-un folder*)

```
path = 'D:\carte Matlab\poze';
folder = dir(path);
len_folder = size(folder,1);
for index = 1:len_folder
    fisier = folder(index).name;
    len_fisier = length(fisier);
    if (~folder(index).isdir()) &&...
        strcmpi(fisier(len_fisier-3:len_fisier),'.jpg')
        nume_poza = [path,'\',fisier];
        imagine = imread(nume_poza);
        figure(1)
            imshow(imagine)
            pause(0.5)
    end
end
```

Comenzile `tic-toc`

Combinația de comenzi `tic-toc` permite măsurarea timpului de rulare de când este întâlnită comanda `tic` până când este întâlnită comada `toc`. Dacă se folosește sintaxa `start = tic`, în variabila `start` se va salva momentul de timp la care a fost rulată această linie de cod. Comanda `toc(start)` va calcula intervalul de timp necesar execuției codului cuprins între `start = tic` și `toc(start)`.

- Să se calculeze timpul necesar generării unui vector cu 10000 de elemente, de la 1 la 10000, fără a folosi buclă `for`. Apoi să se calculeze timpul necesar generării aceluiași vector folosind însă o buclă `for`.

```
tic;
x = 1 : 10000;
toc
start = tic;
for i = 1 : 10000
    y(i) = i;
end
toc(start)
```

În urma rulării programului anterior, în *Command Window* se va afișa:

```
Elapsed time is 0.000259 seconds.
Elapsed time is 0.004693 seconds.
```

Comanda `help`

Comanda `help nume` oferă informații despre parametrul `nume` care poate fi funcție, metodă, instrucțiune etc.

Sintaxă: `help nume`

- Să se afle informații despre funcția `abs`.

```
>> help abs
 abs    Absolute value.
    abs(X) is the absolute value of the elements of X. When
    X is complex, abs(X) is the complex modulus (magnitude) of
    the elements of X.

    See also sign, angle, unwrap, hypot.

Other functions named abs

    Reference page in Help browser
doc abs
```

Pentru mai multe informații se accesează documentul din link-ul afișat la final, în cazul de față doc abs.

Comanda `lookfor`

Dacă doriți să utilizați o funcție despre care știți că este implementată în MATLAB, dar nu îi știți numele, o puteți căuta după un cuvânt cheie precedat de `lookfor`.

Sintaxă: `lookfor cuvânt_cheie`

- Doriți să calculați media elementelor dintr-un vector dar nu știți ce funcție să folosiți. Un cuvânt cheie după care puteți căuta este *average*:

```
>> lookfor average
localavfit                  - Construct "average fit" model
mean                        - Average or mean value.
```

MATLAB-ul va începe să afișeze toate funcțiile care pot avea legătură cu acest cuvânt cheie căutat. În acest caz observăm că a doua funcție găsită este `mean`, a cărei descriere sugerează că este funcția căutată. Pentru a opri rularea unei aplicații se poate folosi combinația de taste CTRL + C atunci când fereastra *Command Window* este selectată.

Afișarea valorilor variabilelor în *Command Window*

Pentru a afișa valoarea unei variabile în fereastra *Command Window* se poate folosi comanda `disp`.

Sintaxă: `disp(A)`

Comanda `disp(A)` afișează în *Command Window* valoarea variabilei `A`, fără a afișa însă și numele variabilei.

```
>> A = 5;
>> B = [1 2 3];
>> C = 'afisare mesaj';

>> disp(A)
     5
>> disp(B)
     1     2     3
>> disp(C)
afisare mesaj
```

O altă variantă de a afișa valoarea unei variabile în *Command Window* este de a scrie numele variabilei. Spre deosebire de folosirea comenzii `disp`, această modalitate afișează și numele variabilei urmat de semnul egal.

```
>> A = 5;
>> A
A =
     5

>> B = [1 2 3; 4 5 6]
B =
     1     2     3
     4     5     6
```

Observație: în MATLAB, dacă o linie de cod nu se încheie cu punct și virgulă, în *Command Window* se va afișa valoarea variabilei inițializată în acea linie de cod.

```
>> A = [1 2 3 4 5];
>> B = [2 4 6]
B =
     2     4     6
```

Se observă că spre deosebire de variabila `A`, variabila `B` nu a mai fost afișată în *Command Window*. Ambele variabile se găsesc însă în *Workspace*.

2.1. Variabile

În MATLAB, variabilele sunt nume simbolice pentru valori de tip numeric, șir de caractere, structuri etc. Spre deosebire de alte limbaje, în MATLAB nu este necesară definirea tipului variabilelor sau a dimensiunii, acestea fiind deduse din context la inițializare. Numele unei variabile trebuie să respecte următoarele reguli:

- să înceapă obligatoriu cu o literă.
- să conțină doar litere, cifre sau "_".
- să fie diferit de cuvintele cheie ale MATLAB-ului (precum `if`, `for`, `function` etc). Pentru lista completă a cuvintelor cheie din MATLAB rulați comanda `iskeyword`.

- Să se inițializeze variabila `algoritm` cu valoarea `'K-means'` și variabila `numar_iteratii` cu valoarea `10`.

```
algoritm = 'K-means';
nr_iteratii = 10;
```

După rularea programului ce conține liniile de mai sus, se poate observa că în *Workspace* au apărut variabilele `algoritm` de tip *char* și `numar_iteratii` de tip *double*. Despre aceste tipuri de date se va discuta mai pe larg în subcapitolul următor.

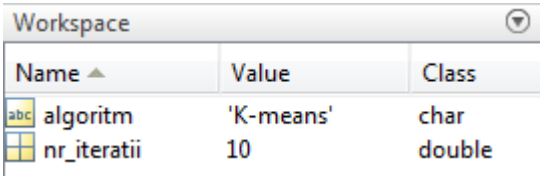


Figura 2.1. Inițializare variabile

MATLAB-ul este *case sensitive*, adică face distincție între litere mici și litere mari.

- Fie variabila `a` = 2. Să se calculeze `b` = `a` + 3 și apoi `c` = `A` + 3.

```
a = 2;
b = a + 3
```

În *Command Window* se va afișa:

```
b =
     5
```

Dacă se va scrie însă:

```
a = 2;
c = A + 3
```

Se va afișa un mesaj de eroare deoarece nu există variabila `A`.

```
Undefined function or variable 'A'.
```

Este indicat să se evite ca numele variabilei să fie același cu cel al unei funcții existente în MATLAB (de exemplu `max`, `sum`, `mean` etc). În general, numele de variabilă are prioritate față de numele funcției.

- Fie vectorul X având valorile [1, 2, 6, 3]. Să se calculeze valoarea maximă a vectorului X folosind funcția `max`.

```
X = [1 2 6 3];
max(X)
```

Rezultatul va fi 6. Să se adauge acum în cod o variabilă numită `max`.

```
max = 3;
X = [1 2 6 3];
max(X)
```

În *Command Window* se va afișa un mesaj de eroare deoarece în loc să se folosească funcția `max` de determinare a maximului, MATLAB-ul încearcă să folosească variabila `max`. Mesajul de eroare este în acest caz:

```
Index exceeds matrix dimensions.
```

Pentru a afla numărul maxim de caractere din denumirea unei variabile se folosește funcția `namelengthmax`. Această funcție poate fi folosită pentru aflarea numărului de caractere ale oricărui identificator (variabilă, nume de funcție, nume fișier *m*-file etc).

```
>> namelengthmax
ans =
    63
```

Pentru a șterge toate variabilele din *Workspace* se folosește comanda `clear all`. Pentru a șterge doar o anumită variabilă din *Workspace* se folosește comanda `clear nume_variabila`.

Variabile predefinite

În MATLAB există o serie de variabile predefinite dintre care amintim: `i` și `j` (unitatea imaginară), `pi`, `eps`, `Inf`, `NaN`. Valorile acestor variabile pot fi modificate în timpul programului, deși acest lucru nu este recomandat.

- **unitatea imaginară** notată în MATLAB cu `i` sau `j`. Se folosește pentru generarea numerelor complexe și are proprietatea că $i^2 = -1$. În MATLAB, pentru a defini un număr complex se poate scrie în mai multe moduri: $z = a + bi$, $z = a + bj$, $z = a + b * i$ sau $z = a + b * j$.

Să se genereze numărul complex $z = 2 + 3i$. Să se calculeze i^2.

```
>> z = 2 + 3*i  % alternativ z = 2 + 3i
z =
   2.0000 + 3.0000i

>> i^2
ans =
    -1
```

Observație: în cazul în care se suprascrie variabila `i`, aceasta nu mai este recunoscută ca unitate imaginară dacă se folosește sintaxa $z = a + b * i$.

Să se inițializeze variabila `i` cu valoarea 2. Să se calculeze $z = 2 + 3i$.

```
>> i = 2;
>> z = 2 + 3*i
z =
     8

>> z = 2 + 3i
z =
   2.0000 + 3.0000i
```

- **litera grecească π**, notată în MATLAB cu `pi`.

Să se afișeze π cu 4 zecimale și apoi cu 15 zecimale. Să se calculeze $\sin(\pi/6)$.

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.141592653589793
>> sin(pi/6)
ans =
    0.5000
```

- **`NaN`** provine de la Not-a-Number și se obține în urma unei operații matematice nedefinite precum 0/0, Inf – Inf etc.

```
>> 0/0
ans =
   NaN

>> Inf - Inf
ans =
   NaN
```

- **variabila `eps`,** reprezintă cel mai mic număr din MATLAB astfel încât `1 < 1 + eps`. Se folosește destul de des atunci când dorim ca o variabilă să aibă o valoare cât mai mică și diferită de zero. De asemenea se folosește la însumarea cu numitorul unei fracții pentru a evita împărțirea la zero.

```
>> eps
ans =
  2.2204e-016
```

Obs: `e-016` $= 10^{-16}$

- Să se calculeze $sinc(x) = \frac{\sin(x)}{x}$ pentru $x \in \{-1, -0.5,\ 0,\ 0.5,\ 1\}$.

Atunci când $x = 0$, evaluarea expresiei în MATLAB devine nedeterminată (0/0).

```
x = [-1:0.5:1];
semnal_1 = sin(x)./x
semnal_1 =
    0.8415    0.9589       NaN    0.9589    0.841
```

Pentru a evita această nedeterminare, se adună `eps` la vectorul `x`.

```
y = x + eps;
semnal_2 = sin(y)./(y)
semnal_2 =
    0.8415    0.9589    1.0000    0.9589    0.8415
```

- **`Inf`,** adică infinit. Se obține prin împărțirea la zero. Se folosește de obicei atunci când dorim ca o variabilă să aibă cea mai mare valoare.

```
>> 5/0
ans =
   Inf

>> a = Inf % alternativ a = inf
a =
   Inf
```

2.2. Tipuri de date (clase)

În MATLAB pot fi folosite mai multe tipuri de date (sau clase) pentru valori *numerice, șiruri de caractere* sau *valori logice*. Dacă se dorește folosirea unei variabile care să înglobeze mai mulți parametri aparținând unor clase diferite se pot folosi tipurile de date *table*, *cell* sau *struct*. Acestea sunt principalele tipuri de date utilizate în MATLAB, reprezentate și în diagrama de mai jos.

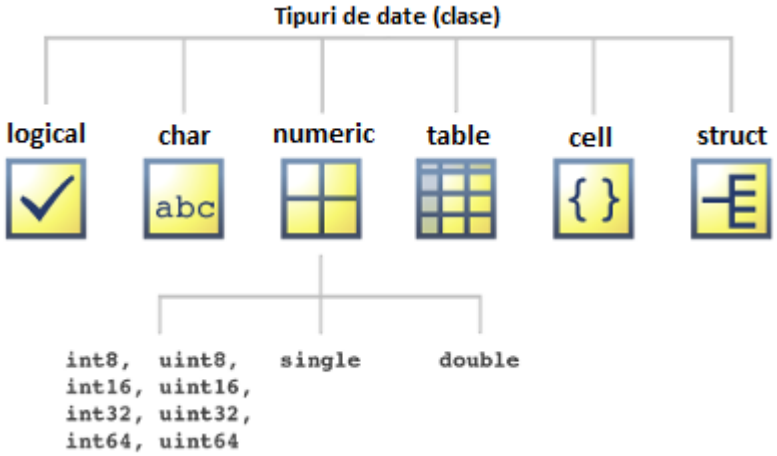


Figura 2.2. Principalele tipuri de date în MATLAB [1]

Pentru valori numerice, tipul implicit de date este *double*.

2.1.1 Tipul de date logical

O variabilă de tip *logical* poate avea doar valorile 0 sau 1. Pentru memorarea unui scalar de tip *logical* este nevoie de 1 octet.

`A = logical(B)` va avea următorul rezultat: `A` va avea valoarea logică `1` pentru orice valoare a lui `B` nenulă; `A` va avea valoarea logică `0` pentru valori nule ale lui `B`.

- Fie un vector `B` cu valori de tip *double*. Să se convertească într-un vector `A` cu valori de tip *logical*.

```
>> B = [-1 0 2.75 -1.4 0 34];
>> A = logical(B)
A =
     1     0     1     1     0     1
```

O variabilă de tip *logical* se poate obține atunci când se caută elementele dintr-o matrice (sau vector) ce respectă o anumită proprietate. De asemenea, se poate obține o matrice de tip *logical* atunci când se transformă o imagine grayscale într-o imagine alb-negru.

Fie o imagine grayscale de dimensiune 5 x 5 (care conține pixeli cu valori pseudorandom). Să se transforme imaginea grayscale într-o imagine binară astfel: toți pixelii cu intensitatea nivelului de gri mai mică decât 100 vor deveni negri (vor avea valoarea 0) și toți pixelii cu intensitatea nivelului de gri mai mare sau egală cu 100 vor deveni albi (vor avea valoarea 1).

```
% se genereaza o matrice de tip double de dimensiune 5 x 5
% cu valori random numere naturale in intervalul 0 - 255
A = randi([0, 255],5)
% matricea B va fi de tip logical
B = (A >= 100)
```

Variabila `A` generată pseudorandom va conține elementele:

```
>> A
A =
   229    79   232   128   228
    25    45   161   110    35
    11    86    25   255    99
   142    53   100   207   237
   197   130    13   124   234
```

Variabila `B` de tip `logical` va avea valorile:

```
>> B
B =
     1     0     1     1     1
     0     0     1     1     0
     0     0     0     1     0
     1     0     1     1     1
     1     1     0     1     1
```

2.1.2 Tipul de date char

O variabilă *char* (un caracter) este memorat pe 2 octeți. Mesajele (*string*) sunt interpretate de MATLAB ca matrice (adesea vectori linie) de variabile *char*. Pentru a scrie un mesaj, acesta trebuie să fie încadrat de ‘ ’ (două semne apostrof).

Figura 2.3. Tipul de date *char*

În exemplul din *Figura 2.3*, deoarece variabila `mesaj` conține 18 caractere, aceasta este stocată pe 36 de octeți. Dacă variabila `X` stochează o valoare numerică, atunci `char(X)` va avea ca rezultat codul ASCII asociat acelei valori (pentru valori între 0 și 127). Pentru valori între 128 și 65535 rezultatul variază în funcție de setările calculatorului.

```
>> var1 = char(97)
var1 =
a

>> var2 = char(123)
var2 =
{

% caracterul cu codul ASCII egal cu 10 este "linie noua"
>> var3 = char(10)
var3 =
```

Funcții utile în prelucrarea șirurilor de caractere

Concatenarea mai multor șiruri de caractere se realizează cu funcția `strcat`.

Sintaxă: `sir = strcat(s1, s2, ..., sn)`.

```
>> s1 = 'sir ';
>> s2 = 'de ';
>> s3 = 'caractere';
>> sir = strcat(s1, s2, s3)
sir =
sirdecaractere
```

Observație: folosind funcția `strcat` nu se pot adăuga spații între șirurile de caractere.

O altă variantă de a concatena șiruri de caractere (care permite și păstrarea spațiilor libere) este următoarea: `[s1, ..., sn]`.

```
>> s1 = 'sir ';
>> s2 = 'de ';
>> s3 = 'caractere';
>> sir = [s1, s2, s3]
sir =
sir de caractere
```

Conversia unui șir de caractere într-un număr, se realizează cu funcția `str2num` (sau funcția `str2double`). *Observație*: conversia este posibilă numai dacă șirul de caractere este reprezentarea unei valori numerice.

Sintaxă: `val_num = str2num(sir_caractere)`

Figura 2.4. Conversia din șir de caractere în valoare numerică

Dacă se dorește conversia unei valori numerice într-un șir de caractere se folosește funcția `num2str`.

Sintaxă: `val_str = num2str(val_numerică)`

Figura 2.5. Conversia din valoare numerică în șir de caractere

- Să se calculeze suma dintre două variabile numerice `a` și `b`. Apoi să se afișeze mesajul '`a + b =` ' urmat de suma dintre `a` și `b`.

```
>> a = 2;
>> b = 3;
>> expresie = ['a + b = ', num2str(a+b)];
>> disp(expresie)
    a + b = 5
```

- Să se afișeze mesajul următor pe 3 linii "*Acesta este un exemplu de folosire a funcției disp în care afișarea se face pe mai multe linii*".

```
>> disp(['Acesta este un exemplu', char(10),...
    'de folosire a functiei disp', char(10),...
    'in care afisarea se face pe mai multe linii'])

Acesta este un exemplu
de folosire a functiei disp
in care afisarea se face pe mai multe linii
```

Observație: dacă o linie de cod este prea lungă, aceasta poate fi scrisă pe mai multe linii folosind trei puncte (`...`) ca în exemplul de mai sus.

2.1.3 Tipul de date numeric

O variabilă numerică poate fi de tipul *integer*, *single* sau *double*.

2.1.3.1 Tipul de date integer

Acest tip de date poate fi *cu semn* sau *fără semn*, pe 1, 2, 4 sau 8 octeți.

Tabel 2.1. Tipul de date *integer*

Clasa	Domeniul de valori	Descriere
int8	$-2^7 \div 2^7-1$	Signed 8-bit integer
int16	$-2^{15} \div 2^{15}-1$	Signed 16-bit integer
int32	$-2^{31} \div 2^{31}-1$	Signed 32-bit integer
int64	$-2^{63} \div 2^{63}-1$	Signed 64-bit integer
uint8	$0 \div 2^8-1$	Unsigned 8-bit integer
uint16	$0 \div 2^{16}-1$	Unsigned 16-bit integer
uint32	$0 \div 2^{32}-1$	Unsigned 32-bit integer
uint64	$0 \div 2^{64}-1$	Unsigned 64-bit integer

MATLAB-ul stochează implicit datele numerice în formatul *double*. Pentru a stoca în format *integer* trebuie făcută conversia din *double* în tipul *integer* dorit.
Observații:

- dacă numărul ce se dorește a fi convertit într-un tip *integer* este un număr zecimal, atunci acesta va fi rotunjit la cel mai apropiat întreg.
- dacă numărul ce se dorește a fi convertit este mai mare/mic decât limita maximă/minimă a domeniului de valori a tipului de date dorit, atunci acesta va fi convertit la limita maximă/minimă a intervalului în care ia valori tipul de date în care se face conversia.

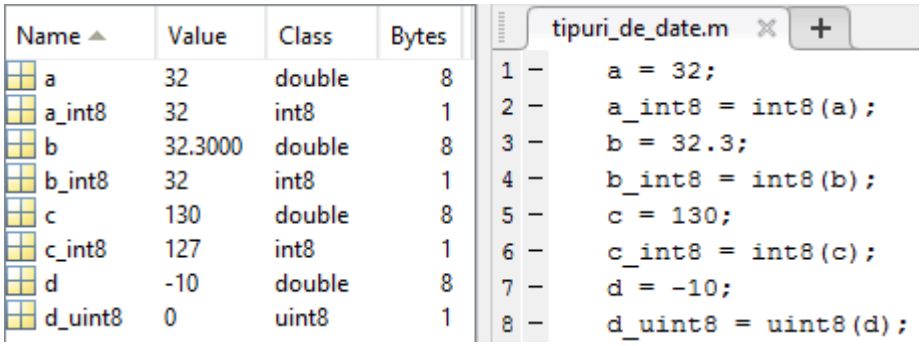


Figura 2.6. Conversia din `double` în `int8` și `uint8`

Dacă se dorește stocarea unei imagini grayscale (cu valori întregi între 0 și 255) nu are sens să salvăm imaginea în format *double* (8 octeți/pixel), este suficient să salvăm în format *uint8* (1 octet/pixel).

Pentru a afla limitele domeniului de valori ale unui tip de date *integer* se folosesc funcțiile `intmin` respectiv `intmax`.

- Să se afle limitele intervalului tipului de date `int8`.

```
>> intmin('int8')
ans =
 -128

>> intmax('int8')
ans =
         127
```

2.1.3.2 Tipul de date single

Este folosit pentru numere reale în precizie simplă. O variabilă de tip *single* este reprezentată pe 4 octeți. Funcțiile `realmin` și `realmax` întorc cea mai mică valoare pozitivă (respectiv cea mai mare valoare pozitivă) care poate fi stocată într-o variabilă de tip *real* (*single* sau *double*).

- Să se afle limitele intervalului tipului de date `single`.

```
val_min = realmin('single');
val_max = realmax('single');
disp(['O variabila de tip single poate lua valori intre', char(10),...
    num2str(-val_max),' si ', num2str(-val_min), char(10), ...
    ' si intre ',char(10), num2str(val_min), ' si ', num2str(val_max)])
```

Rezultatul rulării codului de mai sus este următorul:

```
      O variabila de tip single poate lua valori intre
      -3.402823466385289e+38 si -1.1755e-38
       si intre
      1.1755e-38 si 3.402823466385289e+38
```

Obs: `e+38` $= 10^{38}$

2.1.3.3 Tipul de date double

Este folosit pentru numere reale în dublă precizie cu virgulă mobilă. O variabilă de tip *double* este reprezentată pe 8 octeți. Acest tip de date este ales implicit de MATLAB pentru valorile numerice.

Intervalul de valori pentru o variabilă *double* este:

- pentru valori negative între -1.79769e+308 și -2.22507e-308
- pentru valori pozitive între 2.22507e-308 și 1.79769e+308

Formatul implicit de afișare a unui număr real este cu 4 cifre după virgulă. Acesta este formatul `short`. De multe ori ne interesează însă să vedem mai multe zecimale ale numărului și atunci putem folosi formatul `long` (permite vizualizarea a 15 cifre după virgulă). În afară de formatele `short` și `long` mai există și alte formate: `hex`, `compact` etc (pentru a vedea toate formatele disponibile rulați `help format`).

- Să se afișeze `pi` cu 4 zecimale și $\sqrt{2}$ cu 15 zecimale.

```
      >> format short % implicit
      >> pi
      ans =
          3.1416

      >> format long
      >> sqrt(2)
      ans =
         1.414213562373095
```

Pentru a afla distanța dintre un număr și următorul număr mai mare în dublă precizie se folosește sintaxa `eps(număr)`.

2.1.4 Tipul de date cell

O astfel de variabilă conține mai multe celule, fiecare celulă putând conține orice tip de dată, inclusiv o altă variabilă *cell*. Pentru a defini o variabilă *cell*, conținutul ei trebuie scris între acolade. Pentru a accesa un element al celulei, se folosește sintaxa `variabilă{index}`.

- Să se salveze într-o variabilă `A` de tip *cell* numele unui oraș și numărul de locuitori ai acelui oraș. Să se salveze apoi în `var1` primul element din `A` și în `var2` al doilea element din `A`.

```
A = {'Sibiu',425906}
var1 = A{1}
var2 = A{2}
```

În urma rulării codului de mai sus, în *Command Window* se va afișa:

```
A =
    'Sibiu'    [425906]

var1 =
     Sibiu

var2 =
     425906
```

Așa cum se poate observa din fereastra *Workspace*, `var1` este de tip *char* și `var2` este de tip *double*.

Observație: Pentru a accesa un element al unei variabile de tip *cell* se poate folosi și sintaxa `var = variabilă(index)` în loc de `var = variabilă{index};` în acest caz, variabila `var` va fi de tip `cell`.

```
>> A = {'Sibiu',425906};
>> var = A(1)
var =
    'Sibiu'
```

Pentru a extrage în mod direct al `n`-lea element dintr-o matrice ce face parte dintr-o celulă, se poate folosi sintaxa: `variabilă_cell{index}(n)`.

- Fie o variabilă `cell` ce conține trei parametri: un vector linie ce conține date de tip *double* (rezultatele unor algoritmi de clasificare), un vector linie ce conține date de tip `logic` (1 pentru rezultate mai mari de 90% și 0 în rest), și o variabilă de tip `cell` ce conține metodele utilizate. Să se afișeze al doilea element al fiecărui parametru.

```
rezultate = [87.3 67.22 93.12];
rezultate_admise = (rezultate > 90);
metode = {'Metoda1', 'Metoda2' ,'Metoda3'};
A = {rezultate, rezultate_admise, metode}
val1 = A{1}(2)
val2 = A{2}(2)
val3 = A{3}(2)
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
A =
    [1x3 double]    [1x3 logical]    {1x3 cell}

val1 =
   67.2200

val2 =
     0

val3 =
    'Metoda2'
```

S-au obținut astfel variabilele `val1` de tip *double*, `val2` de tip *logical* și `val3` de tip *cell*. Pentru ca `val3` să fie de tip *char* se putea folosi `val3 = A{3}{2}` în loc de `val3 = A{3}(2)`.

```
>> val3 = A{3}{2}
val3 =
Metoda2
```

Observație: folosirea celulelor este foarte utilă atunci când se dorește afișarea unui text pe mai multe linii (alternativă la folosirea lui `char(10)`; vezi exemplul de la tipul *char*).

Să se afișeze mesajul următor pe 3 linii "*Acesta este un exemplu de folosire a celulelor pentru afișarea unui text pe mai multe linii*".

```
>> disp({'Acesta este un exemplu';...
      'de folosire a celulelor';...
      'pentru afisarea unui text pe mai multe linii'})

    'Acesta este un exemplu'
    'de folosire a celulelor'
    'pentru afisarea unui text pe mai multe linii'
```

2.1.5 Tipul de date struct

O variabilă de tip `struct` poate conține mai multe câmpuri ale căror valori pot aparține oricărui tip de date.

Sintaxă: `S = STRUCT('param1',VAL1,'param2',VAL2,...)`

- Să se realizeze o variabilă de tip `struct` care să conțină parametrii unei rețele SOM: topologie, dimensiune și număr neuroni din stratul de intrare.

```
topo = 'hexagonala';
dim = [10, 10];
size_input = 23;
retea_SOM = struct('topologie', topo, 'dimensiune', dim,...
    'numar_neuroni_input',size_input);
```

Pentru a accesa un element al structurii se folosește sintaxa: `nume_structura.param`. Pentru a accesa parametrul `dimensiune` se va scrie:

```
>> retea_SOM.dimensiune
ans =
    10    10
```

- Fie o imagine grayscale care conține fundal și 3 obiecte. În urma segmentării imaginii s-a obținut matricea etichetelor `A` (fiecare element din `A` este un număr ce indică clasa căreia aparține pixelul de pe acea poziție). Să se salveze într-o matrice coordonatele centrelor tuturor celor 3 obiecte.

Observație: în MATLAB există funcția `regionprops` care pentru fiecare regiune din imagine calculează diverse proprietăți precum: centroidul, aria, excentricitatea etc. Proprietatea *Area* întoarce numărul pixelilor dintr-o regiune, proprietatea *Centroid* întoarce coordonatele centrului unei regiuni. Un pixel aparține unei regiuni dacă are eticheta diferită de 0. Dacă eticheta este 0 atunci acel pixel aparține fundalului.

```
A = [0 0 0 0 0;
     0 1 1 1 3;
     0 0 0 0 3
     2 2 2 2 3
     0 0 0 0 3];
regiuni = regionprops(A,'Area','Centroid');
```

În acest moment s-a creat variabila `regiuni` care este o *structură* cu 2 câmpuri (*Area* și *Centroid*).

```
      >> regiuni
      regiuni =
      3x1 struct array with fields:
          Area
          Centroid
```

Dacă se va da *dublu click* pe variabila `regiuni` din fereastra *Workspace* o să apară următorul tabel:

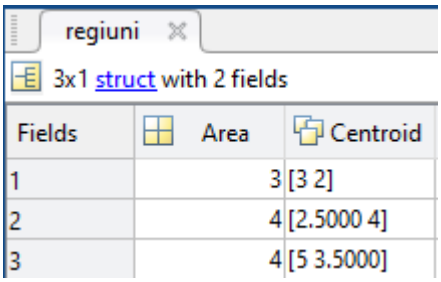


Figura 2.7. Vizualizare variabilă de tip *struct*

Variabila `regiuni` are 3 linii deoarece în imagine sunt 3 regiuni.

- Să se salveze într-un vector coloană numărul de pixeli din fiecare regiune. Să se salveze într-o matrice centrele tuturor regiunilor.

Observație: se va folosi funcția `cat` care realizează concatenarea elementelor de-a lungul unei dimensiuni specificate (dimensiune = 1➔ concatenare pe verticală, dimensiune = 2 ➔ concatenare pe orizontală).

```
>> nr_pixeli_regiuni = cat(1, regiuni.Area)
nr_pixeli_regiuni =
     3
     4
     4
>> centre_regiuni = cat(1, regiuni.Centroid)
centre_regiuni =
    3.0000    2.0000
    2.5000    4.0000
    5.0000    3.5000
```

Dacă se dorește însă să se afișeze doar numărul de pixeli ai regiunii cu eticheta 2, atunci se va scrie:

```
>> regiuni(2).Area
ans =
     4
% sunt 4 pixeli cu eticheta 2
```

2.1.6 Tipul de date table

Acest tip de date este foarte util atunci când se dorește stocarea tabelară a datelor experimentale (datele pot fi organizate ca într-un fișier *Excel*).

Sintaxă: `tabel = table(var1,...,varN)`

Un tabel poate conține date de diferite tipuri însă pe fiecare coloană trebuie să existe același număr de elemente. Există multe moduri în care poate fi inițializat un tabel și multe funcții pentru a prelucra apoi datele din tabel. În continuare vor fi exemplificate doar câteva noțiuni de bază. Pentru mai multe informații accesați *help*-ul funcției `table`.

- Pentru segmentarea unor imagini s-au folosit următoarele metode: {*RBF*, *Perceptron Multistrat* și *Bayes*}. Scorurile de clasificare corectă au fost de {89.5%, 92.72% respectiv de 87.77%}. Să se salveze aceste date într-un tabel.

```
algoritmi = {'RBF'; 'Perceptron Multistrat'; 'Bayes'};
procentaj = [89.5; 92.72; 87.77];
tabel_rezultate = table(algoritmi, procentaj)
```

Rezultatul rulării liniilor de mai sus este:

algoritmi	procentaj
'RBF'	89.5
'Perceptron Multistrat'	92.72
'Bayes'	87.77

După cum se poate observa în exemplul de mai sus, numele unei coloane este identic cu numele variabilei în care au fost salvate valorile acelei coloane (în speță *algoritmi* și *procentaj*). Dacă se dorește ca numele unei coloane să fie diferit de numele variabilei, se poate specifica acest lucru folosind parametrul 'VariableNames'.

- Pentru exemplul de mai sus, numele coloanelor să fie *Metode* și *Rezultate*.

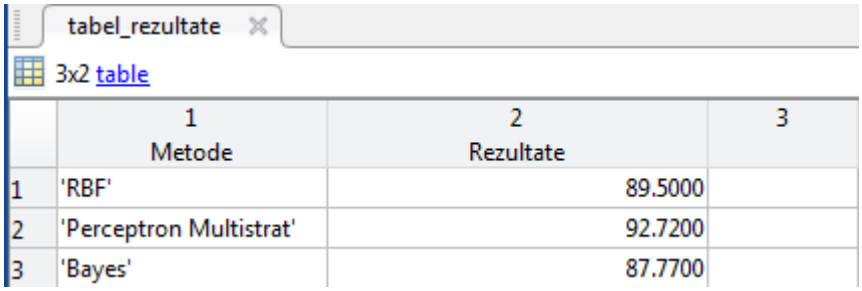
```
algoritmi = {'RBF'; 'Perceptron Multistrat'; 'Bayes'};
procentaj = [89.5; 92.72; 87.77];
tabel_rezultate = table(algoritmi, procentaj,...
    'VariableNames',{'Metode' 'Rezultate'})
```

Rezultatul rulării liniilor de mai sus este:

tabel_rezultate =

Metode	Rezultate
'RBF'	89.5
'Perceptron Multistrat'	92.72
'Bayes'	87.77

Dacă dați dublu-click pe variabila `tabel_rezultate` din *Workspace*, se va deschide următorul tabel:



tabel_rezultate

3x2 table

	1 Metode	2 Rezultate	3
1	'RBF'	89.5000	
2	'Perceptron Multistrat'	92.7200	
3	'Bayes'	87.7700	

Figura2.8. Vizualizarea unei variabile de tip *table*

Pentru a accesa un element dintr-un tabel se folosește sintaxa: `nume_tabel.nume_coloană(index)`.

- Pentru exemplul de mai sus, să se găsească metoda pentru care s-a obținut cel mai bun rezultat.

```
[sortare_rez, poz] = sort(tabel_rezultate.Rezultate, 'descend')
% sortare_rez este un vector coloana ce contine
% rezultatele sortate descrescator

% vectorul poz contine pozitiile initiale ale rezultatelor
% tabel_rezultate.Rezultate(poz) = sortare_rez
metoda_castigatoare = tabel_rezultate.Metode(poz(1));
disp(['Metoda care a dus la cel mai bun rezultat este:',...
      metoda_castigatoare])
```

În urma rulării codului de mai sus, în *Command Window* se va afișa:

```
sortare_rez =
   92.7200
   89.5000
   87.7700
poz =
     2
     1
     3
'Metoda care a dus la cel mai bun rezultat este:'
'Perceptron Multistrat'
```

În MATLAB se pot salva și citi date cu aproape orice format; se pot importa date din *Excel*, se pot citi și salva imagini, se pot încărca și reda semnale audio și semnale video, se pot salva și încărca datele obținute în urma simulărilor din MATLAB etc

3.1 Salvarea și încărcarea fișierelor **.mat*

Pentru salvarea variabilelor în fișiere **.mat* se folosește funcția `save`.

Sintaxă: `save('nume_fisier', 'var_1', 'var_2', … ,'var_n')`

Observație: dacă se dorește salvarea tuturor variabilelor din *Workspace* se folosește sintaxa `save('nume_fisier')`.

- Să se genereze pseudorandom două matrice: o matrice `A` cu 3 linii și 100 de coloane cu valori distribuite uniform între 1 și 2 și o matrice `B` cu 3 linii și 50 de coloane cu valori distribuite uniform între 2 și 3. Să se salveze în fișierul *matrice_pseudorandom.mat* cele două matrice.

```
A = ones(3,100) + rand(3,100);
B = 2*ones(3,50) + rand(3,50);
save('matrice_pseudorandom', 'A','B')
```

La sfârșitul rulării programului de mai sus, în folderul curent o să apară fișierul *matrice_pseudorandom.mat.*

Pentru încărcarea variabilelor dintr-un fișier **.mat* se folosește funcția `load`.

Sintaxă: `load('nume_fisier')`

Folosind sintaxa de mai sus se vor încărca toate variabilele salvate în fișierul *nume_fișier.mat.*

Observație: Dacă se dorește încărcarea doar a anumitor variabile se folosește sintaxa: `load('nume_fisier','var_1', 'var_2')` iar în acest caz se vor încărca doar variabile `var_1` și `var_2`.

- Să se încarce toate variabile din fișierul *matrice_pseudorandom.mat* salvat anterior. Să se calculeze media elementelor din `A` și media elementelor din `B`.

```
clear all
load('matrice_pseudorandom')
medie_A = mean(A(:))
medie_B = mean(B(:))
```

În urma rulării programului de mai sus se vor afișa rezultatele `medie_A = 1.5219` și `medie_B = 2.5154`.

3.2 Citirea, scrierea și redarea semnalelor audio

Citirea unui semnal audio

`Audioread` este o funcție de citire a fișierelor audio care înlocuiește funcții mai vechi (cum ar fi `auread` și `wavread`). Sintaxa este simplă, având la intrare numele sau calea completă a fișierului și opțional următorii parametri (numărul eșantionului de *start* și al celui de *stop*, în caz că se dorește încărcarea unei porțiuni din semnal, respectiv tipul de date al variabilei de ieșire), iar la ieșire semnalul audio `Y` și frecvența de eșantionare `Fs`.

Sintaxă: `[Y, Fs] = audioread('nume_fisier', [start stop], tip_date)`

Dacă semnalul are un singur canal audio, ieșirea `Y` va fi un vector coloană. Dacă semnalul este stereo sau sunt mai multe canale, fiecare canal va fi pe câte o coloană din matricea `Y`.

- Să se citească un semnal audio.

```
[semnal, Fs] = audioread('Prokofiev.wav');
```

În urma rulării programului de mai sus, în fereastra *Workspace* au apărut variabilele `semnal` și `Fs`. Se poate observa că semnalul citit este pe un singur canal, a fost eșantionat cu `Fs` = 48000Hz și are 960000 de eșantioane.

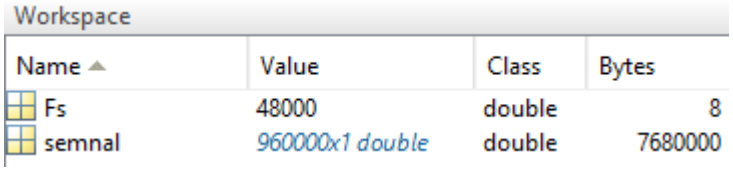


Figura 3.1. Fereastra *Workspace* după citirea semnalului `'Prokofiev.wav'`

Scrierea unui semnal audio

Scrierea fișierelor de tip audio se poate realiza folosind funcția `audiowrite`.

Sintaxă: `audiowrite('nume_fisier', Y, FS)`

`Y` este semnalul ce se dorește a se salva iar `Fs` este frecvența de eșantionare.

Formatul în care este scris fișierul este dedus din extensia parametrului *nume_fisier*.

```
Format          File Extension(s)    Compression Method
-------------------------------------------------------
Wave            .wav                 None
MPEG-4 Audio    .m4a, .mp4           AAC
FLAC            .flac                FLAC (Lossless)
Ogg/Vorbis      .ogg, .oga           Vorbis
```

Figura 3.2. Formatul și extensia unui semnal audio

Parametrii opționali pentru scrierea fișierului audio sunt: numărul de biți pe eșantion; rata de compresie (kbps), pentru formate audio cu compresie; titlu, artist, comentariu, care reprezintă informații adiționale atașate semnalului audio.

- Să se citească un semnal audio `Y`. Să se formeze un alt semnal audio `X` care să conțină doar prima jumătate a semnalului audio `Y`. Să se salveze semnal audio `X`.

```
[Y, Fs] = audioread('Prokofiev.wav');
X = Y(1:length(Y)/2);
audiowrite('Prokofiev_2.wav', X, Fs)
```

Redarea unui semnal audio

Pentru redarea unui semnal audio se folosește funcția `sound`.

Sintaxă: `sound(Y, Fs)`

`Y` este semnalul ce se dorește a se reda iar `Fs` este frecvența de eșantionare.

- Să se citească un semnal audio `Y`. Să se redea semnalul audio citit.

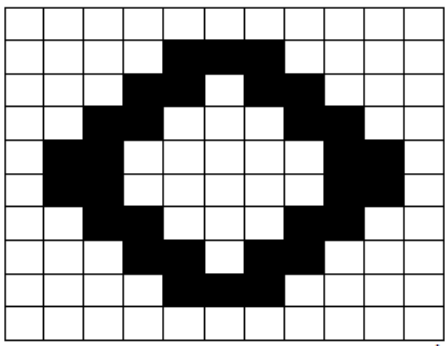
```
[Y, Fs] = audioread('Prokofiev.wav');
sound(Y, Fs)
```

Pentru a extrage informații despre semnalul audio, fără a citi semnalul în prealabil, se poate folosi funcția `audioinfo`.

3.3 Citirea, afișarea și salvarea imaginilor

Imaginile digitale sunt reprezentate în MATLAB folosind matrice.

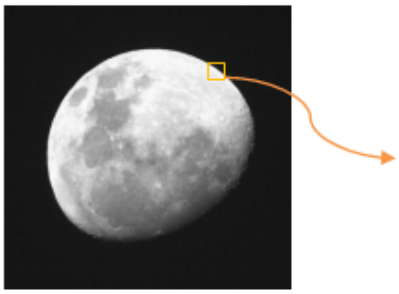
Imagine binară. O imagine binară (*alb – negru*) poate fi reprezentată folosind o matrice ce conține numai valorile 0 și 1 (unde 0 reprezintă *negru* iar 1 reprezintă *alb*).



1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1
1	1	1	0	0	1	0	0	1	1	1
1	1	0	0	1	1	1	0	0	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	0	1	1	1	1	1	0	0	1
1	1	0	0	1	1	1	0	0	1	1
1	1	1	0	0	1	0	0	1	1	1
1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

Figura 3.3. Reprezentarea matriceală a unei imagini binare (alb – negru)

Imagine grayscale. Imaginile cu niveluri de gri (imagini *grayscale*) pot fi și ele reprezentate ca matrice, fiecare element al matricei reprezentând intensitatea pixelului respectiv. Valorile intensităților se exprimă în mod uzual pe 8 biți, cu alte cuvinte sunt disponibile 256 de niveluri de gri pentru intensitatea fiecărui pixel.
Observație: o imagine *grayscale* poate fi normată, având doar valori în intervalul [0, 1] (unde 0 reprezintă negru iar 1 reprezintă alb).



26	27	24	25	25	25	20	24	24	24
29	27	26	25	28	24	24	25	25	25
94	34	28	26	27	23	25	24	22	24
237	171	59	29	27	25	25	25	25	24
246	245	220	131	41	28	26	26	26	27
239	243	247	243	187	67	29	27	26	24
239	241	245	246	246	217	98	31	29	26
240	239	239	241	244	246	233	141	39	27
236	238	240	236	240	241	246	237	168	47
235	233	235	232	234	235	240	245	241	182

Figura 3.4. Reprezentarea matriceală a unei imagini grayscale

Imagine color. O imagine în spațiul color RGB este reprezentată în MATLAB ca o matrice cu trei straturi: stratul de roșu (Red), stratul de verde (Green) și stratul de albastru (Blue).

- Exemplu de imagine RGB în MATLAB.

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Strat_R

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Strat_G

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Strat_B

Dacă imaginea `I` are pe primul strat matricea `Strat_R`, pe stratul 2 matricea `Strat_G` și pe stratul 3 matricea `Strat_B`, atunci imaginea va arăta astfel:

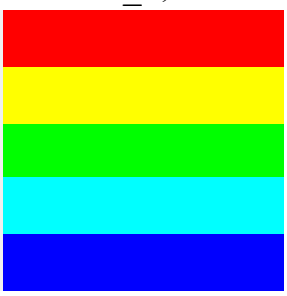


Figura 3.5. Imaginea `I`

Observație: există mai multe spații de culoare care se utilizează în funcție de aplicație: CMYK (pentru printare), HSV, CIELa*b* etc.

3.3.1 Citirea imaginilor folosind funcția `imread`

Funcția `imread` permite citirea imaginilor binare, grayscale și color.

Sintaxă: `A = imread('nume_imagine', 'ext')`, unde:

- parametrul `nume_imagine` reprezintă numele imaginii
- parametrul `ext` reprezintă extensia imaginii (`jpg`, `png`, `bmp` etc)

Observație. Dacă lipsește parametrul `ext`, atunci numele imaginii trebuie să conțină și extensia (ex: `imread('imagine.jpg')`). Dacă imaginea nu se află în folderul curent, la numele imaginii trebuie specificată întreaga cale.

Dacă imaginea este grayscale atunci matricea `A` va avea dimensiunea `M x N`, unde `M` reprezintă numărul de linii și `N` numărul de coloane.

Dacă imaginea este color, atunci matricea `A` va avea dimensiunea `M x N x 3`, cele 3 straturi reprezentând planurile RGB (*Observație*: pentru imaginile `tiff` pot exista mai multe straturi).

- Să se citească o imagine și să se afișeze dimensiunile acesteia.

```
>> I = imread('corabie.jpg');
>> [m,n,p] = size(I)
m =
         514
n =
         685
p =
         3
```

Pentru a realiza conversia unei imagini color într-o imagine grayscale se folosește funcția `rgb2gray`.

Sintaxă: `I_gray = rgb2gray(I_RGB)`

Funcția `rgb2gray` realizează conversia unei imagini `I_RGB` din spațiul color într-o imagine `I_gray` grayscale.

- Să se facă conversia `rgb2gray` și să se afișeze dimensiunile imaginii obținute.

```
>> I = imread('corabie.jpg');
>> J = rgb2gray(I);
>> [m,n,p] = size(J)
m =
         514
n =
         685
p =
         1
```

3.3.2 Afișarea unei imagini folosind funcția `imshow`

Funcția `imshow` permite afișarea unei imagini binare, grayscale sau color.

Sintaxă: `imshow(I)`

Observații:

- Pentru o imagine color, dacă valorile imaginii sunt de tip `double`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[1,1,1]` reprezintă un pixel alb.
- Pentru o imagine color, dacă valorile imaginii sunt de tip `uint8` sau `uint16`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[255,255,255]` reprezintă un pixel alb.

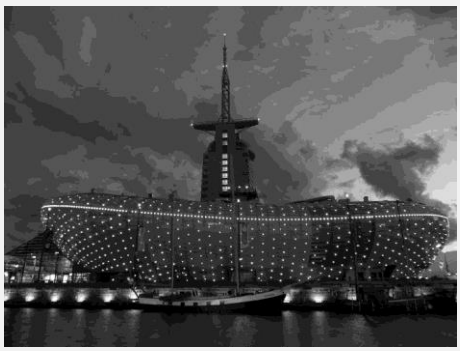
Ținând cont de afirmațiile de mai sus, trebuie avut grijă ca:

- Dacă imaginea este de tip `double` și valorile pixelilor pe fiecare plan de culoare sunt cuprinse între 0 și 255, pentru a putea reprezenta corect imaginea trebuie folosită sintaxa: `imshow(I/255)`, altfel orice pixel cu valoarea mai mare de 0 va fi reprezentat cu alb.

- Să se citească o imagine RGB și să se afișeze. Să se transforme imaginea color într-o imagine grayscale și să se afișeze.

```
I_RGB = imread('corabie.jpg');
I_gray = gray2rgb(I_RGB);
figure(1), imshow(I_RGB)
figure(2), imshow(I_gray)
```

Imagine RGB



Imagine grayscale

Pentru o imagine grayscale, funcția `imshow` poate fi apelată și astfel:

- `imshow(I, [val_min, val_max]);` în acest caz, valorile dintre *val_min* și *val_max* sunt scalate între *negru* și *alb;* pixelii cu valori mai mari decât `val_max` vor fi reprezentați cu *alb* iar pixelii cu valori mai mici decât `val_min` vor fi reprezentați cu *negru;*
- `imshow(I, [ ]);` în acest caz, nivelurile de gri ale imaginii sunt aduse între *negru* și *alb*; sintaxa este echivalentă cu `imshow[min(I(:)) max(I(:))]`

3.3.3 Afișarea unei imagini folosind funcția `image`

Funcția `image` permite afișarea unei matrice bidimensionale ca o imagine grayscale și a unei matrice cu 3 straturi ca o imagine color.

Sintaxă: `image(I)`

Ca și în cazul funcției `imshow`:

- Pentru o imagine color, dacă valorile imaginii sunt de tip `double`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[1,1,1]` reprezintă un pixel alb.
- Pentru o imagine color, dacă valorile imaginii sunt de tip `uint8` sau `uint16`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[255,255,255]` reprezintă un pixel alb.

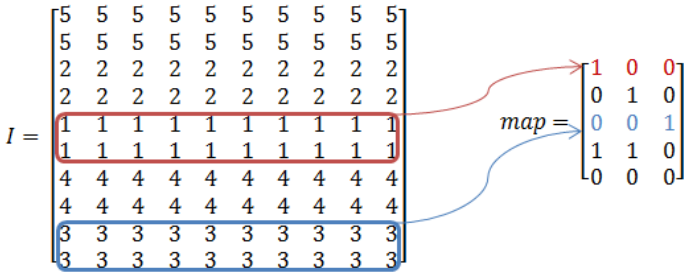
Observație: Spre deosebire de funcția `imshow`, funcția `image` afișează și coordonatele pixelilor.

- Să se citească o imagine color și apoi să se afișeze.

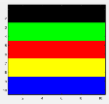
```
I_RGB = imread('bulz.jpg');
figure(1), image(I_RGB)
```

Dacă matricea ce se dorește a fi afișată ca imagine este de dimensiuni `M x N` (având un singur strat), se poate asocia fiecărei valori din matrice o culoare prin intermediul funcției `colormap(map)`; `map` este o matrice cu 3 coloane, iar numărul de linii este egal cu numărul culorilor utilizate. Prima coloană reprezintă componenta de `roșu`, a doua coloană reprezintă componenta de `verde` iar a treia coloană reprezintă componenta de `albastru`.

- Fie o matrice cu 10 linii și 10 coloane având valorile [1, 2, 3, 4, 5]. Se dorește să se reprezinte această matrice ca o imagine colorată astfel: 1➔roșu, 2➔verde, 3➔albastru, 4➔galben, 5➔negru.




```
I = [5*ones(2,10); 2*ones(2,10); ones(2,10); 4*ones(2,10);
3*ones(2,10)];
map = [1 0 0; 0 1 0; 0 0 1; 1 1 0; 0 0 0];
image(I), colormap(map)
```



Există o serie de hărți predefinite precum: `gray`(niveluri de gri), `autumn` (nuanțe de portocaliu și roșu), `vga`(16 culori) etc. Pentru a vedea mai multe hărți de culoare tastați `help GRAPH3D`.

3.3.4 Afișarea unei imagini folosind funcția `imagesc`

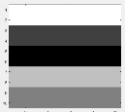
Funcția `imagesc` permite afișarea unei matrice `I` ca o imagine. Spre deosebire de funcția `image`, funcția `imagesc` scalează valorile astfel încât să se folosească întreaga hartă de culoare.

Sintaxă: `imagesc(I)`

Exemplu: dacă o matrice conține doar valorile 1, 2, 3, 4, 5 și se afișează folosind harta de culori `gray(256)`, adică 256 de niveluri de gri, cea mai mare valoare (adică 5) va fi afișată cu alb, cea mai mică valoare (adică 1) va fi afișată cu negru, iar valorile intermediare (2, 3, 4) vor fi afișate cu valori intermediare ale nivelurilor de gri.

- Fie o matrice cu 10 linii și 10 coloane având valorile [1, 2, 3, 4, 5]. Se dorește să se reprezinte această matrice folosind harta de culori `gray` cu 256 niveluri de gri.

```
I = [5*ones(2,10); 2*ones(2,10); ones(2,10); 4*ones(2,10);
3*ones(2,10)];
imagesc(I), colormap(gray(256))
```



3.3.5 Salvarea imaginilor folosind funcția `imwrite`

Sintaxă: `imwrite(I, 'nume_imagine', 'ext')`
Se salvează matricea `I` ca imagine cu numele *nume_imagine.ext*

- parametrul nume_`imagine` reprezintă numele imaginii
- parametrul `ext` reprezintă extensia imaginii (`jpg`, `png`, `bmp` etc)

Observație. Dacă lipsește parametrul `ext`, atunci numele imaginii trebuie să conțină și extensia (ex: `imwrite(I, 'nume_imagine.jpg')`). Dacă `nume_imagine` conține și calea către un folder, atunci imaginea va fi salvată în acel folder; dacă nu, imaginea va fi salvată în folderul curent.

- Să se modifice imaginea de mai jos astfel încât roșiile să fie portocalii. Să se salveze imaginea în fișierul `roșii_portocalii.jpg`.

```
I = imread('rosii.jpg');
I_modif = I;
[y, x] = find((I(:,:,1)-I(:,:,2)>60)&(I(:,:,1)-I(:,:,3)>40));
for i = 1:length(y)
    I_modif(y(i),x(i),1) = I(y(i),x(i),1);
    I_modif(y(i),x(i),2) = I(y(i),x(i),1)/2;
    I_modif(y(i),x(i),3) = 0;
end
% se scrie imaginea I_modif
imwrite(I_modif, 'rosii_portocalii.jpg')
I_salvat = imread('rosii_portocalii.jpg');
figure(1), imshow(I)
figure(2), imshow(I_salvat)
```

Imagine originală

Imagine modificată și salvată

3.3.6 Operații asupra directoarelor și fișierelor externe

Pe lângă citirea/scrierea din/în fișiere, cu ajutorul MATLAB-ului se pot face și operații asupra structurii de fișiere externe, cum ar fi generarea și ștergerea de directoare, redenumirea, copierea sau mutarea fișierelor etc.

`isdir(DIR)` este o funcție ajutătoare care returnează 1 atunci când parametrul `DIR` este cale către un director și 0 în rest.

`[SUCCESS,MESSAGE,MESSAGEID] = mkdir(NEWDIR)` generează un director nou la calea indicată de parametrul `NEWDIR`. Parametrii de ieșire sunt opționali, însă pot ajuta la identificarea erorilor în cazul în care operația de creare a directorului a eșuat.

`rmdir(DIR)` șterge directorul de la calea indicată de parametrul `DIR`. `rmdir(DIR, 's')` șterge directorul DIR și toate subdirectoarele din acesta.

Pentru toate aceste comenzi, calea poate fi relativă sau absolută.

- Exemplu de generare a unui director în cazul în care acesta nu există deja.

```
if isdir('.. /tempdir')
    rmdir('../tempdir','s');
end
mkdir('../tempdir');
```

`[SUCCESS,MESSAGE,MESSAGEID]=copyfile(SOURCE,DESTINATION)` copiază fișierul sau directorul `SOURCE` la locația `DESTINATION`. Dacă `DESTINATION` lipsește, MATLAB-ul încearcă să copieze fișierul în directorul curent.

`[SUCCESS,MESSAGE,MESSAGEID]=movefile(SOURCE,DESTINATION)` are sintaxa similară cu `copyfile`. Fișierul sau directorul este mutat de la locația `SOURCE` la locația `DESTINATION`. Dacă `DESTINATION` lipsește, MATLAB încearcă să mute fișierul în directorul curent. Funcția `movefile` poate fi folosită și pentru redenumirea unui fișier:

```
movefile('..\aceeasi\cale\numeVECHI', '..\aceeasi\cale\numeNOU');
```

`delete(FILENAME)` șterge fișierul `FILENAME` de pe calculator. Ca parametru al funcției se pot folosi *wildcards*, de exemplul `delete('*.p')` pentru ștergerea tuturor fișierelor precompilate.

3.4 Funcții pentru selectarea regiunilor de interes dintr-o imagine

Funcția `imcrop`

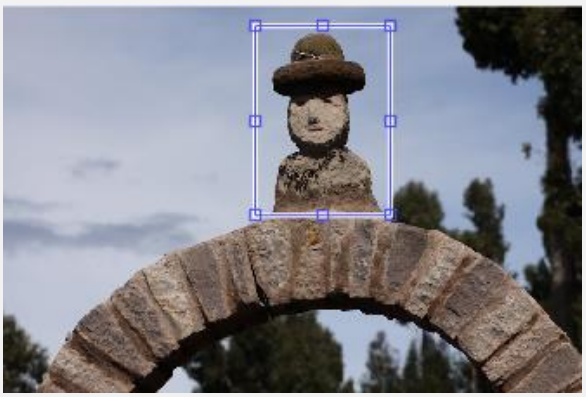
Funcția `imcrop` selectează dintr-o imagine o regiune de interes sub formă rectangulară. Coordonatele regiunii de interes:

- pot fi stabilite de utilizator, prin selectarea manuală a regiunii de interes din imagine (*Sintaxa 1*)
- se pot scrie într-un vector ce va fi trimis ca parametru de intrare al funcției `imcrop` (*Sintaxa 2*)

Sintaxă 1: `I_crop = imcrop(I)`
Folosind sintaxa de mai sus, utilizatorului i se permite să selecteze regiunea de interes, prin marcarea unei regiuni rectangulare în imaginea `I`, în zona de interes. Regiunea selectată se salvează în imaginea `I_crop`.

- Selectarea unei zone de interes rectangulare dintr-o imagine.

```
I = imread('peru.jpg');
I_crop = imcrop(I);
figure(1), imshow(I)
figure(2), imshow(I_crop)
```



Selectarea zonei de interes

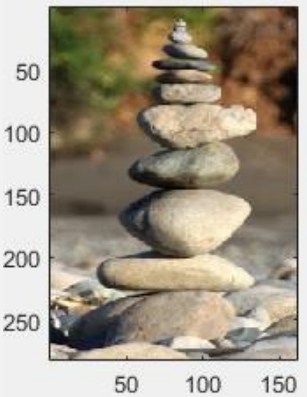
Zona de interes

Sintaxă 2: `I_crop = imcrop(I, [xmin, ymin, L, H])`
Folosind sintaxa de mai sus se poate selecta din imaginea `I` o suprafață rectangulară care pornește din colțul stânga sus al imaginii având coordonatele (`xmin, ymin`), lungimea `L` și înălțimea `H`. Regiunea selectată se salvează în imaginea `I_crop`.

- Selectarea unei zone de interes rectangulare prestabilite.

```
xmin = 400;
ymin = 140;
L = 160;
H = 280;
I = imread('pietre.jpg');
I_crop = imcrop(I,[xmin, ymin, L, H]);
figure(1), image(I)
figure(2), image(I_crop)
```

Imagine originală



Zona rectangulară selectată

Funcția `roipoly`

Funcția `roipoly` selectează dintr-o imagine o regiune de interes (ROI = Region of Interest) sub formă poligonală. Vârfurile poligonului se marchează prin click pe imagine. Poligonul se închide prin dublu-click.

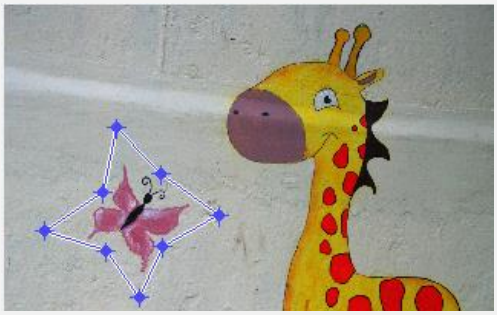
Sintaxă: `[BW,xi,yi] = roipoly(I),` unde:

- `I` este imaginea din care se selectează regiunea de interes.
- `BW` este o imagine binară de aceeași dimensiune cu imaginea `I`. `BW` are valoarea 1 pentru pixelii selectați în interiorul poligonului și 0 în rest.
- `xi,yi` sunt coordonatele punctelor ce alcătuiesc poligonul.

Observație: Funcția `roipoly` poate selecta un poligon prestabilit dacă la intrare primește coordonatele vârfurilor poligonului.

- Selectarea unei zone de interes poligonale dintr-o imagine.

```
I = imread('pictura.jpg');
BW = uint8(roipoly(I));
BW = repmat(BW,1,1,3);
I_seg = I.*BW;
figure(1), imshow(I)
figure(2), image(BW*255)
figure(3), image(I_seg)
```



Selectarea regiunii de interes

Regiunea de interes

Funcția `ginput`

Funcția `ginput` permite selectarea a `N` puncte din imaginea curentă și întoarce coordonatele `X` și `Y` ale punctelor.

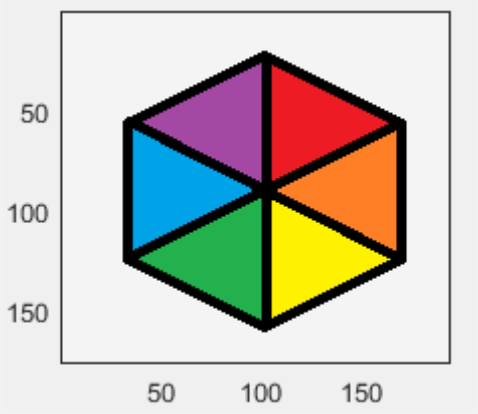
Sintaxă: `[X,Y] = ginput(N)`, unde:

- `N` reprezintă numărul de puncte selectate.
- `X` conține indicii coloanelor din imagine de unde au fost selectate punctele.
- `Y` este un vector cu indicii liniilor din imagine de unde au fost selectate punctele.

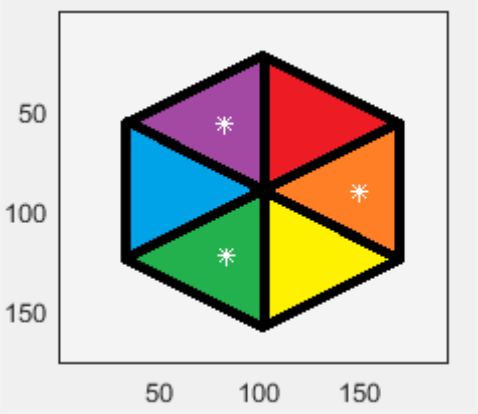
Observație: `[x,y] = ginput()` permite selectarea unui singur punct. Dacă rezultatul este perechea `[a,b]` ➔ a fost selectat pixelul de pe coloana `a` și linia `b`.

- Selectarea a 3 puncte dintr-o imagine și marcarea lor cu puncte albe.

```
I = imread('hexagon.png');
N = 3;
figure(1), image(I), truesize
[X, Y] = ginput(N);
figure(2),  image(I), truesize
hold on
    plot(X,Y,'w*')
hold off
% punctele selectate sunt [X, Y] = [83 58;149 90;82 126]
```



Imagine originală



Marcare puncte selectate

3.5 Citirea tuturor fișierelor dintr-un folder

Pentru a citi toate fișierele și subfolderele dintr-un folder se poate folosi funcția `dir`.
Sintaxă: `dir('folder')`
Rezultatul întors de funcția `dir` este salvat într-o variabilă de tip *struct* cu următoarele câmpuri: *name* (reprezintă numele fișierului sau folderului), *date*(ultima dată când s-a modificat fișierul sau folderul), *bytes*(numărul de octeți), *isdir*(1 dacă este folder, 0 dacă este fișier), *datenum.*

- Să se afișeze numele tuturor fișierelor dintr-un folder.

```
% path contine calea relativa sau absoluta catre folderul dorit
path = 'D:\carte Matlab\poze';
folder = dir(path);
len_folder = size(folder,1);
for index = 1:len_folder
    if (~folder(index).isdir())
        disp(folder(index).name)
    end
end
```

- Să se redenumească toate fișierele **.jpg* dintr-un folder.

```
% path contine calea relativa sau absoluta catre folderul dorit
path = 'D:\carte Matlab\poze';
folder = dir(path);
len_folder = size(folder,1);
k = 0;
for index = 1:len_folder
    fisier = folder(index).name;
    len_fisier = length(fisier);
    if (~folder(index).isdir()) &&...
        strcmpi(fisier(len_fisier-3:len_fisier),'.jpg')
            k = k+1;
            nume_vechi = [path,'\',fisier];
            nume_nou = [path,'\',num2str(k),'.jpg'];
            movefile(nume_vechi,nume_nou);
    end
end
```

Atenție! Această porțiune de cod redenumește fișierele. In urma rulării, vechile denumiri ale fișierelor **.jpg* vor fi pierdute, iar noile denumiri vor fi *1.jpg*, *2.jpg* etc, în ordinea alfabetică a vechilor denumiri.

4. Operații matematice cu scalari

Orice valoare numerică salvată într-o matrice de dimensiune 1 x 1 (o linie și o coloană) este un scalar. În MATLAB se pot realiza majoritatea operațiilor matematice folosind scalari: operațiile de bază (adunare, scădere, înmulțire, împărțire, ridicare la putere), funcții de aproximare, funcții trigonometrice, exponențiale, logaritmice etc.

Operații matematice de bază

Simbol	Operație	Exemplu
+	adunare	a + b
-	scădere	a - b
*	înmulțire	a * b
/	împărțire la dreapta	a/b
\	împărțire la stânga	a\b (echivalent cu b/a)
^	ridicare la putere	a^b (a la puterea b)

MATLAB-ul respectă regula matematică în ceea ce privește ordinea efectuării operațiilor: în lipsa parantezelor, mai întâi se efectuează ridicarea la putere, apoi înmulțirile și împărțirile iar la final adunările și scăderile.

- Să se calculeze $r = 2 + 3^2 \cdot 2 - 4$.

```
>>  r = 2 + 3^2 * 2 - 4
r =
    16
```

Pentru a indica ordinea realizării operațiilor se folosesc paranteze rotunde.

- Să se calculeze $r = (2 + 3^2) \cdot 2 - 4$.

```
>> r = (2 + 3^2) * 2 - 4
r =
    18
```

Funcții de rotunjire

Funcție MATLAB	Descriere
`round(x)`	Rotunjirea lui `x` la cel mai apropiat întreg
`ceil(x)`	Rotunjirea lui `x` la cel mai apropiat întreg spre plus infinit
`floor(x)`	Rotunjirea lui `x` la cel mai apropiat întreg spre minus infinit
`fix(x)`	Rotunjirea lui `x` la cel mai apropiat întreg spre zero

- Să se rotunjească numărul 3.7 la cel mai apropiat întreg, numărul -3.7 la cel mai apropiat întreg spre plus infinit și 3.7 la cel mai apropiat întreg spre minus infinit.

```
>> round(3.7)
ans =
     4
>> ceil(-3.7)
ans =
    -3
>> floor(3.7)
ans =
     3
```

Funcții matematice uzuale

Funcție în MATLAB	Expresia matematică echivalentă	Exemplu
`sin(x)`	`sin(x)` cu `x` exprimat în radiani	`sin(pi/2) = 1`
`cos(x)`	`cos(x)` cu `x` exprimat în radiani	`cos(pi/2) = 0`
`tan(x)`	`tg(x)` cu `x` exprimat în radiani	`tan(pi/4) = 1`
`atan(x)`	`arctg(x)` cu `x` exprimat în radiani	`atan(1) = 0.7854`
`sqrt(x)`	Rădăcina pătrată $\sqrt{x}$	`sqrt(9) = 3`
`exp(x)`	Funcția exponențială e^x	`exp(1) = 2.7183`
`abs(x)`	Funcția modul $\|x\|$	`abs(-5.2) = 5.2`
`log(x)`	Logaritmul natural $\ln(x)$	`log(10) = 2.3026`
`log10(x)`	Logaritmul în baza 10 $log_{10}(x)$	`log10(10) = 1`
`rem(x,y)`	Restul împărțirii lui `x` la `y`	`rem(15,4) = 3`

- Să se afle rezultatul expresiei $sin^2(x) + cos^2(x)$ pentru $x = \pi/6$.

```
>> x = pi/6;
>> (sin(x))^2+ (cos(x))^2
ans =
     1
```

- Să se calculeze densitatea de probabilitate a unei distribuții normale $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-m)^2}{2\sigma^2}}$ în $x = -2$. Se cunosc media $m = 1$ și varianța $\sigma^2 = 0.5$.

```
>> m = -2;
>> var = 0.5;
>> x = -2;
>> f = 1/sqrt(2*pi*var)*exp(-(x-m)^2/(2*var))
f =
   0.5642
```

Operatori relaționali

Operator	Semnificație	Exemplu
==	egal	`if (a == 2)…`
~=	diferit	`if (a ~= 2)…`
<	strict mai mic	`if (a < 2)…`
<=	mai mic sau egal	`if (a <= 2)…`
>	strict mai mare	`if (a > 2)…`
>=	mai mare sau egal	`if (a >= 2)…`

Să se genereze pseudorandom un număr natural `x` între 1 și 100. Dacă numărul este par atunci se va afișa mesajul "*Numărul x este par*", altfel se va afișa mesajul "*Numărul x este impar*", unde caracterul *x* va fi înlocuit cu valoarea variabilei `x`. *Observație*: funcția `randi` generează pseudorandom un număr natural.

```
x = randi([1, 100]);
if(rem(x,2)==0)
    disp(['Numarul ', num2str(x), ' este par'])
else
    disp(['Numarul ', num2str(x), ' este impar'])
end
```

În urma rulării de 2 ori a programului de mai sus s-a afișat în *Command Window*:

```
Numarul 98 este par
Numarul 49 este impar
```

Operatori logici

Dintre operatorii logici amintim `and`(&&), `or` (| |) și `not`(~).

Operator	Semnificație	Exemplu
&& (and)	ȘI logic	`1&&0=0, 1&&1=1, 1&&2=1, 0&&3=0, 0&&0=0`
|| (or)	SAU logic	`1||0=1, 1||1=1, 1||2=1, 0||3=1, 0||0=0`
~ (not)	NU logic	`~1=0, ~0=1, ~2=0`

Fie `x`, `y` și `z` valorile a 3 pixeli consecutivi aflați pe linia unei imagini grayscale. Dacă `x` are o valoare cuprinsă în intervalul (80, 120) și `z` are o valoare cuprinsă în intervalul (90, 110), atunci `y` va avea valoarea 100, altfel `y` = 0.

```
>> x = 90;
>> z = 106;
>> y = 100*((x > 80 && x < 120)&&(z > 90 && z < 110))
y =
  100
```

Se observă că pentru valorile lui `x` și `z` alese, se respectă condițiile și `y` devine 100. În exemplul de mai jos `x` va avea o valoare în afara intervalului iar `y` va deveni 0.

```
>> x = 78;
>> z = 106;
>> y = 100*((x > 80 && x < 120)&&(z > 90 && z < 110))
y =
     0
```

Operații cu numere complexe

Fie numărul complex cu forma generică $z = a + b \cdot i$. Știm că acest număr complex are conjugatul $\bar{z} = a - b \cdot i$, partea reală $Re(z) = a$, partea imaginară $Im(z) = b$ și modulul $abs(z) = \sqrt{a^2 + b^2}$. În MATLAB, pentru lucrul cu numere complexe se folosesc cel mai adesea următoarele funcții:

- `real`, pentru a afla partea reală a unui număr complex
- `imag`, pentru a afla partea imaginară a unui număr complex
- `conj`, pentru a determina conjugatul unui număr complex
- `abs`, pentru determinarea modulului unui număr complex

Fie un număr complex. Să se calculeze partea reală, partea imaginară, conjugatul și modulul numărului complex.

```
>> z = 3 + 5*i; % alternativ z = 3 + 5i
>> real(z)
ans =
     3

>> imag(z)
ans =
     5

>> conj(z)
ans =
   3.0000 - 5.0000i

>> abs(z)
ans =
    5.8310
```

Elementul de bază cu care lucrează MATLAB-ul este **matricea**, acest lucru sugerându-l chiar numele de MATLAB, care vine de la **"matrix laboratory"**. MATLAB-ul este optimizat pentru calcul matriceal, operațiile cu matrice (așa cum se va vedea) fiind foarte ușor de realizat.
Atenție! Forma de plural a cuvântului **matrice** este tot **matrice**.

O matrice de `m` linii și `n` coloane se definește matematic astfel:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Din punct de vedere al MATLAB-ului:

- un **scalar** (un număr) este o matrice cu o linie și o coloană (1 x 1)
- un **vector linie** este o matrice cu o singură linie (1 x `n`)

$$A = [a_{11}\ a_{12}\ a_{13}\ \dots\ a_{1n}]$$

- un **vector coloană** este o matrice cu o singură coloană (`m` x 1)

$$A = \begin{bmatrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{bmatrix}$$

- o matrice cu mai multe straturi este o **matrice tridimensională** (sau *masivă*).

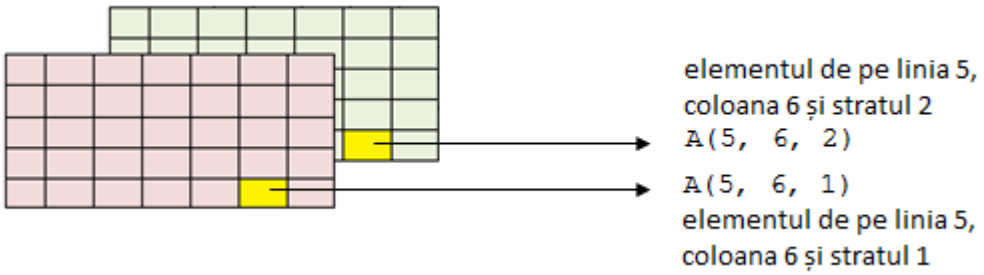
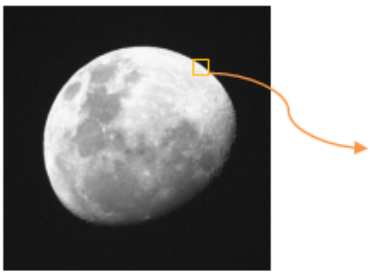


Figura 5.1. Exemplu de matrice cu 5 linii, 7 coloane și 2 straturi

Observație: în această carte, atunci când se vorbește despre *matrice*, aceasta are două dimensiuni (*m*-linii și *n*-coloane, $m > 1$ și $n > 1$). Când se va lucra cu o matrice cu mai multe straturi, se va specifica că este vorba de *matrice multistrat* sau matrice tridimensională (*m*-linii, *n*-coloane și *p* - straturi, $m > 1$, $n > 1$, $p > 1$).

Exemple de utilizare a vectorilor, matricelor și matricelor tridimensionale

- conținutul unei imagini grayscale digitale este o matrice cu aceeași dimensiune cu cea a imaginii (numărul de linii din matrice coincide cu numărul de pixeli pe verticală din imagine și numărul de coloane din matrice corespunde cu numărul de pixeli pe orizontală din imagine). De exemplu, o imagine *grayscale* de 200 x 300 pixeli nu este altceva decât o matrice cu 200 de linii și 300 de coloane; fiecare element al matricei reprezintă intensitatea unui pixel.



26	27	24	25	25	25	20	24	24	24
29	27	26	25	28	24	24	25	25	25
94	34	28	26	27	23	25	24	22	24
237	171	59	29	27	25	25	25	25	24
246	245	220	131	41	28	26	26	26	27
239	243	247	243	187	67	29	27	26	24
239	241	245	246	246	217	98	31	29	26
240	239	239	241	244	246	233	141	39	27
236	238	240	236	240	241	246	237	168	47
235	233	235	232	234	235	240	245	241	182

Figura 5.2. Reprezentarea matriceală a unei imagini digitale grayscale

- conținutul unei imagini digitale color în format RGB reprezintă o matrice tridimensională cu 3 straturi, fiecare strat fiind o matrice cu aceeași dimensiune cu cea a imaginii.

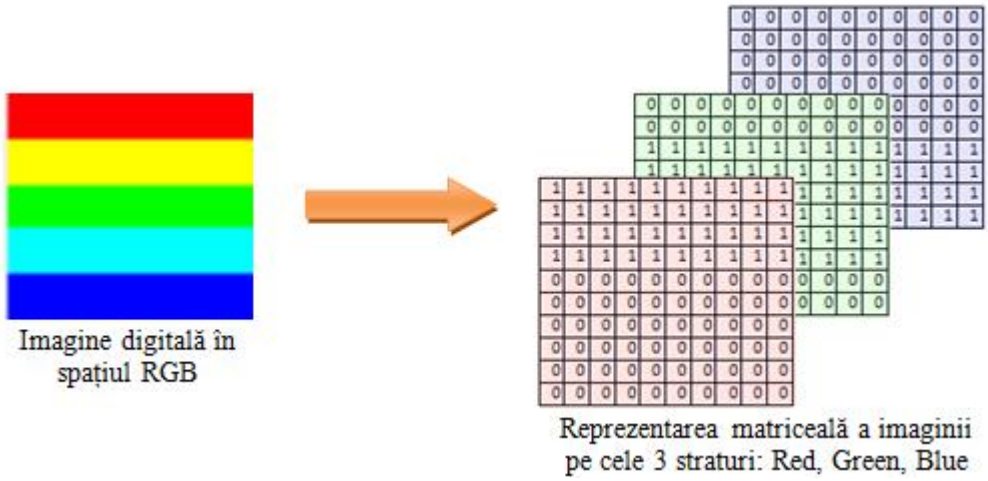


Figura 5.3. Reprezentarea matriceală a unei imagini digitale color

- conținutul unui fișier audio *mono-channel* reprezintă un vector coloană cu un număr de elemente egal cu numărul de eșantioane din semnal.
- conținutul unui fișier audio *dual-channel* reprezintă o matrice cu două coloane și un număr de linii egal cu numărul de eșantioane din fiecare canal al semnalului.

Definirea unei matrice

Pentru a inițializa o matrice în MATLAB trebuie să se respecte următorii pași:

- elementele de pe linii se separă prin *spațiu* sau *virgulă*
- pentru a separa liniile se folosește *punct și virgulă*
- elementele matricei se scriu între *paranteze pătrate*

- Să se genereze în MATLAB matricea $A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$.

```
>> A = [8 1 6; 3 5 7; 4 9 2]
A =

     8     1     6
     3     5     7
     4     9     2
```

Se poate observa că variabila `A` a fost salvată în *Workspace*.

Accesarea unui element dintr-o matrice

Accesarea unui element ce se află pe linia `i` și coloana `j` dintr-o matrice se poate face prin comanda `A(i,j)`.

Atenție! În MATLAB indexarea începe de la 1. Primul element al matricei `A` are coordonatele `A(1,1)` și se află în colțul din stânga sus al matricei.

- Fie matricea $A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$.

- Să se afișeze elementul de pe linia 2 și coloana 3
- Să se înlocuiască elementul de pe linia 2 și coloana 1 cu valoarea -10

```
>> A = [8 1 6; 3 5 7; 4 9 2]
% afisarea elementului de pe linia 2 si coloana 3
>> A(2,3)
ans =
     7
% inlocuirea elementului de pe linia 2 și coloana 1 cu valoarea -10
>> A(2,1) = -10
A =
     8     1     6
   -10     5     7
     4     9     2
```

Se poate realiza și o indexare liniară a elementelor dintr-o matrice prin parcurgerea pe coloane a matricei. De exemplu, pentru o matrice de dimensiune L `x` C, elementul `A(i,j)` poate fi selectat și scriind `A((j-1)·L+i)`. De exemplu, pentru o matrice de dimensiune 2 x 3, `A(1,2) = A(3)`.

- Fie matricea $A = \begin{bmatrix} 2 & -1 & 0 \\ 7 & 3 & 5 \end{bmatrix}$.
- Cu cât este egal `A(3)`?
- Cu cât este egal `A(2) + A(4)`?

```
>> A = [2 -1 0; 7 3 5]
A =
     2    -1     0
     7     3     5

>> A(3)
ans =
    -1

>> A(2) + A(4)
ans =
    10
```

5.1 Operații matematice cu matrice

În MATLAB se pot realiza două tipuri de operații cu matrice: operații care respectă regulile algebrei liniare și operații ce se realizează element cu element (pentru a specifica faptul că operația se realizează element cu element, se adaugă semnul *punct* înaintea operatorului).

Tabel 5.1. Cele mai întâlnite operații cu matrice în MATLAB

Operatori		Observații
+	Adunare	`A + B` (`A` și `B` au aceleași dimensiuni)
-	Scădere	`A - B` (`A` și `B` au aceleași dimensiuni)
*	Înmulțire	`A * B` (numărul de coloane din `A` = numărul de linii din `B`)
/	Împărțire	Pentru matrice pătrate, `A/B = A*inv(B)`
\	Împărțire la stânga	Pentru matrice pătrate, `A\B = inv(A)*B`
^	Ridicare la putere	Pentru matrice pătrate, `A^n = A * A ... * A` (de `n` ori)
.*	Înmulțire element cu element	`A.*B` (`A` și `B` au aceleași dimensiuni) `A(i,j)` se înmulțește cu `B(i,j)`
./	Împărțire element cu element	`A./B` (`A` și `B` au aceleași dimensiuni) `A(i,j)` se împarte la `B(i,j)`
.\	Împărțire element cu element la stanga	`A.\B = B./A` (`A` și `B` au aceleași dimensiuni)
.^	Ridicare la putere element cu element	`A.^n` (se ridică fiecare element din `A` la puterea `n`)

- Exemple de operații cu matrice în MATLAB.

```
A = [1  2; 1 -1];
B = [4 -1; 4 -2];
% Operatii ce respecta regulile algebrei liniare
disp('A + B = '); disp(A + B)
disp('A - B = '); disp(A - B)
disp('A * B = '); disp(A * B)
disp('A / B = '); disp(A / B)
disp('A \ B = '); disp(A \ B)
disp('A^2 = ');   disp(A ^ 2)

% Operatii element cu element
disp('A .* B = '); disp(A .* B)
disp('A ./ B = '); disp(A ./ B)
disp('A .\ B = '); disp(A .\ B)
```

Rezultatele afișate în *Command Window* în urma rulării codului de mai sus vor fi:

```
A + B =
     5     1
     5    -3

A - B =
    -3     3
    -3     1

A * B =
    12    -5
     0     1

A / B =
    2.5000   -2.2500
   -0.5000    0.7500

A \ B =
    4.0000   -1.6667
         0    0.3333

A^2 =
     3     0
     0     3

A .* B =
     4    -2
     4     2

A ./ B =
    0.2500   -2.0000
    0.2500    0.5000
```

Observații:

- prin adunarea/scăderea unui scalar cu o matrice se adună/scade scalarul cu fiecare element al matricei.
- prin înmulțirea/împărțirea unui scalar cu o matrice se înmulțește/împarte scalarul cu fiecare element al matricei.

Să se adune și să se înmulțească o matrice `A` cu valoarea 3.

```
>> A = [1 -3; 2 0]
A =
     1    -3
     2     0

>> 3 + A        % adunarea unei matrice cu un scalar
ans =
     4     0
     5     3

>> 3*A          % inmultirea unei matrice cu un scalar
ans =
     3    -9
     6     0
```

Concatenarea matricelor

Concatenarea matricelor reprezintă procesul de alăturare a mai multor matrice, rezultatul fiind o nouă matrice. Două sau mai multe matrice se pot concatena pe orizontală sau pe verticală.

Concatenarea pe orizontală: `C = [A, B]`

Având două matrice `A` și `B` de dimensiuni `m` x `n` respectiv m x `p`, prin concatenarea lor pe orizontală se obține matricea `C` de dimensiune `m` x (n + `p`).

```
>> A = [1 2 0; 2 3 1]
A =
     1     2     0
     2     3     1

>> B = [0 -1; 1 2]
B =
     0    -1
     1     2

>> C = [A, B]  % concatenare pe orizontala
C =
     1     2     0     0    -1
     2     3     1     1     2
```

Concatenarea pe verticală: **`C = [A; B]`**

Având două matrice `A` și `B` de dimensiuni `m` x n respectiv `p` x `n` , prin concatenarea lor pe verticală se obține matricea `C` de dimensiune (`m+p`) x n.

```
>> A = [1 2; 2 3; 0 1]
A =
     1     2
     2     3
     0     1
>> B = [0 -1; 1 2]
B =
     0    -1
     1     2
>> C = [A; B]  % concatenare pe verticala
C =
     1     2
     2     3
     0     1
     0    -1
     1     2
```

Concatenare folosind funcția `cat`

O altă modalitate de a concatena matrice este folosind funcția `cat`.

Sintaxă: `cat(dim, A, B)` concatenează matricele `A` și `B` după dimensiunea `dim`. (`dim` = 1➔ concatenare pe verticală, `dim` = 2 ➔ concatenare pe orizontală).

Observație: Avantajul funcției `cat` este acela că poate concatena valorile din interiorul unei structuri sau dintr-o variabilă de tip `cell` (pentru o mai bună înțelegere a se vedea și *5.7 Operatorul două puncte*).

```
>> val_cell = {[1 1; 2 0; 3 2]; [1 1; 2 0]};
>> valori_concatenate = cat(1, val_cell{:})
valori_concatenate =
     1     1
     2     0
     3     2
     1     1
     1     0
```

Dacă variabila `val_cell` este parametru al unei structuri, atunci concatenarea se poate face astfel:

```
val_struct = struct('val', val_cell);
val_struc_concatenate = cat(1, val_struct.val)
val_struc_concatenate =
     1     1
     2     0
     3     2
     1     1
     2     0
```

5.2 Operații matematice asupra matricelor

MATLAB-ul este optimizat pentru calcul matriceal, operațiile cu matrice și asupra matricelor fiind foarte ușor de realizat. În continuare se vor prezenta câteva dintre funcțiile des utilizate în prelucrarea elementelor unei matrice: însumarea și înmulțirea elementelor dintr-o matrice, valoarea minimă și valoarea maximă dintr-o matrice, media elementelor, sortarea elementelor, găsirea unui anumit element dintr-o matrice, extragerea elementelor de pe diagonala principală și aflarea dimensiunii unei matrice.

Funcția `sum(A, DIM)` este folosită pentru a însuma elementele unei matrice `A` pe dimensiunea `DIM`. Parametrul `DIM` poate fi `1` (dacă se face suma pe fiecare coloană) sau `2` (dacă se face suma pe fiecare linie). Dacă este omis parametrul `DIM`, se face suma pe coloane, ca și când `DIM` = 1.

- **`sum(A,1)`** va avea ca rezultat un vector linie ce conține suma pe coloane din matricea `A`.
- **`sum(A,2)`** va avea ca rezultat un vector coloană ce conține suma pe linii din matricea `A`.

Observație: **`sum(sum(A))`** va avea ca rezultat suma tuturor elementelor din matricea `A;` același rezultat se obține și dacă se folosește comanda `sum(A(:))` (a se vedea *5.7. Operatorul două puncte*).

```
>>  A =[1 2 0; 2 4 1]
A =
     1     2     0
     2     4     1
% suma elementelor pe coloane; alternativ sum(A)
>> sum(A,1)
ans =
     3     6     1
% suma elementelor pe linii
>> sum(A,2)
ans =
     3
     7
% suma tuturor elementelor din A; alternativ sum(A(:))
>> sum(sum(A))
ans =
    10
```

Funcția `prod(A, DIM)` este folosită pentru a înmulți elementele unei matrice. Modul de utilizare al funcției `prod` este asemănător cu cel al funcției **`sum`**.

Funcția `min(A,[],DIM)` este folosită pentru a determina valoarea minimă dintr-o matrice `A` pe dimensiunea `DIM`. Parametrul `DIM` poate fi `1` (dacă se caută minimul pe fiecare coloană) sau `2` (dacă se caută minimul pe fiecare linie).
Observație: **`min(min(A))`** va avea ca rezultat valoarea minimă din întreaga matrice `A;` alternativ se poate folosi și `min(A(:))`.

```
>> A = [1 3 -1; 2 0 2]
A =
     1     3    -1
     2     0     2

>> min(A,[],1)       % valoarea minima de pe fiecare coloana
ans =
     1     0    -1

>> min(A,[],2)       % valoarea minima de pe fiecare linie
ans =
    -1
     0

>> min(min(A))        % valoarea minima a tuturor elementelor din A
ans =
     -1
```

Dacă funcția `min` are doi parametri de ieșire:

`[valoare_min, poz_min] = min(A,[],DIM)`

atunci primul parametru va conține valoarea minimă găsită iar al doilea parametru va conține poziția minimului găsit (indicele liniei dacă `DIM = 1`, indicele coloanei dacă `DIM = 2`).

```
>> A = [1 3 -1; 2 0 2]
>> [valoare_min, poz_min] = min(A,[],1)
valoare_min =
     1     0    -1
poz_min =
     1     2     1

>> [valoare_min, poz_min] = min(A,[],2)
valoare_min =
    -1
     0
poz_min =
     3
     2
```

Funcția `max(A,[],DIM)` este folosită pentru a determina valoarea maximă dintr-o matrice `A`. Funcția `max` se utilizează în mod asemănător funcției **`min`**.

Funcția `mean(A, DIM)` este folosită pentru a determina valoarea medie dintr-o matrice `A` pe dimensiunea `DIM`. Parametrul `DIM` poate fi `1` (dacă se face media pe fiecare coloană) sau `2` (dacă se face media pe fiecare linie).

Observație: **`mean(mean(A))`** va avea ca rezultat valoarea medie a elementelor din întreaga matrice `A`; se poate folosi și `mean(A(:))`

```
>> A = [1 2 0; 2 4 3]
A =
     1     2     0
     2     4     3

>> mean(A, 1)         % valoarea medie pe fiecare coloana
ans =
    1.5000    3.0000    1.5000

>> mean(A, 2)         % valoarea medie pe fiecare linie
ans =
     1
     3

>> mean(mean(A))      % valoarea medie a tuturor elementelor din A
ans =
     2
```

Funcția `size(A, DIM)` se folosește pentru a determina numărul de elemente ale unei matrice `A` pe dimensiunea `DIM`. Parametrul `DIM` poate fi `1` (dacă se dorește determinarea numărului de linii) sau `2` (dacă se dorește determinarea numărului de coloane).

Observație: **`[m,n] = size(A)`** va avea ca rezultat salvarea numărului de linii în variabila `m` și salvarea numărului de coloane în variabila `n`.

```
>> A = [5 7 1; 2 4 3]
A =
     5     7     1
     2     4     3
>> [m, n] = size(A) % m = numar linii; n = numar coloane
m =
     2
n =
     3
>> m = size(A,1)     % pentru a afla numarul de linii
m =
     2
>> n = size(A,2)     % pentru a afla numarul de coloane
n =
     3
```

Funcția `sort(A,DIM,MOD)` este folosită pentru a sorta elementele unei matrice.

- **`DIM`** poate fi `1` (sortarea se face pe coloane) sau `2` (sortarea se face pe linii).
- **`MOD`** poate fi `'ascend'` pentru sortare în ordine crescătoare sau `'descend'` pentru sortare în ordine descrescătoare.

```
>> A = [5 7 1; 2 4 3]
A =
     5     7     1
     2     4     3

% sorteaza in ordine crescatoare elementele de pe coloane
>> sort(A,1,'ascend')
ans =
     2     4     1
     5     7     3

% sorteaza in ordine descrescatoare elementele de pe coloane
>> sort(A,1,'descend')
ans =
     5     7     3
     2     4     1

% sorteaza in ordine crescatoare elementele de pe linii
>> sort(A,2,'ascend')
ans =
     1     5     7
     2     3     4

% sorteaza in ordine descrescatoare elementele de pe linii
>> sort(A,2,'descend')
ans =
     7     5     1
     4     3     2
```

Funcția `diag(A)`, este folosită pentru a extrage elementele de pe diagonala principală dintr-o matrice pătratică.

```
>> A = [1 2 3; 2 3 0; 1 0 4]
A =
     1     2     3
     2     3     0
     1     0     4
>> diag(A) % elementele de pe diagonala principala
ans =
     1
     3
     4
>> sum(diag(A)) % suma elementelor de pe diagonala principala
ans =
     8
```

Funcția `sortrows(A, COL)` rearanjează liniile din matricea `A` astfel încât elementele din coloana `COL` să fie sortate**.** Dacă `COL` are valoare pozitivă atunci sortarea se face în ordine crescătoare, dacă `COL` are valoare negativă atunci sortarea se face în ordine descrescătoare.

```
A = [10 1 3 -3 2; 8 3 7 0 -2; 12 -1 3 2 5]
A =
    10     1     3    -3     2
     8     3     7     0    -2
    12    -1     3     2     5

% sortarea se face crescator dupa coloana 2
>> sortrows(A,2)
ans =
    12    -1     3     2     5
    10     1     3    -3     2
     8     3     7     0    -2

% sortarea se face descrescator dupa coloana 2
>> sortrows(A,-2)
ans =
     8     3     7     0    -2
    10     1     3    -3     2
    12    -1     3     2     5
```

Observație: dacă se dorește rearanjarea coloanelor din matricea `A` astfel încât elementele din linia `ROW` să fie sortate, se poate folosi următorul algoritm: se transpune matricea `A`, se sortează după coloana `ROW` și matricea rezultată se transpune: (`sortrows(A.', ROW)).'`

```
A = [10 1 3 -3 2; 8 3 7 0 -2; 12 -1 3 2 5]
A =
    10     1     3    -3     2
     8     3     7     0    -2
    12    -1     3     2     5
% sortarea se face crescator dupa linia 2
>> (sortrows(A', 2))'
ans =
     2    -3     1     3    10
    -2     0     3     7     8
     5     2    -1     3    12
% sortarea se face descrescator dupa linia 2
>> (sortrows(A.', -2)).'
ans =
    10     3     1    -3     2
     8     7     3     0    -2
    12     3    -1     2     5
```

Observație: începând cu versiunea de MATLAB R2015b există funcția `sortcols`.

Funcția `find(condiție)` permite găsirea indicilor elementelor dintr-o matrice ce respectă o anumită condiție.

Observație: în acest caz indexarea este una liniară și se obține prin parcurgerea pe coloane a matricei. De exemplu, pentru o matrice `A` de dimensiune *L* `x` *C*, elementul `A(i,j)` are indicele `(j-1)·L+i`; pentru o matrice de 2 linii și 3 coloane: a_{11} are indice 1, a_{21} are indice 2, a_{12} are indice 3, ..., a_{23} are indice 6.

```
>> A = [2 5 -1; 0 8 3]
A =
     2     5    -1
     0     8     3

% gasirea indicilor elementelor de valoare >=5
>> find(A >= 5)
ans =
     3
     4
```

Folosirea funcției `find` cu doi parametri de ieșire:

`[coord_linie, coord_coloană] = find(condiție)`

permite găsirea într-o matrice a coordonatelor elementelor ce respectă o anumită condiție.

```
>> A = [10 1 3 -3 2; 8 3 7 0 -2; 12 -1 3 2 5]
A =
    10     1     3    -3     2
     8     3     7     0    -2
    12    -1     3     2     5

% gasirea coordonatelor elementelor de valoare 0
>> [coord_linie, coord_coloana] = find(A == 0)
coord_linie =
     2
coord_coloana =
     4

% gasirea coordonatelor elementelor >= 7 si < 10
>> [coord_linie, coord_coloana] = find((A >= 7)&(A < 10))
coord_linie =
     2
     2
coord_coloana =
     1
     3

% gasirea numarului de elemente egale cu 3
>> size(find(A==3), 1)
ans =
     3
```

5.3 Operații cu matrice specifice algebrei liniare

Fie o matrice $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Asupra ei se pot efectua mai multe operații matematice specifice algebrei liniare dintre care amintim:

- determinantul: $\det(A) = ad - bc$
- inversa: $inv(A) = A^{-1} = \frac{1}{det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$
- transpusa: $A^T = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$
- valorile proprii ale lui *A*: sunt valorile λ soluții ale ecuației $\det(A - \lambda \cdot I) = 0$.
- vectorii proprii în raport cu *A*: vectorii *x* soluții ale ecuației $A \cdot x = \lambda \cdot x$.

Funcția `det(A)` este folosită pentru a afla determinantul unei matrice `A`.

```
>> A = [5 2 1 ; 7 1 2; 0 2 1]
A =
     5     2     1
     7     1     2
     0     2     1
>> det(A) % determinant
ans =
     -15
```

Funcția `inv(A)` se folosește pentru a afla inversa unei matrice `A`.

```
>> A = [2 5;1 3]
A =
     2     5
     1     3

>> inv(A) % alternativ A^(-1)
ans =
     3    -5
    -1     2
```

Funcția `eig(A)` având doi parametri de ieșire ca în sintaxa de mai jos:

`[vectori_proprii, valori_proprii] = eig(A)`

va salva vectorii proprii ai matricei `A` în variabila `vectori_proprii` și valorile proprii ale matricei `A` în variabila `valori_proprii`.

Observații: variabila `vectori_proprii` este o matrice ce conține pe fiecare coloană câte un vector propriu. Variabila `valori_proprii` este o matrice ce conține pe diagonală valorile proprii.

```
>> A = [2 0; 0 -2];
% calcul vectori proprii si valori proprii
>> [vectori_proprii, valori_proprii] = eig(A)
vectori_proprii =
     0     1
     1     0
valori_proprii =
    -2     0
     0     2

% salvarea valorilor proprii intr-un vector
>> vector_ValProprii = diag(valori_proprii)
vector_ValProprii =
    -2
     2
```

Transpusa unei matrice `A` se determină folosind sintaxa **`A.'`**

Transpus conjugata unei matrice `A` se determină folosind sintaxa **`A'`**

```
>> A = [2 5;1 3]  % matrice cu numere reale
A =
     2     5
     1     3

>> A.'            % transpusa unei matrice cu numere reale
ans =
     2     1
     5     3

>> A'             % transpus conjugata unei matrice cu numere reale
ans =
     2     1
     5     3
% !!! Pentru numere reale se observa ca A' = A.'

>> A = [1+2*i 3; 2+3*i i] % matrice cu numere complexe
A =
   1.0000 + 2.0000i   3.0000
   2.0000 + 3.0000i        0 + 1.0000i

>> A.'   % transpusa unei matrice cu numere complexe
ans =
   1.0000 + 2.0000i   2.0000 + 3.0000i
   3.0000                  0 + 1.0000i

>> A'    % transpus conjugata unei matrice cu numere complexe
ans =
   1.0000 - 2.0000i   2.0000 - 3.0000i
   3.0000                  0 - 1.0000i
% !!! Pentru numere complexe se observa ca A.' difera de A'
```

5.4 Generarea diverselor matrice particulare

De multe ori, în dezvoltarea aplicațiilor precum și în testarea acestora este utilă folosirea matricelor ce conțin valori particulare; de exemplu: o matrice nulă, o matrice ce conține doar valoarea `n` sau matricea unitate.

Funcția `zeros(m, n)` generează o matrice nulă de dimensiune `m x n`.

```
% generarea unei matrice nule de dimensiune 2 x 3
>> B = zeros(2,3)
B =
     0     0     0
     0     0     0
```

- Să se adauge după ultima coloană a unei matrice două coloane de zerouri.

```
>> A = [2 3 -5;-1 2 1]
A =
     2     3    -5
    -1     2     1
>> B = [A, zeros(size(A,1),2)]
B =
     2     3    -5     0     0
    -1     2     1     0     0
```

Observație: dacă se dorește generarea unei matrice nule de dimensiune `m x m`, se poate folosi sintaxa `zeros(m)`.

Funcția `ones(m, n)` generează o matrice de dimensiune `m x n` conținând doar valoarea 1.

```
% generarea unei matrice de dimensiune 2 x 3 continand doar 1
>> A = ones(2,3)
A =
     1     1     1
     1     1     1
% generarea unei matrice de dimensiune 2 x 3 continand doar 5
>> A = 5*ones(2,3)
A =
     5     5     5
     5     5     5
```

Funcția `eye(m)` generează matricea unitate de dimensiune `m x m`.

```
% generarea matricei unitate de dimensiune 3 x 3
>> C = eye(3)
C =
     1     0     0
     0     1     0
     0     0     1
```

Generarea matricelor cu valori pseudorandom

Pentru generarea valorilor pseudorandom sunt disponibile mai multe funcții ce generează valori având diverse distribuții. Dintre aceste funcții amintim: `rand` și `randi` (ambele având *distribuție uniformă*) și `normrnd` cu *distribuție normală*.

Funcția `rand(m, n)` generează pseudorandom o matrice de dimensiune `m x n` cu valori de tip *double*, uniform distribuite în intervalul (0, 1).

- Să se genereze o matrice de dimensiune 2 x 3 ce conține numere generate pseudorandom în intervalul (0, 1) și distribuite uniform.

```
>> D = rand(2,3)
D =
    0.8147    0.1270    0.6324
    0.9058    0.9134    0.0975
```

Dacă se dorește să se genereze valori pseudorandom în intervalul (`a, b`), se poate folosi sintaxa:

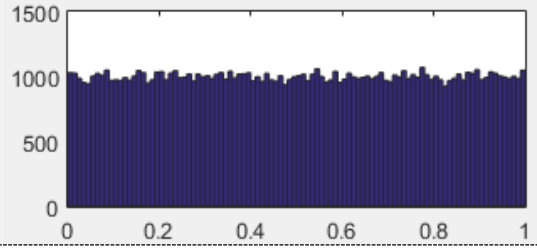
```
valori = a + (b - a) * rand(m, n)
```

- Să se genereze pseudorandom o matrice de dimensiune 2 x 5 cu valori uniform distribuite în intervalul (20, 75).

```
>> a = 20;
>> b = 75;
>> valori = a + (b - a)*rand(2, 5)
valori =
   25.3647   50.0785   73.0689   73.3826   46.6957
   35.3174   72.6629   28.6687   72.6442   64.0154
```

- Să se genereze pseudorandom o matrice cu 5 linii și 20000 de coloane care să conțină elemente în intervalul (0, 1) cu distribuție uniformă. Să se reprezinte histograma tuturor elementelor.

```
>> X = rand(5, 20000);
>> hist(X(:), 100)
```



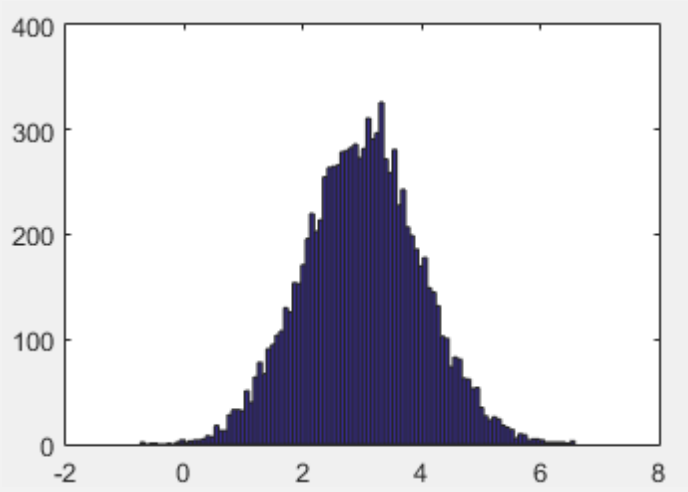
Din histograma de mai sus se poate observa distribuția uniformă a valorilor

Funcția `normrnd(m, sigma, [nr_linii, nr_coloane])`

Generează pseudorandom o matrice de dimensiune **`[nr_linii, nr_coloane]`** de numere având o distribuție gaussiană (normală) de medie `m` și dispersie `sigma`.

- Să se genereze pseudorandom o matrice cu 5 linii și 2000 de elemente având media 3 și dispersia 1. Să se reprezinte histograma tuturor elementelor din matrice.

```
>> X = normrnd(3, 1, [5, 2000]);
>> hist(X(:),100)
```



Din histograma de mai sus se poate observa distribuția gaussiană a valorilor.

Funcția `randi([val_min, val_max], [nr_linii, nr_coloane])`

Generează pseudorandom o matrice de dimensiune **`[nr_linii, nr_coloane]`** de numere naturale distribuite uniform într-un interval **`[val_min, val_max]`**.

- Să se genereze o matrice de dimensiune 2 x 3 ce conține numere naturale generate pseudorandom în intervalul [0, 10].

```
>> val_rand = randi([0, 10], [2,3])
val_rand =
     0     3     1
     2     0     8
```

Atenție! De fiecare dată când se generează numere pseudorandom se vor obține alte valori. Dacă se dorește ca în urma mai multor generări pseudorandom să se obțină același rezultat, se folosește funcția `rng`.

Funcția `rng('default')` setează valorile generatorului pseudorandom pe valorile default, astfel putând ca în urma mai multor generări pseudorandom să se obțină același rezultat.

- Să se genereze pseudorandom matricele `X` și `Y` de dimensiuni 2 linii și 5 coloane, conținând valori numere întregi în intervalul [-10, 10] și având distribuție uniformă. Matricele `X` și `Y` să conțină aceleași valori!

```
setari = rng('default');
X = randi([-10, 10], [2, 5])
rng(setari)
Y = randi([-10, 10], [2, 5])
```

În urma rulării programului de mai sus se va afișa în *Command Window*:

```
X  =
    -4     7    -4     7   -10
     2     8    -8    -8     7

Y  =
    -4     7    -4     7   -10
     2     8    -8    -8     7
```

Atenție! Dacă programul de mai sus este rulat de mai multe ori, variabilele `X` și `Y` vor avea alte valori de la o rulare la alta (vor rămâne însă în continuare egale între ele). Dacă se dorește să se genereze aceleași valori pseudorandom în urma mai multor rulări, trebuie să se salveze parametrul `setari`.

- Să se scrie un program care de fiecare dată când este rulat va genera aceeași matrice `X` de dimensiune 2 linii și 5 coloane, conținând valori numere întregi în intervalul [-5, 5] și având distribuție uniformă.

```
% daca exista fisierul setari_default.mat
if (exist('setari_default.mat') == 2)
    % daca nu exista in Workspace variabila setari
    if (exist('setari') == 0)
        % se incarca variabila setari din fisierul setari_default
        load('setari_default', 'setari');
    end
% daca nu exista fisierul setari_default.mat
else
       % se initializeaza variabila setari cu setarile default
       setari = rng('default');
       % se salveaza variabila setari in fisierul setari_default.mat
       save('setari_default','setari');
end
rng(setari)
X = randi([-5, 5], [2, 5])
```

5.5 Particularizare pentru vectori

Vectorii sunt cazuri particulare de matrice: un vector linie este o matrice cu o singură linie, iar un vector coloană este o matrice cu o singură coloană. Din acest motiv, majoritatea operațiilor cu matrice și asupra matricelor prezentate anterior se pot aplica și vectorilor. În continuare se va prezenta însă și o sintaxă simplificată, specifică lucrului cu vectori, precum și câteva funcții dedicate lucrului cu vectori.

Generarea unui vector

Generarea unui vector linie. Sintaxă: `X = [x1, x2, x3, …, xn]`
Se scriu elementele vectorului pe o linie, între paranteze pătrate. Elementele vectorului pot fi separate de *virgule* sau *de spațiu*.

```
% generarea unui vector linie
>> X = [2, 4, -1, 3, 5]
>> X =
     2     4    -1     3     5
```

Generarea unui vector coloană.
Pentru a genera un vector coloană pot fi folosite două metode.
Sintaxă_1: `X = [x1; x2; x3; …; xn]`
Se scriu elementele vectorului pe o linie, între paranteze pătrate, separate în mod obligatoriu de *punct și virgulă*.

```
% generarea unui vector coloana
>> X = [2; 4; -1; 3]
X =
     2
     4
    -1
     3
```

Sintaxă_2*:* `X = [x1, x2, x3, …, xn].'`
În acest caz se scrie vectorul ca și când ar fi un vector linie și apoi se transpune.

```
% generarea unui vector coloana
>> X = [2 4 -1 3].'
X =
     2
     4
    -1
     3
```

Accesarea unui element dintr-un vector

Fie că este vorba de vector linie sau vector coloană, atunci când se accesează un element al unui vector `A` se folosește sintaxa `A(m)`, unde `m` este indicele elementului.

- Într-un vector linie `X` să se interschimbe elementul de pe poziția 2 cu elementul de pe poziția 5.

```
>> X = [7 4 0 2 9 3];
>> a = X(2);
>> X(2) = X(5);
>> X(5) = a;
>> X
X =
     7     9     0     2     4     3
```

Operații asupra vectorilor

Operațiile asupra matricelor prezentate anterior pot fi folosite și pentru vectori, cu observația că nu trebuie să se mai specifice dimensiunea pe care se face operația respectivă (vectorul având o singură dimensiune).

Exemplificarea modului de realizare a operațiilor asupra elementelor dintr-un vector.

```
>> X = [4 2 1 8 5];
>> sum(X)   % suma tuturor elementelor din vectorul A
ans =
    20

>> prod(X)  % produsul tuturor elementelor din vectorul A
ans =
   320

>> min(X)   % cea mai mica valoare din vectorul A
ans =
     1

>> max(X)   % cea mai mare valoare din vectorul A
ans =
     8

>> mean(X)  % valoarea medie a elementelor din A
ans =
     4

>> sort(X, 'ascend')  % sorteaza vectorul A in ordine crescatoare
ans =
     1     2     4     5     8

>> find(X < 8)
 % cauta in vectorul A pozitiile tuturor elementelor < 8
ans =
     1     2     3     5
```

Funcții dedicate lucrului cu vectori

Funcția `length(X)` determină numărul de elemente dintr-un vector `X`.
Rezultatul este echivalent cu `size(X,1)` (dacă este vector coloană) sau `size(X,2)` (dacă este vector linie).

```
>> X = [2 4 -1 3 5];
% determinarea numarului de elemente din vectorul linie X
>> length(X)
ans =
     5
```

- Să se afle numărul elementelor dintr-un vector `X` mai mari decât 5.

```
>> X = [2 6 -1 3 5 9 15 3];
>> length(find(X > 5))
ans =
     3
```

Funcția `linspace(prima_val, ultima_val, N)` generează `N` valori echidistante în intervalul `[prima_val, ultima_val]`.

- Să se genereze un vector `X` cu 5 valori în intervalul [20, 80]. Între oricare două valori consecutive diferența să fie aceeași.

```
>> X = linspace(20, 80, 5)
X =
    20    35    50    65    80
```

Funcția `diag(X)` creează o matrice diagonală care are pe diagonala principală elementele din vectorul `X`. *Observație:* dacă parametrul funcției `diag` este o matrice, atunci `diag(X)` extrage elementele de pe diagonala principală a matricei.

- Fie valorile proprii $\lambda_1 = 1, \lambda_2 = -1, \lambda_3 = 2$. Să se genereze matricea diagonală care să conțină aceste valori.

```
>> val_proprii = [1, -1, 2];
>> D = diag(val_proprii)
D =
     1     0     0
     0    -1     0
     0     0     2
```

5.6 Generalizare pentru matrice tridimensionale

Matricele tridimensionale sunt o generalizare a matricelor (bidimensionale); o matrice tridimensională are mai multe straturi, fiecare strat fiind o matrice.

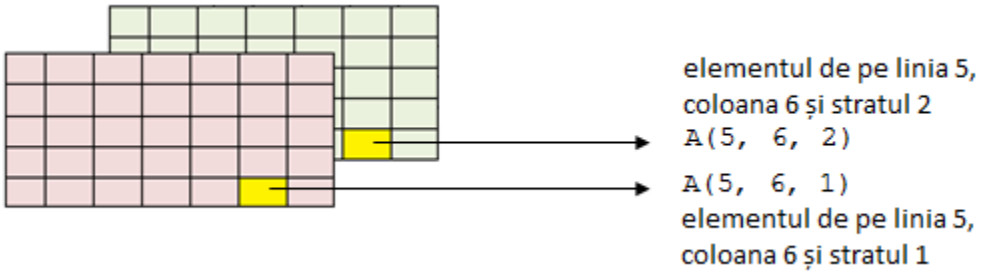


Figura 5.4. Exemplu de matrice cu 5 linii, 7 coloane și 2 straturi

Multe dintre funcțiile prezentate anterior la lucrul cu matrice se vor folosi și în continuare, prin generalizare de la spațiul 2D la spațiul 3D.

Operațiile cu matrice tridimensionale sunt foarte utile de exemplu în procesarea imaginilor color (RGB) digitale în care fiecare strat de culoare (Roșu, Verde, Albastru) este practic o matrice.

Definirea unei matrice tridimensionale

Pentru a defini o matrice tridimensională trebuie să se definească elementele de pe fiecare strat în parte (fiecare strat este o matrice).

Observație: `A (:, :, k)` se referă la totalitatea elementelor de pe stratul `k` din matricea tridimensională `A` (a se vedea *5.7 Operatorul două puncte*).

```
% definirea unei matrice cu 2 linii, 3 coloane si 3 straturi
>> A(:,:,1) = [1 4  0; 2 -1 3];
>> A(:,:,2) = [2 0 -3; 1  0 0];
>> A(:,:,3) = [1 2  0; 0  1 0];
>> A
A(:,:,1) =
     1     4     0
     2    -1     3

A(:,:,2) =
     2     0    -3
     1     0     0

A(:,:,3) =
     1     2     0
     0     1     0
```

Accesarea unui element într-o matrice tridimensională

Pentru a accesa un element dintr-o matrice tridimensională trebuie să se specifice indicele liniei, indicele coloanei și indicele stratului.

Sintaxă: `x = A(m,n,k)` salvează în variabila `x` valoarea din matricea tridimensională `A` de la linia `m`, coloana `n` și stratul `k`.

```
% definirea unei matrice cu 2 linii, 3 coloane si 2 straturi
>> A(:,:,1) = [1 4  0; 2 -1 3];
>> A(:,:,2) = [2 0 -3; 1  0 0];
>> A
A(:,:,1) =
     1     4     0
     2    -1     3
A(:,:,2) =
     2     0    -3
     1     0     0
% se salveaza in x elementul de la linia 2, coloana 1 si stratul 2
>> x = A(2, 1, 2)
x =
     1
```

Se poate realiza și o indexare liniară a elementelor dintr-o matrice tridimensională. Ordinea de parcurgere este: prima coloană din primul strat, a doua coloană din primul strat ș.a.m.d. până la ultima coloană din primul strat; apoi se continuă în același mod cu straturile următoare. Pentru exemplul de mai sus `A(1,3,2)= A(11)=-3.`

Operații cu matrice tridimensionale

Pentru matricele tridimensionale, operațiile de adunare, scădere și înmulțire se pot realiza numai pe straturile corespondente (fiecare strat din `A` cu fiecare strat din `B`). În continuare se vor exemplifica adunarea și înmulțirea.

Adunarea. **Sintaxă:** `A + B` (`A` și `B` trebuie să aibă aceleași dimensiuni)

```
% suma a doua matrice tridimensionale
>> A(:,:,1) = [1  4  0; 2 -1 3];
>> A(:,:,2) = [2  0 -3; 1  0 0];
>> B(:,:,1) = [1  0  0; 0  0 1];
>> B(:,:,2) = [0 -1  0;-1  0 1];
>> C = A + B
C(:,:,1) =
     2     4     0
     2    -1     4
C(:,:,2) =
     2    -1    -3
     0     0     1
```

Înmulțirea. Sintaxă `A(:,:,k)*B(:,:,k)`

Înmulțirea se poate realiza numai pe straturi (nu se pot înmulți direct două matrice tridimensionale). Stratul `k` din `A` se înmulțește cu stratul `k` din `B`. Numărul de coloane din `A` trebuie să fie egal cu numărul de linii din `B`.

```
% inmultirea a doua matrice tridimensionale
>> A(:,:,1) = [1 4  0; 2 -1 3]; A(:,:,2) = [2 0 -3; 1  0 0];
>> B(:,:,1) = [2 0 ;3 1; 0 -1]; B(:,:,2) = [1 0 ;0 1; 1 -1];
>> C(:,:,1) = A(:,:,1)*B(:,:,1);C(:,:,2) = A(:,:,2)*B(:,:,2);
>> C
C(:,:,1) =
    14     4
     1    -4
C(:,:,2) =
    -1     3
     1     0
```

Operațiile cu matrice tridimensionale se pot realiza și element cu element. În continuare se vor exemplifica înmulțirea și ridicarea la putere element cu element.

Înmulțirea element cu element. **Sintaxă:** `A .* B` (se înmulțește fiecare element din `A` cu elementul corespondent din `B`).

```
% inmultirea element cu element a doua matrice tridimensionale
>> A(:,:,1) = [1  4  0; 2 -1 3]; A(:,:,2) = [2  0 -3; 1  0 0];
>> B(:,:,1) = [1  0  0; 0  0 1]; B(:,:,2) = [0 -1  0;-1  0 1];
>> C = A .* B
C(:,:,1) =
     1     0     0
     0     0     3
C(:,:,2) =
     0     0     0
    -1     0     0
```

Ridicarea la putere element cu element. **Sintaxă:** `A .^n` (se ridică fiecare element din `A` la puterea `n`).

```
% ridicarea la puterea a 2-a element cu element
>> A(:,:,1) = [1 4  0; 2 -1 3]; A(:,:,2) = [2 0 -3; 1  0 0];
>> A.^2
ans(:,:,1) =
     1    16     0
     4     1     9
ans(:,:,2) =
     4     0     9
     1     0     0
```

Concatenarea matricelor tridimensionale

Două sau mai multe matrice tridimensionale pot fi concatenate pe orizontală, pe verticală sau pe straturi.

Concatenare pe verticală: `C = [A; B]` sau `C = cat(1,A,B)`

Fie două matrice tridimensionale `A` și `B` de dimensiuni `[m,n,p]` respectiv `[t,n,p]`. Prin concatenarea lor pe orizontală se obține o matrice de dimensiuni `[m+t,n,p]`.

```
% concatenare pe verticala
>> A(:,:,1) = [1  4  0; 2 -1 3];
>> A(:,:,2) = [2  0 -3; 1  0 0];
>> B(:,:,1) = [1  0  4];
>> B(:,:,2) = [0 -1  0];


>> A
A(:,:,1) =
     1     4     0
     2    -1     3
A(:,:,2) =
     2     0    -3
     1     0     0

>> B
B(:,:,1) =
     1     0     4
B(:,:,2) =
     0    -1     0

>> C = cat(1, A, B) sau C = [A; B]
C(:,:,1) =
     1     4     0
     2    -1     3
     1     0     4
C(:,:,2) =
     2     0    -3
     1     0     0
     0    -1     0
```

Concatenare pe orizontală: `C =[A, B]` sau `C = cat(2,A,B)`

Fie două matrice tridimensionale `A` și `B` de dimensiuni `[m,n,p]` respectiv `[m,t,p]`. Prin concatenarea lor pe orizontală se obține o matrice de dimensiuni `[m,n+t,p]`.

```
% concatenare pe orizontala
>> A(:,:,1) = [1 4  0; 2 -1 3];
>> A(:,:,2) = [2 0 -3; 1  0 0];
>> B(:,:,1) = [1  0; 0 0];
>> B(:,:,2) = [0 -1;-1 0];
>> A
```

```
A(:,:,1) =
     1     4     0
     2    -1     3
A(:,:,2) =
     2     0    -3
     1     0     0

>> B
B(:,:,1) =
     1     0
     0     0
B(:,:,2) =
     0    -1
    -1     0

>> C = [A, B]              % sau C = cat(2, A, B)
C(:,:,1) =
     1     4     0     1     0
     2    -1     3     0     0
C(:,:,2) =
     2     0    -3     0    -1
     1     0     0    -1     0
```

Concatenare pe straturi: `cat(3,A,B)`

Fie două matrice tridimensionale `A` și `B` de dimensiuni [`m,n,p`] respectiv [`m,n,t`]. Prin concatenarea lor pe straturi se obține o matrice de dimensiuni [`m,n,p+t`].

```
% concatenare pe straturi
>> A(:,:,1) = [1  4  0; 2 -1 3]; A(:,:,2) = [2  0 -3; 1  0 0];
>> B(:,:,1) = [1  0  0; 0  0 1]; B(:,:,2) = [0 -1  0;-1  0 1];
>> B(:,:,3) = [2 -1  1;-1  1 1];

>> C = cat(3,A,B)
C(:,:,1) =
     1     4     0
     2    -1     3
C(:,:,2) =
     2     0    -3
     1     0     0
C(:,:,3) =
     1     0     0
     0     0     1
C(:,:,4) =
     0    -1     0
    -1     0     1
C(:,:,5) =
     2    -1     1
    -1     1     1
```

Operații asupra matricelor tridimensionale

Pentru o matrice tridimensională `A`, comanda `B = A(:,:,k).'` salvează în matricea `B` transpusa stratului `k` din `A`. Dacă se dorește permutarea dimensiunilor, atunci se poate folosi funcția `permute`.

Funcția `permute(A, [v_ordine])`, schimbă ordinea dimensiunilor matricei tridimensionale `A` conform elementelor din vectorul `v_ordine`.
Elementele vectorului `v_ordine` pentru o matrice tridimensională sunt: 1, 2 și 3. Dacă `v_ordine = [1, 2, 3]`, atunci `A` rămâne neschimbată. Dacă `v_ordine = [2, 1, 3]`, atunci liniile din `A` devin coloane (se transpune fiecare strat).

```
% transpunerea tuturor straturilor dintr-o matrice tridimensionala
>> A(:,:,1) = [1  0  0; 0  0 1];
>> A(:,:,2) = [0 -1  0;-1  0 1];
>> A(:,:,3) = [2 -1  1;-1  1 1];
>> A
A(:,:,1) =
     1     0     0
     0     0     1
A(:,:,2) =
     0    -1     0
    -1     0     1
A(:,:,3) =
     2    -1     1
    -1     1     1

>> B = permute(A,[2,1,3])
B(:,:,1) =
     1     0
     0     0
     0     1
B(:,:,2) =
     0    -1
    -1     0
     0     1
B(:,:,3) =
     2    -1
    -1     1
     1     1
```

Majoritatea funcțiilor utilizate pentru realizarea operațiilor asupra matricelor pot fi folosite și în contextul matricelor tridimensionale. În continuare se va exemplifica folosirea acestor funcții pentru cea de-a treia dimensiune (pe straturi).

- **`sum(A, 3)`** realizează suma pe straturi a elementelor dintr-o matrice 3D. Rezultatul este o matrice 2D cu același număr de linii și de coloane cu cel al matricei 3D. Pentru a obține suma tuturor elementelor, este nevoie de sintaxa `sum(sum(sum(A)))` sau `sum(A(:))`.

```
>> A(:,:,1) = [1  0  0; 0  0 1];
>> A(:,:,2) = [0 -1  0;-1  0 1];
>> A(:,:,3) = [2 -1  1;-1  1 1];
>> B = sum(A, 3) % suma pe straturi intr-o matrice tridimensionala
B =
     3    -2     1
    -2     1     3

>> C = sum(sum(sum(A))) % suma tuturor elementelor
C =
     4
```

- **`prod(A,3)`** realizează produsul pe straturi a elementelor dintr-o matrice 3D. Pentru a obține produsul tuturor elementelor, este nevoie de sintaxa `prod(prod(prod(A)))` sau `prod(A(:))`.

```
% produsul pe straturi intr-o matrice tridimensionala
>> A(:,:,1) = [1 -1  0; 1  0 1];
>> A(:,:,2) = [0 -1  0;-1  0 1];
>> A(:,:,3) = [2 -1  1;-1  1 2];
>> B = prod(A, 3)
B =
     0    -1     0
     1     0     2
```

- **`min(A,[],3)`** și **`max(A,[],3)`** vor avea ca rezultat câte o matrice cu același număr de linii și coloane ca și matricea tridimensională `A`; acestea conțin valorile minime, respectiv maxime, pe straturi.

```
% calculul maximului pe straturi intr-o matrice tridimensionala
>> A(:,:,1) = [1 -1  0; 1  0 1];
>> A(:,:,2) = [0 -1  0;-1  0 1];
>> A(:,:,3) = [2 -1  1;-1  1 2];
>> B = max(A,[],3)
B =
     2    -1     1
     1     1     2
```

- **`mean(A,3)`** calculează valorile medii ale elementelor corespondente pe straturi.

```
% calculul mediei pe straturi intr-o matrice tridimensionala
>> A(:,:,1) = [2 -1  0; 1  0 1];
>> A(:,:,2) = [0 -1  0;-3  0 1];
>> B = mean(A,3)
B =
     1    -1     0
    -1     0     1
```

- funcția **`sort(A, DIM, MOD)`** acceptă pentru parametrul `DIM` și valoarea 3, caz în care se realizează sortarea pe straturi.

```
% sortarea pe straturi intr-o matrice tridimensionala
>> A(:,:,1) = [2 -3  4; 1  2  4];
>> A(:,:,2) = [0 -1  0;-3  0  1];
>> B = sort(A, 3, 'ascend')
B(:,:,1) =
     0    -3     0
    -3     0     1
B(:,:,2) =
     2    -1     4
     1     2     4
```

- funcția **`size(A)`** acceptă 3 parametri de ieșire, indicând numărul de linii, numărul de coloane și numărul de straturi (în această ordine).

```
% determinarea dimensiunilor intr-o matrice tridimensionala
>> A(:,:,1) = [1 -1  0; 1  0 1];
>> A(:,:,2) = [0 -1  0;-1  0 1];
>> A(:,:,3) = [2 -1  1;-1  1 2];
>> A(:,:,4) = [2 -1  0;-1  0 2];
>> [m,n,p] = size(A)
m =
     2
n =
     3
p =
     4
```

Funcțiile `zeros(m, n, p)` și `ones(m, n, p)` generează matrice tridimensionale de zerouri și de unități.
Funcțiile `diag` și `eye` nu sunt definite pentru matrice tridimensionale.

5.7 Operatorul *două puncte* (:)

Acest operator facilitează foarte mult lucrul cu matrice (în speță lucrul cu vectori și matrice tridimensionale) și poate fi folosit în mai multe contexte.

Caz 1: `x = val_start : pas : val_stop`
Se generează un vector `x`, cu valori între `val_start` și `val_stop`, din `pas` în `pas`. Dacă nu se scrie variabila `pas`, se consideră implicit `pas = 1`.

- Generarea unui vector `x` cu valori între 1 și 8.

```
>> x = 1 : 8
x =
     1     2     3     4     5     6     7     8
```

- Generarea unui vector `x` cu valori între 1 și 100, din 10 în 10.

```
>> x = 1 : 10 : 100
x =
     1    11    21    31    41    51    61    71    81    91
```

Caz 2: `B = A (x : y , m : n)`
Se salvează în matricea `B` elementele din matricea `A` cuprinse între liniile `x` și `y` și între coloanele `m` și `n`.

- Să se salveze în matricea `B` elementele din matricea `A` cuprinse între liniile 2 și 3 și între coloanele 3 și 5.

```
>> A = [2 9 0 -1 5 8; 0 -3 2 7 4 3; 9 0 3 8 -1 6]
A =
     2     9     0    -1     5     8
     0    -3     2     7     4     3
     9     0     3     8    -1     6

>> B = A(2:3, 3:5)
B =
     2     7     4
     3     8    -1
```

În cazul în care se folosesc toate elementele de pe o dimensiune (toate liniile, toate coloanele sau toate straturile) se poate scrie doar "`:`" pentru dimensiunea respectivă.

- `A(:,n)` ➔ coloana `n`
- `A(m,:)` ➔ linia `m`
- `A(:,:,k)` ➔ stratul `k`
- `A(end,:)` ➔ ultima linie
- `A(:,end)` ➔ ultima coloană
- `A(:,:,end)` ➔ ultimul strat

Să se salveze în matricea `B` coloana 2 din matricea `A`.

```
>> A = [ 2 3 4 0 1; -1 0  3 6 8]
A =
     2     3     4     0     1
    -1     0     3     6     8

>> B = A( : , 2)
B =
     3
     0
```

Să se însumeze toate elementele de pe ultima linie din matricea `A`.

```
>> A = [ 2 3 4 0 1; -1 0  3 6 8]
A =
     2     3     4     0     1
    -1     0     3     6     8

>> sum(A(end,:))
ans =
    16
```

Să se salveze în matricea `B` linia 2 și coloanele 3 și 5 din matricea `A`.

```
>> A = [ 2 3 4 0 1; -1 0  3 6 8]
A =
     2     3     4     0     1
    -1     0     3     6     8
>> B = A(2, [3, 5])
B =
     3     8
```

- Să se salveze în matricea `B` al doilea strat din matricea `A`.

```
>> A(:,:,1) = [1 2 0; 5 3 2];
>> A(:,:,2) = [3 0 1; 8 1 0];
>> A(:,:,3) = [1 0 0; 0 1 0];
>> B = A(:,:,2)
B =
     3     0     1
     8     1     0
```

Caz 3. `B = A(:)` are ca efect scrierea elementelor din `A` într-un vector coloană `B`. Dacă `A` este un vector linie atunci `B` va fi un vector coloană. Același lucru se obține folosind operația `B = A.'`

- Să se transforme un vector linie într-un vector coloană folosind *două puncte*.

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

>> B = A(:)% transformarea unui vector linie intr-un vector coloana
B =
     1
     2
     3
```

Dacă `A` este o matrice cu `m` linii și `n` coloane, atunci `B` va fi un vector coloană cu `m·n` elemente (concatenarea se face pe coloane).

- Să se transforme o matrice într-un vector coloană.

```
>> A = [1 3 5; 2 4 6]
>> B = A(:)  % transformarea unei matrice intr-un vector coloana
B =
     1
     2
     3
     4
     5
     6
```

Dacă `A` este o matrice tridimensională cu `m` linii, `n` coloane și `p` straturi atunci `B` va fi un vector coloană cu `m·n·p` elemente.

Ștergerea unui element dintr-un vector sau matrice

Pentru a șterge al `n`-lea element dintr-un vector `A` se folosește sintaxa `A(n) = []`.

```
>> A = [1 2 3 4 5]
A =
     1     2     3     4     5

>> A(3) = []    % se sterge al 3-lea element
A =
     1     2     4     5
```

Pentru a șterge o coloană dintr-o matrice, se folosește sintaxa `A(:,n)=[]`, unde `n` este indicele coloanei ce se dorește a se șterge. Pentru a șterge o linie `m` dintr-o matrice se folosește sintaxa `A(m,:)=[]`.

```
>> A = [1 0 -1 3; 0 2 3 3]
A =
     1     0    -1     3
     0     2     3     3

>> A(:, 3) = [] % se sterge a 3-a coloana
A =
     1     0     3
     0     2     3
```

Pentru a șterge un strat `k` dintr-o matrice `A` tridimensională se folosește sintaxa `A(:,:,k)=[]`.

```
>> A(:,:,1) = [1 2 0; 5 3 2];
>> A(:,:,2) = [3 0 1; 8 1 0];
>> A(:,:,3) = [1 0 0; 0 1 0];
>> A(:,:,2) = []  % se sterge al 2-lea strat
A(:,:,1) =
     1     2     0
     5     3     2

A(:,:,2) =
     1     0     0
     0     1     0
```

De asemenea se pot șterge simultan mai multe elemente. Dacă se dorește să se șteargă de exemplu elementele dintr-un vector `X` de la poziția `n` până la poziția `n+k`, se folosește sintaxa: `X(n:n+k)=[]`.

6. Instrucțiuni decizionale și repetitive

În MATLAB există următoarele instrucțiuni decizionale: instrucțiunea `if` (folosită împreună cu `else` și `elseif`) și instrucțiunea `switch` (folosită împreună cu `case` și `otherwise`) și instrucțiuni repetitive: `for` și `while` (folosite împreună cu `break` și `continue`).

Instrucțiunea `if`. Execută un set de instrucțiuni când o expresie este adevărată.

```
if (expresie)
      [instrucțiuni_1]
else
      [instrucțiuni_2]
end
```

Dacă valoarea `expresiei` este adevărată, atunci se execută blocul de instrucțiuni `instrucțiuni_1`, altfel se execută blocul de instrucțiuni `instrucțiuni_2`.

- Dacă `a` este egal cu `b` să se afișeze mesajul '*a este egal cu b*' altfel să se afișeze mesajul '*a este diferit de b*'.

```
a = 2;
b = 3;
if(a == b)
    disp('a este egal cu b')
else
    disp('a este diferit de b')
end
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
a este diferit de b
```

Dacă este nevoie să se folosească mai multe `if`-uri, pentru simplificarea scrierii se poate folosi `if` în combinație cu `elseif`, astfel:

```
if (expresie)
      [instrucțiuni_1]
elseif (expresie)
      [instrucțiuni_2]
else
      [instrucțiuni_3]
end
```

Observație: `elseif` nu se termină cu `end`.

Să se verifice și să se afișeze dacă o variabilă `a` este mai mică decât 2, mai mare decât 2 sau egală cu 2.

```
a = 5;
if(a < 2)
    disp('a < 2')
elseif(a > 2)
    disp('a > 2')
else
    disp('a = 2')
end
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
a > 2
```

Instrucțiunea `switch`. Se folosește atunci când se dorește implementarea unor secvențe imbricate mai complexe**.**

```
switch (expresie)
     case (valoare_1)
          [instrucțiuni_1]
     case (valoare_2)
          [instrucțiuni_2]
      ...
     otherwise,
          [instrucțiuni_otherwise]
end
```

Dacă variabila `expresie` are valoarea egală cu `valoare_1` atunci se execută blocul de instrucțiuni `instrucțiuni_1`, dacă are valoarea egală cu `valoare_2` atunci se execută blocul de instrucțiuni `instrucțiuni_2`, ș.a.m.d. altfel se execută blocul de instrucțiuni `instrucțiuni_otherwise`.

Să se folosească instrucțiunea `switch` pentru a determina dacă variabila `filtru` are valoarea `FTJ` (Filtru Trece Jos) sau `FTS` (Filtru Trece Sus).

```
filtru = 'FTJ';
    switch(filtru)
        case 'FTJ'
            disp('ai ales filtru FTJ')
        case 'FTS'
            disp('ai ales filtru FTS')
        otherwise
            disp('ai ales altceva decat FTJ sau FTS')
    end
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
ai ales filtru FTJ
```

Instrucțiunea `for`. Este utilă pentru realizarea unei structuri repetitive, condiționată anterior.

```
for var = val_start : pas : val_stop
          [instrucțiuni]
end
```

Pentru variabila `var` luând valori în intervalul `val_start ÷ val_stop`, din `pas` în `pas`, se execută blocul de instrucțiuni. Dacă nu se scrie variabila `pas`, se consideră `pas` = 1.

Fie un vector `X`. Să se genereze vectorul `Y` care să conțină următoarele elemente $[X(1),\ X(2)^2,\ X(3)^3, \dots, X(n)^n]$.

```
Y = [];
X = [2 -3 -2 10 0];
for i = 1 : length(X)
    Y(i) = X(i)^i;
end
disp(Y)
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
     2     9    -8    10000     0
```

Instrucțiunea `while`. Execută un set de instrucțiuni atâta timp cât o expresie este adevărată.

```
while (expresie)
      [instrucțiuni]
end
```

Să se genereze numere pseudorandom uniform distribuite în intervalul (0,1) până când se obține un număr mai mare decât 0.9.

```
i = 1;
numar(i) = rand;
    while(numar(i) < 0.9 )
        i = i + 1;
        numar(i) = rand;
    end
disp(numar)
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
    0.6324    0.0975    0.2785    0.5469    0.9575
```

Observație: folosind generatoare pseudorandom, la rulări diferite se obțin rezultate diferite (a se vedea *5.4. Generarea matricelor cu valori pseudorandom*).

Instrucțiunea `break`. Forțează întreruperea unei bucle `for` sau `while`.

Folosind o buclă `for`, să se găsească primul element dintr-un vector `X` care este precedat de un număr impar și urmat de un număr par. Să se afișeze numărul găsit și apoi să se iasă din bucla `for`.

```
X = [1 4 3 6 4 5 8 2];
for i = 2 : length(X)-1
    % rem(X,Y) calculeaza restul impartirii lui X la Y
    if(rem(X(i-1),2)~=0)&(rem(X(i+1),2)==0)
        disp('numarul gasit este:'), disp(X(i))
        break
    end
end
```

În urma rulării programului de mai sus, în *Command Window* se va afișa doar:

```
numarul gasit este:
     6
```

Observație: dacă nu s-ar fi folosit instrucțiunea `break` atunci s-ar fi afișat 6 și 8.

Instrucțiunea `continue`. Este folosită într-o buclă `for` sau `while` pentru a forța trecerea la următoarea iterație, fără a mai rula instrucțiunile care urmează între instrucțiunea `continue` și `end`-ul buclei `for`.

Să se determine care dintre elementele unui vector sunt numere pare și care sunt numere impare.

```
X = [7 2 10 3 8 5 1 9 12 11 13];
nr_pare = [];
nr_impare = [];
for i = 1 : length(X)
    if(rem(X(i), 2))
        nr_impare = [nr_impare, X(i)];
        continue;
    end
    nr_pare = [nr_pare, X(i)];
end
disp(['numere impare: ', num2str(nr_impare)])
disp(['numere pare: ', num2str(nr_pare)])
```

În urma rulării programului de mai sus, în *Command Window* se va afișa:

```
numere impare: 7   3   5   1   9  11  13
numere pare: 2  10   8  12
```

Exemple de probleme care se pot rezolva
mai rapid folosind lucrul cu matrice decât utilizând bucla FOR

MATLAB-ul este optimizat să lucreze cu matrice. De aceea, este bine să se folosească buclele `for` și `while` doar atunci când nu sunt alte alternative sau se dorește o scriere mai detaliată a codului.

- Să se realizeze suma a oricăror două elemente consecutive dintr-un vector.

Varianta 1 (cu FOR)

```
X = [2 3 0 1 -3];
for i = 1 : length(X) -1
    suma(i) = X(i) + X(i+1);
end

suma =
     5     3     1    -2
```

Varianta 2 (fără FOR)

```
X = [2 3 0 1 -3];
suma = X(1, 1:end-1) + X(1, 2:end);

suma =
     5     3     1    -2
```

- Să se ridice la pătrat fiecare element dintr-un vector.

Varianta 1 (cu FOR)

```
X = [2 3 0 1 -3];
for i = 1 : length(X)
    Y(i) = X(i)^2;
end

Y =
     4     9     0     1     9
```

Varianta 2 (fără FOR)

```
X = [2 3 0 1 -3];
Y = X.^2;

y =
     4     9     0     1     9
```

- Să se copieze toate numerele pozitive dintr-un vector `X` într-un vector Y.

Varianta 1 (cu FOR)

```
X = [5 8 -2 2 3 -1 4 2 -6];
j = 1;
for i = 1 : length(X)
    if(X(i)>=0)
        Y(j) = X(i);
        j = j + 1;
    end
end
```

Varianta 2 (fără FOR)

```
X = [5 8 -2 2 3 -1 4 2 -6];
Y = X(find(X >= 0));
```

În ambele cazuri se obține `Y = [5 8 2 3 4 2]`

- Fie un vector `x` de valori distincte. Să se genereze un vector `y` care să indice maximele locale:

$y(n) = 0$ dacă $x(n+0) > x(n-1)$ și $x(n+0) > x(n+1)$

$y(n) = 1$ dacă $x(n+1) > x(n-1)$ și $x(n+1) > x(n+0)$

$y(n) = -1$ dacă $x(n-1) > x(n+0)$ și $x(n-1) > x(n+1)$

Varianta 1 (cu FOR)

```
X = [2 3 0 1 -3];
X = [-inf, X, -inf];
for i = 2 : length(X)-1
    if ( (X(i) > X(i+1)) && (X(i) > X(i-1)) )
        Y(i-1) = 0;
    elseif ( (X(i-1) > X(i+1)) && (X(i-1) > X(i)) )
        Y(i-1) = -1;
    else
        Y(i-1) = +1;
    end
end
```

Varianta 2 (fără FOR)

```
X = [2 3 0 1 -3];
len = length(X);
X_matrice = [-inf, X(1:len-1); X; X(2:len), -inf];
[val,poz] = max(X matrice,[],1);
Y = poz-2;
```

În ambele cazuri se obține `Y = [1 0 -1 0 -1]`

7.1 Funcţii

Funcțiile sunt programe care rezolvă anumite cerințe (de exemplu calculul mediei elementelor dintr-o matrice, transformarea unei imagini color într-o imagine grayscale, redarea unui semnal audio, antrenarea unei rețele neurale etc). Atunci când avem de dezvoltat o aplicație mai complexă este bine ca aceasta să fie împărțită în module mai mici care să rezolve doar anumite cerințe, fiecare modul fiind implementat într-o funcție. Această abordare modulară face ca un program să fie ușor de urmărit și de un alt utilizator și permite un *debug* mai eficient. O aplicație care a fost dezvoltată modular este mult mai ușor de dezvoltat ulterior, prin simpla adăugare a altor module (funcții).

Un alt avantaj al folosirii funcțiilor este *reutilizarea*, ceea ce înseamnă că o funcție odată creată poate fi folosită de oricâte ori (evitându-se astfel multiplicarea redundantă a codului). De asemenea, o funcție creată pentru o aplicație poate fi apoi refolosită într-o altă aplicație.

Sintaxa unei funcții este:

```
function [out1, out2, ...] = NumeFunctie(in1, in2, ...)
      [instrucțiuni]
end % end nu este intotdeauna obligatoriu
```

unde:

- `NumeFunctie` reprezintă numele funcției; numele funcției începe cu literă și poate conține numai litere, cifre și underscore (_)
- `out1`, `out2`, ... reprezintă parametrii de ieșire și sunt salvați într-un vector (de aceea sunt între paranteze pătrate).
- `in1`, `in2`, ... reprezintă parametrii de intrare, sunt argumentele funcției și se scriu între paranteze rotunde.

Observații:

- numele fișierului în care se salvează funcția trebuie să coincidă cu numele funcției; dacă numele funcției este *NumeFunctie* atunci numele fișierului trebuie să fie *NumeFunctie.m*
- prima linie care se execută din fișierul în care este salvată funcția conține antetul funcției (înainte de antet pot exista doar comentarii).
- nu este obligatoriu ca o funcție să aibă parametri de intrare și nici parametri de ieșire.

Este recomandat ca imediat după antetul funcției să existe comentarii sugestive referitoare la: utilitatea funcției, care este semnificația parametrilor de intrare și a celor de ieșire, când a fost modificată funcția ultima dată și de către cine etc (aceste comentarii vor fi afișate atunci când se rulează comanda `>> help NumeFunctie`).

- Să se implementeze o funcție care să calculeze diferența dintre valoarea maximă și valoarea minimă dintr-o matrice.

```
function diferenta = diferenta_max_min(A)
% Functia diferenta_max_min calculeaza diferenta dintre valoarea
maxima si valoarea minima a matricei A
% Parametru de intrare: matricea A
% Parametru de iesire: variabila diferenta in care se salveaza
% diferenta dintre valoarea maxima si valoarea minima a lui A

  val_min = min(A(:));
  val_max = max(A(:));
  diferenta = val_max - val_min;
```

Fișierul în care se va salva funcția de mai sus se va numi *diferenta_max_min.m*

- Să se afișeze în *Command Window* help-ul funcției `diferenta_max_min`

```
>> help diferenta_max_min
Functia diferenta_max_min calculeaza diferenta dintre valoarea
maxima si valoarea minima a matricei A
  Parametru de intrare: matricea A
  Parametru de iesire: variabila diferenta in care se salveaza
diferenta dintre valoarea maxima si valoarea minima a lui A
```

Pentru a putea utiliza o funcție, aceasta se apelează folosind sintaxa:

```
[out1, out2, ...] = NumeFunctie(in1, in2, ...)
```

O funcție se poate apela din *Command Window*, dintr-un program script sau dintr-o altă funcție. Pentru ca o funcție să fie găsită atunci când este apelată, aceasta trebuie să se afle în directorul curent sau într-unul dintre directoarele aflate în căile de căutare ale MATLAB-ului. Pentru a adăuga un director în căile de căutare, se folosește sintaxa `addpath('cale absolută sau cale relativă către director')`.

- Folosind funcția `diferenta_max_min` creată mai sus, să se calculeze diferența dintre maximul și minimul unei matrice `A`.

```
>> A = [1 -2 0; 3 8 2; 1 9 0];
>> dif = diferenta_max_min(A)
dif =
    11
```

Un fișier poate conține mai multe funcții: prima funcție numită și *funcție principală* urmată de alte funcții numite *subfuncții*. În acest caz numele fișierului trebuie să coincidă cu numele funcției principale.

```
function [out1, out2,…] = NumeFunctie(in1, in2,…)
                              [instrucțiuni]
function [outSF1_1, outSF1_2,…] = NumeSF_1(inSF1_1, inSF1_2,…)
                              [instrucțiuni_SF1]
function [outSF2_1, outSF2_2, …] = NumeSF_2(inSF2_1, inSF2_2,…)
                              [instrucțiuni_SF2]
                                …
```

Utilizarea mai multor funcții în același fișier va fi foarte folositoare atunci când se va lucra cu interfețe grafice (a se vedea *10. Interfață grafică în Matlab*).

Observații:
Dacă într-un fișier există o singură funcție, este opțională terminarea acesteia cu `end`. Dacă un fișier conține mai multe funcții, ori se termină toate cu `end`, ori nu se va folosi `end` la sfârșitul niciunei funcții.

În MATLAB există posibilitatea folosirii funcțiilor imbricate. O *funcție imbricată* este o funcție conținută în interiorul unei alte funcții numită *funcție părinte*. O funcție imbricată poate accesa și modifica variabile din funcția părinte fără ca aceste variabile să fie transmise însă ca parametri ai funcției. Funcțiile imbricate nu vor fi detaliate în această carte.

Variabile locale și variabile globale

Variabilele folosite în funcțiile din MATLAB sunt în mod implicit *variabile locale*. Cu alte cuvinte, o variabilă `x` este vizibilă doar în funcția în care a fost generată.

Fie fișierul *suma.m* care conține funcția principală `suma(x,y)` și care apelează apoi subfuncția `afisare()`. Ne dorim ca apelând funcția `suma(x,y)`, în `z` să se calculeze suma dintre `x` și `y` și apoi să se apeleze funcția `afisare()` care realizează afișarea în *Command Window* a lui `z`. Așa cum este scris programul următor, afișarea lui `z` nu va avea loc, deoarece variabila `z` este vizibilă doar în funcția `suma(x,y)`.

```
function suma(x, y)
z = x + y;
afisare()

function afisare()
disp(['x + y = ', num2str(z)])
```

Dacă se va apela din *Command Window* funcția `suma`, se va obține un mesaj de eroare.

```
>> suma(2,3)
Undefined function or variable 'z'.
```

Pentru a rezolva problema de mai sus, trebuie declarată variabila `z` ca **variabilă globală**. Dacă o variabilă este declarată globală în mai multe funcții, atunci toate acele funcții vor folosi copia acelei variabile. Orice modificare a variabilei realizată într-o funcție este vizibilă și în celelalte funcții în care variabila a fost declarată globală. În consecință programul de mai sus poate fi rescris astfel:

```
function suma(x, y)
global z
z = x + y;
afisare()

function afisare()
global z
disp(['x + y = ', num2str(z)])
```

Dacă se va apela funcția `suma` din *Command Window*, se va obține:

```
>> suma(2,3)
x + y = 5
```

O alternativă la programul de mai sus este transmiterea variabilei `z` ca parametru al funcției `afisare`.

```
function suma(x, y)
z = x + y;
afisare(z)

function afisare(z)
disp(['x + y = ', num2str(z)])
```

7.2 Aplicații ale operațiilor cu matrice

În continuare se vor prezenta câteva procesări simple realizate asupra imaginilor, semnalelor audio și datelor obținute în urma unor experimente. Toate procesările prezentate sunt implementate fără a folosi funcții dedicate ci doar simple operații cu matrice prezentate în capitolele anterioare.

Adăugarea a *n* secunde de liniște într-un semnal audio

Să se adauge *n* secunde de liniște în mijlocul unui semnal audio monocanal. Dacă semnalul audio este pe un singur canal, rezolvarea problemei presupune practic adăugarea de zero-uri în mijlocul unui vector. Se va implementa o funcție care primește ca parametru de intrare semnalul audio, numărul secundelor de liniște și frecvența cu care a fost eșantionat semnalul (pentru a se ști câte zerouri să se genereze). Funcția va întoarce semnalul modificat.

```
function semnal_cu_liniste = adaugare_liniste(y, Fs, n)
% y = semnalul audio caruia i se adauga n secunde de liniste la
jumatate
% Fs = frecventa cu care a fost esantionat semnalul
% n = numarul secundelor de liniste care se adauga
lungime_semnal = length(y);
semnal_liniste = zeros(2*Fs,1);
partea_I = y(1:round(lungime_semnal/2));
partea_II = y(round(lungime_semnal/2)+1:end);
semnal_cu_liniste = [ partea_I; semnal_liniste; partea_II];
```

Programul principal din care se apelează funcția de mai sus este:

```
[y, Fs] = audioread('Prokofiev.wav');
semnal_cu_liniste = adaugare_liniste(y, Fs, 2);
sound(semnal_cu_liniste, Fs)
```

În urma rulării programului principal se va reda semnalul audio original care va avea la mijloc 2 secunde de liniște.

Implementarea efectelor de *fade in* și *fade out* pentru un semnal audio

Pentru un semnal audio monocanal să se implementeze efectul de *fade* (*fade in:* se modifică volumul semnalului audio astfel încât acesta să pornească de la zero iar la final să ajungă la volumul semnalului nemodificat; *fade out*: se modifică volumul semnalului audio astfel încât acesta să pornească de la volumul inițial iar la final să ajungă la zero). Modificarea volumului se va face liniar.

Dacă semnalul are *N* eșantioane, rezolvarea problemei pentru efectul *fade in* presupune ca primul eșantion al semnalului să se înmulțească cu zero, al doilea să se înmulțească cu 1/*(N* – 1) al treilea cu 2/(*N* – 1) etc, ultimul eșantion înmulțindu-se cu 1(adică rămâne așa cum era). Pentru efectul *fade out*, primul eșantion de înmulțește cu 1, al doilea se înmulțește cu (*N* - 2)/(*N* - 1) iar ultimul cu zero. Se va implementa o funcție care va primi ca parametri de intrare semnalul original și efectul dorit: '*fadein*' sau '*fadeout*'. La ieșire, funcția va întoarce semnalul audio modificat.

```
function semnal_fade = efect_fade(y, param)
% y = semnalul audio caruia i se adauga efect de fade in
% param poate avea valorile:
% 'fadein' daca se doreste efect fade in
% 'fadeout' daca se doreste efect fade out
volum_in = 0:1/(length(y)-1):1;
volum_out = sort(volum_in,'descend'); % sau: flip(volum_in)
switch(param)
    case('fadein')
        semnal_fade = y.*volum_in';
    case('fadeout')
        semnal_fade = y.*volum_out';
end
```

Programul principal din care se apelează funcția de mai sus este:

```
[y, Fs] = audioread('D:\carte Matlab\audio\FranzVonSuppe.wav');
% efect de fade in
y_fade_in = efect_fade(y,'fadein');
sound(y_fade_in,Fs)

% efect de fade out
y_fade_out = efect_fade(y,'fadeout');
sound(y_fade_out,Fs)
```

Să se concateneze două semnale audio monocanal. Înainte de concatenare să se realizeze efect de *fade out* pentru primul semnal și efect de *fade in* pentru al doilea semnal. Se va folosi funcția *efect_fade* implementată anterior. *Observație*: ambele semnale au fost eșantionate cu aceeași frecvență de eșantionare.

```
[y1, Fs] = audioread('D:\carte Matlab\audio\FranzVonSuppe.wav');
[y2, Fs] = audioread('D:\carte Matlab\audio\Prokofiev.wav');
% efect de fade out
y1_fade_out = efect_fade(y1,'fadeout');

% efect de fade in
y2_fade_in = efect_fade(y2,'fadein');

y = [y1_fade_out; y2_fade_in];
sound(y, Fs)
```

Încadrarea unei imagini într-un chenar

Să se încadreze o imagine grayscale cu o bordură neagră. Rezolvarea presupune practic încadrarea unei matrice cu valori de zero. Se va implementa o funcție care primește ca parametru de intrare imaginea și numărul de linii și de coloane cu care se dorește a se realiza încadrarea. La ieșire funcția va întoarce imaginea încadrată.

```
function img_incadrata = bordare_imagine(imagine, M)
% imagine = imaginea ce se doreste a fi incadrata
% M = numar linii si coloane pentru incadrare
% img_incadrata = imaginea incadrata
bordura_sus = zeros(M, size(imagine,2) + 2*M);
bordura_jos = bordura_sus;
bordura_stanga = zeros(size(imagine,1),M);
bordura_dreapta = bordura_stanga;
img_incadrata = [bordura_sus;...
                 bordura_stanga, imagine, bordura_dreapta;...
                 bordura_jos];
% alternativ: [rows,cols,lays] = size(imagine);
% img_incadrata = zeros(rows + 2*M, cols + 2*M, lays);
% img_incadrata(M+1:M+rows,M+1:M+cols,:) = imagine;
```

Programul principal din care se apelează funcția de mai sus este:

```
imagine = imread('catedrala.jpg');
figure(1), imshow(imagine)
M = 50;
img_incadrata = bordare_imagine(imagine, M);
figure(2), imshow(img_incadrata)
```

În urma rulării vor rezulta următoarele imagini:

FImagine originală

Imagine încadrată

Adăugarea unui pătrat verde în mijlocul unei imagini

Să se adauge un pătrat verde în mijlocul unei imagini color. Dimensiunea pătratului va fi *p*% din lățimea imaginii. Se va implementa o funcție care va avea ca parametri de intrare imaginea și procentul *p*. Funcția va întoarce imaginea care va avea în centru pătratul verde.

```
function imag = adauga_patrat(imagine, p)
% functia adauga_patrat va adauga un patrat in mijlocul unei imagini
% dimensiunea patratului este de p/100 din latimea imaginii
[latime, lungime, straturi] = size(imagine);
dim_patrat = round(latime*p/100);
imag = imagine;
imag(round(latime/2-dim_patrat/2):round(latime/2+dim_patrat/2),...
round(lungime/2-dim_patrat/2):round(lungime/2+dim_patrat/2),1)=0;
imag(round(latime/2-dim_patrat/2):round(latime/2+dim_patrat/2),...
round(lungime/2-dim_patrat/2):round(lungime/2+dim_patrat/2),2)=255;
imag(round(latime/2-dim_patrat/2):round(latime/2+dim_patrat/2),...
round(lungime/2-dim_patrat/2):round(lungime/2+dim_patrat/2),3)=0;
```

Programul principal din care se apelează funcția de mai sus este:

```
imagine = imread('D:\carte Matlab\poze\nazca.jpg');
imag_cu_patrat=adauga_patrat(imagine, 5);
figure(1), imshow(imag_cu_patrat)
```

Rezultatul este următorul:

Imagine în care s-a adăugat un pătrat verde

Decodarea unui mesaj folosind codul cu triplă repetiție

Să se decodeze un mesaj binar codat folosind codul cu repetiție triplă și afectat de erori. Se va implementa o funcție care va avea ca parametru de intrare mesajul codat și afectat de erori. Funcția va întoarce mesajul decodat.Decodarea mesajelor codate cu ajutorul codului cu repetiție triplă se poate face prin logică majoritară: mesajul se împarte în grupuri de câte 3 biți; dacă într-un grup, doi biți au valoarea 0, bitul original a fost 0; dacă într-un grup, doi biți au valoarea 1, bitul original a fost 1.

```
function mesaj_decodat = decodareRepetTripla(mesaj_codat_cu_erori)

% mesaj_codat_cu_erori = un vector linie ( sir de biti ) corespunzand unui mesaj codat cu codul cu repetitie tripla si afectat de erori sporadice
% mesaj_decodat = mesajul original, obtinut prin decodarea mesaj_codat (si corectarea implicita a erorilor)

Grupuri = reshape(mesaj_codat_cu_erori,3, []);
SumaInGrup = sum(Grupuri,1);
mesaj_decodat = (SumaInGrup >= 2);
```

Programul principal din care se va apela funcția de mai sus este:

```
mesaj_codat_cu_erori =  [0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1];
mesaj_decodat = decodareRepetTripla(mesaj_codat_cu_erori)
```

In urma rulării, se obține:

```
>> mesaj_decodat =
     0     1     1     0     1     1
```

Calculul matricei de confuzie

Să se calculeze matricea de confuzie cunoscând ieșirile obținute ale unui algoritm de clasificare precum și ieșirile dorite.

Explicații: în cazul algoritmilor de clasificare, matricea de confuzie reprezintă o metodă tabelară de afișare a rezultatelor, în care sunt scoase în evidență erorile sistematice de clasificare. Dimensiunea matricei de confuzie este `K x K`, unde `K` reprezintă numărul de clase folosite pentru clasificare iar elementul `M(i,j)` reprezintă numărul elementelor din clasa `i` clasificate în clasa `j`.

Elementele clasificate corect se găsesc pe diagonala principală. Pentru a extrage numărul total al elementelor clasificate corect, se calculează urma matricei (suma elementelor de pe prima diagonală).

Exemplu: Un algoritm de clasificare este testat pe un lot de test ce conține 15 vectori cu următoarea distribuție pe clase:

- Primii 5 vectori fac parte din clasa R (ROȘU),
- Următorii 5 vectori fac parte din clasa G (VERDE),
- Ultimii 5 vectori fac parte din clasa B (ALBASTRU).

În urma rulării algoritmului, vectorii sunt clasificați astfel:
ieșiri = ['R','R','R','R','B', 'G','R','R','R','G', 'B','B','G','B','B']
Rezolvare:
Se notează clasele (R = 1, G = 2, B = 3).
Se construiesc vectorii `D` (ieșirile dorite) și `O` (ieșirile obținute).

D = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3];
O = [1, 1, 1, 1, 3, 2, 1, 1, 1, 2, 3, 3, 2, 3, 3].

Matricea de confuzie obținută va fi: $M = \begin{bmatrix} 4 & 0 & 1 \\ 3 & 2 & 0 \\ 0 & 1 & 4 \end{bmatrix}$

```
function matrice_confuzie = calculMatriceConfuzie(D,O)
% D = vectorul iesirilor dorite
% O = vectorul iesirilor obtinute
% D si O trebuie sa contina valori numere naturale intre 1 si K
% K = numarul de clase

% matrice_confuzie(i,j) contine numarul de vectori din clasa i
% clasificati in clasa j
 K = max(D(:));
 matrice_confuzie = zeros(K,K);
for i = 1:K
    for j = 1:K
        matrice_confuzie(i,j) = length( find( (D==i)&(O==j) ) );
    end
end
```

Programul principal din care se apelează funcția de mai sus este:

```
D = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3];
O = [1, 1, 1, 1, 3, 2, 1, 1, 1, 2, 3, 3, 2, 3, 3];
M = calculMatriceConfuzie(D,O);
tabel_rezultate = table(M(:,1), M(:,2),M(:,3),'VariableNames',...
{'Vectori_clasif_in_C1', 'Vectori_clasif_in_C2',
'Vectori_clasif_in_C3'} ,...
'RowNames',{'Vectori_din_C1', 'Vectori_din_C2', 'Vectori_din_C3'})
```

În urma rulării programului principal se va obține:

```
matrice_de_confuzie =
```

	Vectori_clasif_in_C1	Vectori_clasif_in_C2	Vectori_clasif_in_C3
Vectori_din_C1	4	0	1
Vectori_din_C2	3	2	0
Vectori_din_C3	0	1	4

8. Calcul parametric (simbolic) în MATLAB

Pentru a rezolva o problemă, de multe ori se fac mai întâi calculele la nivel parametric și abia la final se realizează înlocuirea numerică. În continuare se vor prezenta câteva dintre funcțiile MATLAB de bază care permit calculul parametric precum și rezolvarea ecuațiilor și a sistemelor de ecuații.

Variabile simbolice

Rezolvarea parametrică se poate realiza în MATLAB folosind *variabile simbolice*. Acestea se declară folosind instrucțiunile:

- **`sym`** pentru o singură variabilă simbolică:
  `x = sym('a')` ➔ se construiește un obiect `x` de clasă *symbol* pentru parametrul `a`.
- **`syms`** pentru a declara mai rapid una sau mai multe variabile simbolice:
  `syms x y` ➔ echivalent cu `x = sym('x')` și `y = sym('y')`.

Folosind variabile simbolice se pot genera expresii matematice.

Fie $S(a,b) = c^2 + 3a + 2b + 2c^2 + 5a - b$ și $E(a,b) = b - a + 3$. Să se calculeze $S(a,b) + E(a,b)$.

```
>> syms a b c
>> S = c^2 + 3*a + 2*b + 2*c^2 + 5*a - b;
>> E = b - a + 3;
>> rezultat = S + E
rezultat =
  3*c^2 + 7*a + 2*b + 3
```

Observație: variabila `rezultat` va fi tot o variabilă simbolică.

Funcția `expand(expresie)`. Este folosită pentru a putea dezvolta expresii simbolice polinomiale și nu numai (trigonometrice, exponențiale etc)

Fie $S(a,b) = (a+b)^2 + 2b^2 - a^2$. Să se dezvolte expresia *S*.

```
>> syms a b
>> S = (a+b)^2+2*b^2-a^2
 S =
 (a + b)^2 - a^2 + 2*b^2
>> expand(S)
 ans =
 3*b^2 + 2*a*b
```

Alte exemple pentru utilizarea funcției `expand`:

```
>> syms x y z a b
>> S = x*(y + z);
>> expand(S)
ans =
x*y + x*z

>> S = (a + b)*(a - b);
>> expand(S)
ans =
a^2 - b^2

>> S = x^(a + b);
>> expand(S)
ans =
x^a*x^b

>> S = sin(a + b);
>> expand(S)
ans =
cos(a)*sin(b) + cos(b)*sin(a)

>> S = sin(2*a);
>> expand(S)
ans =
2*cos(a)*sin(a)

>> S = exp(x + y);
>> expand(S)
ans =
exp(x)*exp(y)
```

Funcția `simplify(expresie)`. Este folosită pentru a aduce la o formă cât mai simplă o expresie matematică.

- Fie $S(x) = x^2 - 6x + 8$. Să se aducă expresia *S* la forma restrânsă.

```
>> syms x
>> S = x^2-6*x+8;
>> simplify(S)
ans =
(x - 2)*(x - 4)
```

Funcția `pretty(expresie)` afișează o expresie într-un format cât mai apropiat scrierii matematice.

```
syms a b
S = (a+b)^2+2*b^2-a^2;
pretty(S)
```

La rularea programului se va afișa în *Command Window*:

```
                2     2       2
(a + b)   - a   + 2 b
```

Funcția `subs(expresie, {simboluri}, {val_numerice})`
Realizează înlocuirea numerică a parametrilor dintr-o expresie.

- Să se afle valoarea expresiei $E = x - y^2 - 2z + 4$, pentru `x = 1`, `y = 3` și `z = 2`.

```
>> syms x y z
>> E = x - y^2 - 2*z + 4;
>> rezultat = subs(E, {'x','y','z'}, {1, 3, 2})
rezultat =
 -8
```

Observație: atunci când se folosește funcția `subs`, expresia matematică poate fi scrisă și ca șir de caractere, fără a mai declara variabile simbolice. Rezultatul întors va fi o variabilă simbolică.

```
>> E = 'x - y^2 - 2*z + 4';
>> rezultat = subs(E, {'x','y','z'}, {1, 3, 2})
 rezultat =
 -8
```

Dacă expresia are un singur parametru simbolic se poate folosi sintaxa simplificată: `subs(expresie, val_numerică)`.

- Fie $S(x) = x^2 - 6x + 8$. Să se afle $S(3)$.

```
>> E = 'x^2-6*x+8';
>> subs(E,3)
ans =
    -1
```

Operații parametrice cu matrice

Pentru a genera o matrice simbolică `A` de dimensiune `m` x `n`, se folosește sintaxa: `A = sym('a',[m,n])`. Elementele matricei `A` vor fi în acest caz simboluri notate astfel: `a1_1`, `a1_2` …

- Să se genereze o matrice simbolică `A` cu 2 linii și 3 coloane.

```
>> A = sym('a',[2,3])
 A =
 [ a1_1, a1_2, a1_3]
 [ a2_1, a2_2, a2_3]
```

- Să se genereze o matrice simbolică `A` cu 2 linii și 3 coloane și o matrice simbolică `B` cu 3 linii și 2 coloane. Să se determine `A x B`.

```
>> A = sym('a',[2,3])
 A =
[ a1_1, a1_2, a1_3]
[ a2_1, a2_2, a2_3]

>> B = sym('b',[3,2])
 B =

[ b1_1, b1_2]
[ b2_1, b2_2]
[ b3_1, b3_2]

>> C = A*B
 C =
 [ a1_1*b1_1 + a1_2*b2_1 + a1_3*b3_1, a1_1*b1_2 + a1_2*b2_2 + a1_3*b3_2]
 [ a2_1*b1_1 + a2_2*b2_1 + a2_3*b3_1, a2_1*b1_2 + a2_2*b2_2 + a2_3*b3_2]
```

Rezolvarea ecuațiilor

Pentru rezolvarea ecuațiilor se poate folosi funcția **`solve`** cu sintaxa `solve(expresie, var)`. Parametrul `var` specifică variabila considerată necunoscută în ecuație.

- Fie ecuația $x^2 - y^2 = 0$. Să se calculeze *x* în funcție de *y* și *y* în funcție de *x*.

```
>> E = 'x^2 - y^2 = 0';
>> solutie_x = solve(E, 'x')
 solutie_x =
  y
 -y

>> solutie_y = solve(E, 'y')
solutie_y =
   x
  -x
```

Dacă se dorește rezolvarea unei ecuații ce conține o singură variabilă simbolică atunci nu mai este nevoie să se specifice care este necunoscuta, se folosește direct sintaxa `solve(expresie)`.

- Să se rezolve ecuația $x^2 - 1 = 0$.

```
>> E = 'x^2 -1 = 0';
>> solutie = solve(E)
 solutie =
  -1
   1
```

Observație: variabila `solutie` este de tip *sym*. Dacă se dorește transformarea variabilei `solutie` într-o variabilă de tip numeric, de exemplu *double*, se va face conversia `solutie = double(solutie)`.

- Să se afle valoarea parametrică a lui `x` din ecuația $x - y^2 - 2z + 4 = 0$. Pentru `y` = 3 și `z` = 2 să se determine valoarea numerică a lui `x`.

```
>> E = 'x-y^2 - 2*z + 4 = 0';
>> solutie_x = solve(E, 'x');
>> rezultat = subs(solutie_x, {'y','z'}, {3, 2})
rezultat =
 9
```

Rezolvarea sistemelor de ecuații

Fie sistemul de ecuații $\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$, cu necunoscutele *x* și *y*. Acest sistem se poate rezolva:

- **Numeric** (se determină *x* și *y* cunoscând valorile pentru $a_1, b_1, c_1, a_2, b_2, c_2$)
- **Parametric** (se determină *x* și *y* în funcție de $a_1, b_1, c_1, a_2, b_2, c_2$)

În continuare se vor prezenta cele două modalități de rezolvare a sistemelor de ecuații în MATLAB.

Rezolvarea numerică a unui sistem de ecuații

Pentru rezolvarea numerică a sistemelor de ecuații se folosește funcția **`solve`**, cu sintaxa `solve(expresie_1, expresie_2, … , expresie_n)`. Fiecare expresie reprezintă o ecuație.

- Să se determine valorile necunoscutelor *x* și *y* din următorul sistem de ecuații:

$$\begin{cases} 3x + y = 5 \\ 2x - 3y = -4 \end{cases}$$

```
ec1 = '3*x+y=5';
ec2 = '2*x-3*y = -4';
solutie = solve(ec1, ec2);
```

Variabila `solutie` de mai sus este o structură ce conține valorile tuturor necunoscutelor, în cazul de față `x` și `y`. Pentru a afla valorile lui x și y se va scrie în continuarea programului:

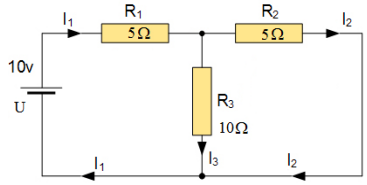
```
x = double(solutie.x)
y = double(solutie.y)
```

În urma rulării programului vor rezulta valorile `x` = 1 și `y` = 2.

Rezolvarea parametrică a unui sistem de ecuații

Pentru rezolvarea parametrică a sistemelor de ecuații se folosește funcția **`solve`** cu sintaxa: `solve(expresie_1, … , expresie_n, var_1, … ,var_t)`. Fiecare expresie reprezintă câte o ecuație; `var_1…var_t` reprezintă necunoscutele.

- Un circuit electric este descris de următorul sistem de ecuații:



$$\begin{cases} I_1 = I_2 + I_3 \\ U = I_2 \cdot R_2 + I_1 \cdot R_1 \\ I_2 \cdot R_2 = I_3 \cdot R_3 \end{cases}$$

Să se determine valoarea lui I_1 în funcție de parametrii U, R_1, R_2 și R_3.

```
ec1 = 'I1=I2+I3';
ec2 = 'U=I2*R2+I1*R1';
ec3 = 'I2*R2=I3*R3';
solutie = solve(ec1, ec2, ec3,'I1,I2,I3');
I1 = solutie.I1;
pretty(I1)
```

În urma rulării programului de mai sus, se va afișa valoarea lui I_1:

```
                R2 U + R3 U
--------------------------------------
R1 R2 + R1 R3 + R2 R3
```

Dacă se cunosc valorile numerice ale parametrilor U, R_1, R_2 și R_3 (U = 10V, R_1 = 5Ω, R_2 = 5Ω și R_3 = 10Ω) se poate afla valoarea numerică a lui I_1 astfel:

```
I1_num = subs(I1,{'R1','R2','R3','U'},{5, 5, 10, 10})
```

La fel se poate proceda și pentru aflarea valorilor celorlalți doi curenți.

```
ec1 = 'I1=I2+I3';
ec2 = 'U=I2*R2+I1*R1';
ec3 = 'I2*R2=I3*R3';
solutie = solve(ec1, ec2, ec3,'I1,I2,I3');
I2 = solutie.I2;
I3 = solutie.I3;
pretty(I2)
pretty(I3)
I2_num = subs(I2,{'R1','R2','R3','U'},{5, 5, 10, 10})
I3_num = subs(I3,{'R1','R2','R3','U'},{5, 5, 10, 10})
```

9. Reprezentare grafică în Matlab

În MATLAB, reprezentarea grafică este extrem de variată: se pot face reprezentări 2D, 3D, histograme, reprezentări procentuale, reprezentări vectoriale etc. În continuare vor fi prezentate pe larg reprezentările grafice de bază.

Pentru orice reprezentare grafică este nevoie de o fereastră de afișare. Această fereastră se deschide folosind instrucțiunea `figure`.

Sintaxă: `figure(număr_figură)`

Când reprezentăm un grafic, este bine ca acesta să conțină:

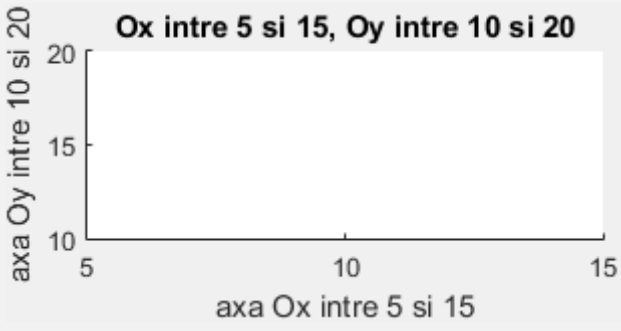
- Titlu reprezentativ pentru graficul respectiv
  Sintaxă: `title('titlul figurii')`
- Semnificația axelor *Ox*, *Oy* și *Oz* (*Oz* doar pentru grafice 3D).
  Sintaxă: `xlabel('semnificația axei Ox')`
  Pentru axele *Oy* și *Oz* există funcțiile `ylabel` și `zlabel`
- În cazul în care se dorește ca valorile axelor să fie în anumite intervale, se poate folosi funcția `axis`.
  Sintaxă: `axis([xmin xmax ymin ymax zmin zmax])` unde: `[xmin xmax ymin ymax zmin zmax])` reprezintă coordonatele minime respectiv maxime ale axelor *Ox, Oy, Oz*

Pentru modificarea limitelor axelor se mai pot folosi funcțiile `xlim`, `ylim` și `zlim`. `xlim([val_min,val_max])` ➔ axa *Ox* va fi între `val_min` și `val_max`. Similar se folosesc și funcțiile `ylim` și `zlim`.

- Să se deschidă o fereastră pentru reprezentare grafică. Axa *Ox* să aibă valori între 5 și 15 iar axa *Oy* să aibă valori între 10 și 20.

```
figure(1)
title('Ox intre 5 si 15, Oy intre 10 si 20')
xlabel('axa Ox intre 5 si 15'), ylabel('axa Oy intre 10 si 20')
axis([5 15 10 20])
```

Rezultatul va fi:



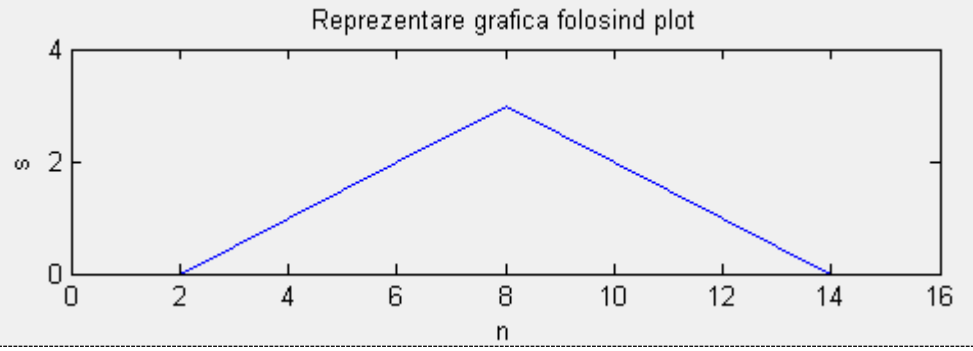
9.1 Reprezentare grafică în spațiul 2-D folosind funcția `plot`

Sintaxă: `plot(x, y)`. Se reprezintă grafic elementele vectorului `y` în funcție de elementele vectorului `x`, folosind interpolarea liniară. Folosind proprietățile implicite ale funcției `plot`, rezultă un grafic conținând perechile `{x(i),y(i)}` unite prin segmente, astfel încât să dea impresia de continuitate.
Atenție! Vectorii `x` și `y` trebuie să aibă aceleași dimensiuni.

- Să se reprezinte grafic vectorul `s` în funcție de vectorul `n`, știind că:
`n = [2, 4, 6, 8, 10, 12, 14]` și `s = [0, 1, 2, 3, 2, 1, 0]`.

```
n = 2 : 2 : 14; % numerele de la 2 la 14 din 2 in 2
s = [0, 1, 2, 3, 2, 1, 0];
figure(1)
plot(n,s), title('Reprezentare grafica folosind plot')
xlabel('n'), ylabel('s'), axis([0, 16, 0, 4])
```

In *figure(1)* se va afișa următorul grafic.



Proprietăți ale funcției `plot`

Pentru a trasa un grafic cu funcția `plot` se pot folosi diverse combinații de linii, markere (puncte pe grafic) și culori, conform tabelului de mai jos.

Tabel 9.1. Proprietăți ale funcției `plot`

Culori		Linii		Markere	
Simbol	**Semnificație**	**Simbol**	**Semnificație**	**Simbol**	**Semnificație**
r	roșu (red)	-	linie continuă ———	.	punct
g	verde (green)	:	linie punctată ·····	o	cerc
b	albastru (blue)	-.	linie întreruptă —·—·	x	cruciuliță
c	turcoaz (cyan)	--	linie întreruptă - - - -	+	plus
m	mov (magenta)	(none)	fără linie	*	steluță
y	galben, (yellow)			s	pătrat (square)
k	negru, (black)			d	romb (diamond)
w	alb, (white)				etc

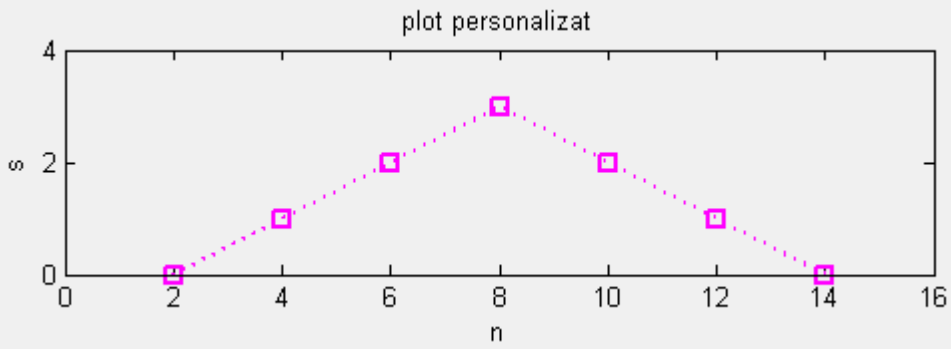
Alte proprietăți:

- `MarkerEdgeColor` pentru a selecta culoarea de contur a markerului
- `MarkerFaceColor` pentru a selecta culoarea de umplere a markerului
- `LineWidth` pentru a seta grosimea liniei
- `MarkerSize` pentru a seta dimensiunea markerului

Să se reprezinte grafic semnalul `s(n)` folosind funcția `plot` cu următorii parametri: linie *punctată*, marker *pătrat*, culoare *magenta*, grosime linie *2*. Se cunosc `n = [2, 4, 6, 8, 10, 12, 14]` și `s = [0, 1, 2, 3, 2, 1, 0]`.

```
n = 2 : 2 : 14; % numerele de la 2 la 14 din 2 in 2
s = [0, 1, 2, 3, 2, 1, 0];
figure(1)
plot(n,s,':ms', 'linewidth',2), title('plot personalizat')
xlabel('n'), ylabel('s'), axis([0, 16, 0, 4])
```

In *figure(1)* se va afișa următorul grafic.



Observații:

- Nu contează ordinea în care sunt scrise cele trei proprietăți (tipul de linie, markere și culori). Pentru a trasa un grafic cu linie *punctată*, marker *pătrat* și culoare *magenta* se poate utiliza oricare dintre următoarele instrucțiuni:

```
plot(n,s,':sm')
plot(n,s,'s:m')
plot(n,s,'m:s')
    etc
```

- Pentru o altă culoare în afara celor menționate mai sus se folosește proprietatea '*color*' urmată de codul *rgb* al culorii.

Pentru a afișa un grafic folosind o nuanță de gri se folosește sintaxa:

```
plot(x, y, 'color', [0.5 0.5 0.5])
```

Pentru a afișa un grafic folosind culoarea portocaliu se folosește sintaxa:

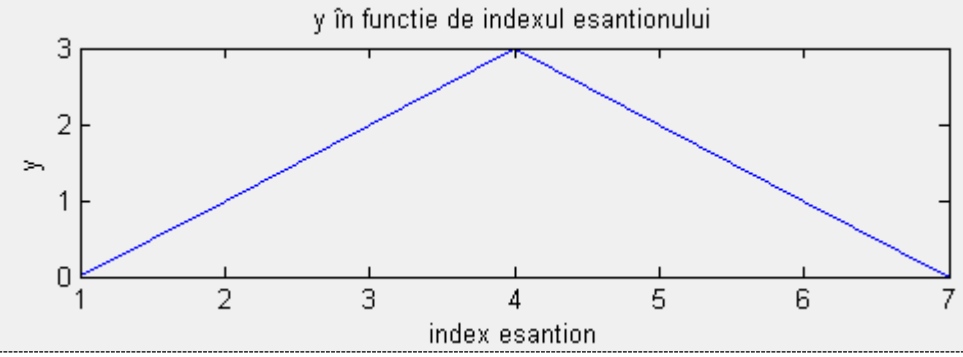
```
plot(x, y, 'color', [1.0 0.5 0.0])
```

- Dacă funcția `plot` se folosește cu un singur parametru, `plot(y)`, în acest caz se consideră că pe axa *Ox* sunt indicii eșantioanelor (numerele `1, 2, 3, …,n,` unde `n` reprezintă numărul de eșantioane din vectorul `y`).

Fie `y = [0, 1, 2, 3, 2, 1, 0]`. Să se reprezinte grafic semnalul `y` în funcție de indexul eșantionului.

```
y = [0, 1, 2, 3, 2, 1, 0];
figure(1)
plot(y), title('y în functie de indexul esantionului')
xlabel('index esantion'), ylabel('y')
```

In *figure(1)* se va afișa următorul grafic.



9.2 Reprezentare grafică în spațiul 2-D folosind funcția `stem`

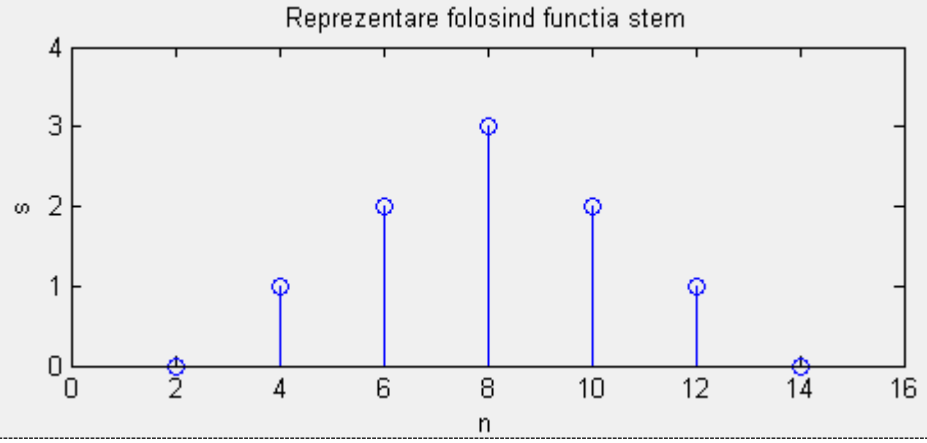
Sintaxă*:* `stem(x,y)`. Se reprezintă grafic elementele vectorului `y` în funcție de elementele vectorului `x`. Se reprezintă doar perechile `{x(i),y(i)}`, fără a mai realiza interpolarea (așa cum se întâmplă în cazul funcției `plot`).

Atenție! Vectorii `x` și `y` trebuie să aibă aceleași dimensiuni.

Să se reprezinte grafic `s[n]` folosind funcția `stem`. Se cunosc vectorii: `n = [2, 4, 6, 8, 10, 12, 14]` și `s = [0, 1, 2, 3, 2, 1, 0]`.

```
n = 2 : 2 : 14; % numerele de la 2 la 14 din 2 in 2
s = [0, 1, 2, 3, 2, 1, 0];
figure(1)
stem(n,s), title('Reprezentare folosind functia stem')
xlabel('n'), ylabel('s'), axis([0, 16, 0, 4])
```

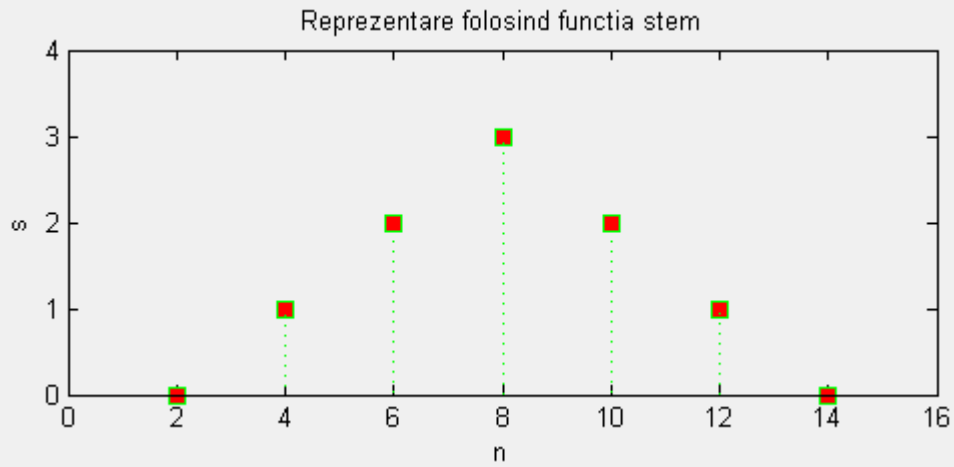
In *figure(1)* se va afișa următorul grafic.



- Fie vectorii `n = [2, 4, 6, 8, 10, 12, 14]` și `s = [0, 1, 2, 3, 2, 1, 0]`. Să se reprezinte grafic vectorul `s` în funcție de vectorul `n` folosind funcția `stem`. Reprezentarea se va face cu linie punctată verde, markere pătrate verzi umplute cu roșu.

```
n = 2 : 2 : 14; % numerele de la 2 la 14 din 2 in 2
s = [0, 1, 2, 3, 2, 1, 0];
figure(1)
stem(n,s,':sg','MarkerFaceColor','red','MarkerEdgeColor','green')
title('Reprezentare folosind functia stem')
xlabel('n'), ylabel('s'), axis([0, 16, 0, 4])
```

In *figure(1)* se va afișa următorul grafic.



9.3 Reprezentarea graficelor în aceeași figură, în același sistem de coordonate

Pentru a reprezenta mai multe grafice în aceeași figură, în același sistem de coordonate, există mai multe modalități:

1. Funcția PLOT utilizată pentru reprezentarea în același sistem de coordonate a mai multor grafice.

Sintaxă: `plot(x1, y1, s1, x2, y2, s2, xN, yN, sN)`

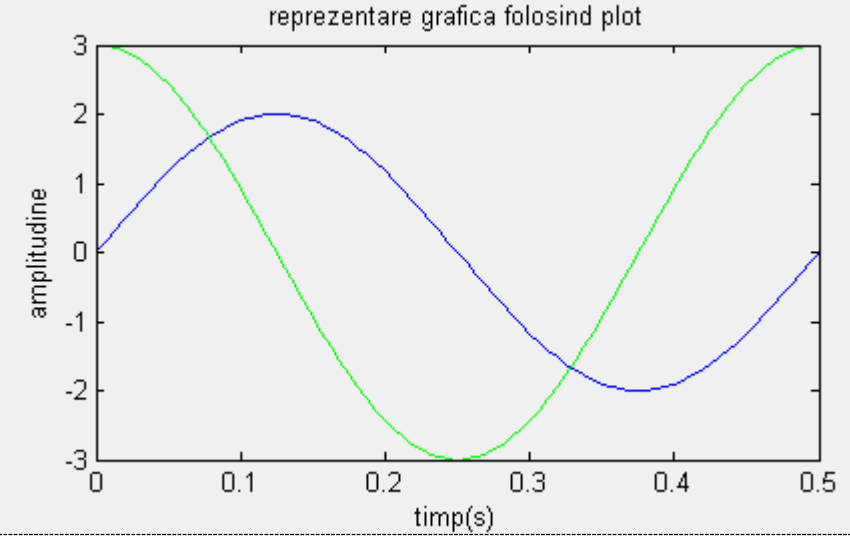
Pentru fiecare grafic în parte se poate specifica, între două semne apostrof, stilul dorit. Cu această metodă toate graficele sunt reprezentate doar cu `plot`.

- Să se reprezinte grafic două sinusoide[1], astfel:

- sinusoida `s1` de amplitudine 2, frecvență 2Hz și fază inițială nulă
- sinusoida `s2` de amplitudine 3, frecvență 2Hz și fază inițială π/2

```
t = 0:1/160:0.5;
s1 = 2*sin(2*pi*2*t);
s2 = 3*sin(2*pi*2*t + pi/2);
figure(2)
plot(t,s1,'b',t,s2,'g')
title('reprezentare grafica folosind plot')
xlabel('timp(s)')
ylabel('amplitudine')
```

In *figure(1)* se va afișa următorul grafic.



[1] Formula unui semnal sinusoidal este $x(t) = A \cdot \sin(2\pi \cdot F \cdot t + \varphi_0)$ unde A = amplitudinea maximă, F = frecvența, φ_0 = faza inițială. În urma conversiei analog numerice rezultă un semnal discret/numeric având formula $x[n] = A \cdot \sin(2\pi \cdot n \cdot \frac{F}{Fs} + \varphi_0)$.

2. Instrucțiunile HOLD ON ... HOLD OFF pentru reprezentarea în același sistem de coordonate a mai multor grafice.

```
hold on
      [reprezentare grafic_1]
      [reprezentare grafic_2]
      ....
      [reprezentare grafic_n]
hold off
```

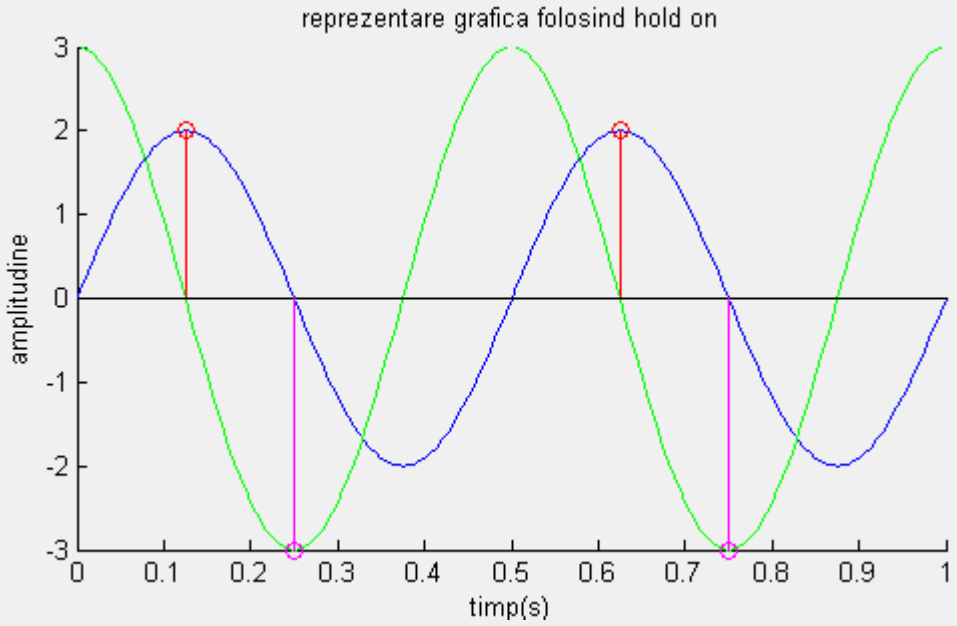
Spre deosebire de metoda prezentată anterior, perechea de instrucțiuni `hold on...hold off` are avantajul că reprezentările grafice se pot face cu orice funcție, nu doar cu `plot`.

- Să se reprezinte grafic 4 semnale:

- sinusoida `s1` de amplitudine 2, frecvență 2Hz și fază inițială nulă (cu `plot`)
- sinusoida `s2` de amplitudine 3, frecvență 2Hz și fază inițială π/2 (cu `plot`)
- punctele de maxim pentru `s1` (cu `stem`)
- punctele de minim pentru `s2` (cu `stem`)

```
A1 = 2; A2 = 3;
F1 = 2; F2 = 2;
Fs = 160; durata = 1;
t = 0:1/Fs:durata;
s1 = A1*sin(2*pi*F1*t);
s2 = A2*sin(2*pi*F2*t + pi/2);

% in pozM se salveaza pozitiile punctelor de maxim din s1
[maxime, pozM] = find(s1==max(s1))
% in pozm se salveaza pozitiile punctelor de minim din s2
[minime, pozm] = find(s2==min(s2))

figure(1)
hold on
    plot(t,s1,'b')
    plot(t,s2,'g')
    stem(t(pozM), s1(pozM),'r')
    stem(t(pozm), s2(pozm),'m')
hold off
title('reprezentare grafica folosind hold on')
xlabel('timp(s)')
ylabel('amplitudine')
```

In *figure(1)* se va afișa următorul grafic.



Observație: pentru o ușoară identificare a graficelor se poate folosi funcția `legend`. Dacă într-o fereastră, în același sistem de coordonate sunt reprezentate de exemplu 2 grafice, se folosește funcția `legend` cu sintaxa:

```
legend(nume_grafic1,nume_grafic2,'Location',POZ)
```

unde POZ reprezintă localizarea legendei în spațiul figurii. POZ poate fi:

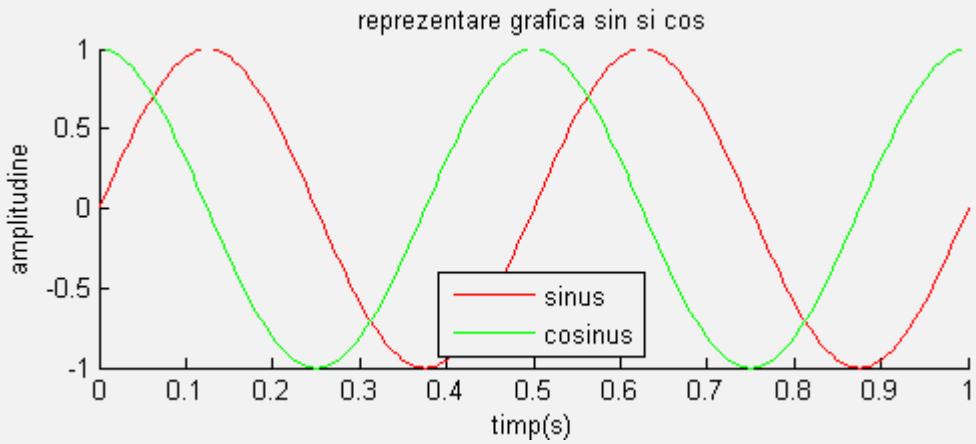
'North'	în interiorul graficului în partea de sus
'South'	în interiorul graficului în partea de jos
'East'	în interiorul graficului în partea dreaptă
'West'	în interiorul graficului în partea stângă
etc	

- Să se reprezinte grafic un sinus și un cosinus și să se afișeze semnificația graficelor folosind funcția `legend`.

```
t = 0:1/160:1;
s1 = sin(2*pi*2*t);
s2 = cos(2*pi*2*t);
figure(1)
hold on
  plot(t, s1, 'r')
  plot(t, s2, 'g')
hold off
legend('sinus', 'cosinus', 'Location', 'South')
title('reprezentare grafica sin si cos')
xlabel('timp(s)')
```

```
ylabel('amplitudine')
```

In *figure(1)* se va afișa următorul grafic.



9.4 Reprezentarea graficelor în aceeași figură, în sisteme de coordonate diferite

Pentru a reprezenta mai multe grafice în aceeași figură, dar în sisteme de coordonate diferite se folosește funcția `subplot`.

Sintaxa: `subplot(m,n,p)`

Rezultatul constă în împărțirea figurii într-o matrice cu `m` linii și `n` coloane. Parametrul `p` reprezintă indicele de ordine al graficelor, numerotarea făcându-se de la stânga la dreapta și de sus în jos.

Pentru a reprezenta într-o figură 6 grafice organizate pe 2 linii și 3 coloane, structura este următoarea:

`subplot(2,3,1)`	`subplot(2,3,2)`	`subplot(2,3,3)`
`subplot(2,3,4)`	`subplot(2,3,5)`	`subplot(2,3,6)`

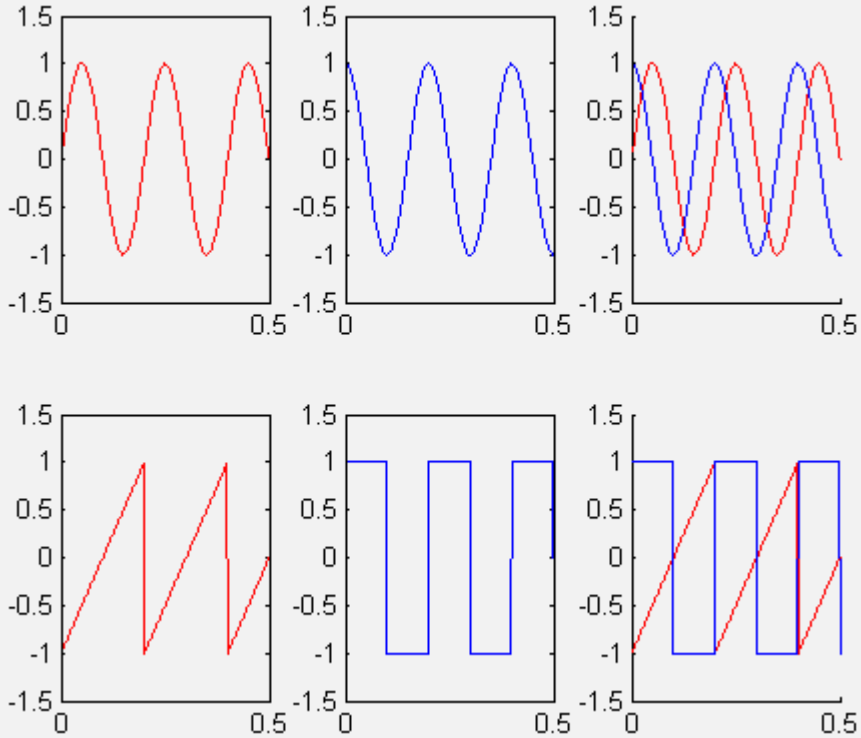
Observație: `subplot` indică doar zona în care se face afișarea, nu face și afișarea. Pentru afișare se folosește una dintre funcțiile `plot`, `stem` etc.

- Să se reprezinte 6 grafice pe 2 linii și 3 coloane, astfel:

Sinus	Cosinus	Sinus și cosinus
Semnal triunghiular	Semnal dreptunghiular	Semnal triunghiular și dreptunghiular

```
F = 5; Fs = 1000;
D = 0.5;
t = 0 : 1/Fs : D;
s1 = sin(2*pi*F*t);       % sinus
s2 = cos(2*pi*F*t);       % cosinus
s3 = sawtooth(2*pi*F*t); % semnal triunghiular
s4 = square(2*pi*F*t);    % semnal dreptunghiular

figure(1)
subplot(2,3,1),plot(t,s1,'r'), axis([0, D, -D-1, D+1])
subplot(2,3,2),plot(t,s2,'b'), axis([0, D, -D-1, D+1])
subplot(2,3,3),
    hold on
        plot(t,s1,'r'),plot(t,s2,'b'), axis([0, D, -D-1, D+1])
    hold off
    axis([0, D, -D-1, D+1])
subplot(2,3,4), plot(t,s3,'r'), axis([0, D, -D-1, D+1])
subplot(2,3,5), plot(t,s4,'b'), axis([0, D, -D-1, D+1])
subplot(2,3,6)
    hold on
        plot(t,s3,'r'),plot(t,s4,'b'), axis([0, D, -D-1, D+1])
    hold off
```

In *figure(1)* se vor afișa următoarele grafice.



9.5 Reprezentarea procentuală a datelor folosind funcția `pie`

Funcția `pie` este utilizată pentru reprezentarea procentuală a datelor sub forma unui disc.

Sintaxă: `pie([val1,val2,…, valN],{'exp1','exp2',…,'expN'})`

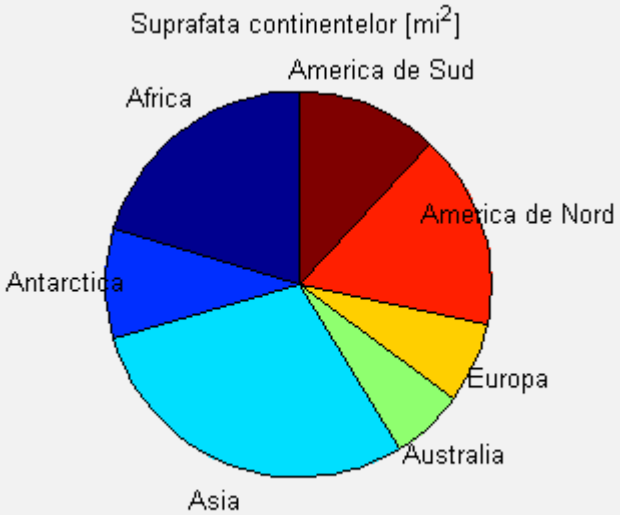
Observație: Pentru a determina aria ocupată de valoarea `valN` se calculează $valN/(val1 + val2 + \cdots + valN)$ iar în dreptul suprafeței ocupate de `valN` va fi titlul `expN`.

- Folosind funcția `pie` să se reprezinte grafic următoarele date.

Continent	Suprafață [mi^2]
Africa	11,730,000
Antarctica	5,300,000
Asia	16,920,000
Australia	3,478,200
Europa	3,930,000
America de Nord	9,460,000
America de Sud	6,890,000

```
suprafata = [11730000, 5300000, 16920000, 3478200, 3930000,
9460000, 6890000];
continente = {'Africa', 'Antarctica', 'Asia', 'Australia',...
'Europa', 'America de Nord', 'America de Sud'}
figure(1)
pie(suprafata, continente)
title('Suprafata continentelor [mi^2]')
```

In *figure(1)* se va afișa următorul grafic.



9.6 Reprezentarea datelor folosind funcția `bar`

Sintaxă: `bar(X, Y)` unde `X` este un vector și `Y` este o matrice. Elementul `Y(i,j)` este reprezentat grafic sub formă de bară la poziția specificată de `X(i)`.

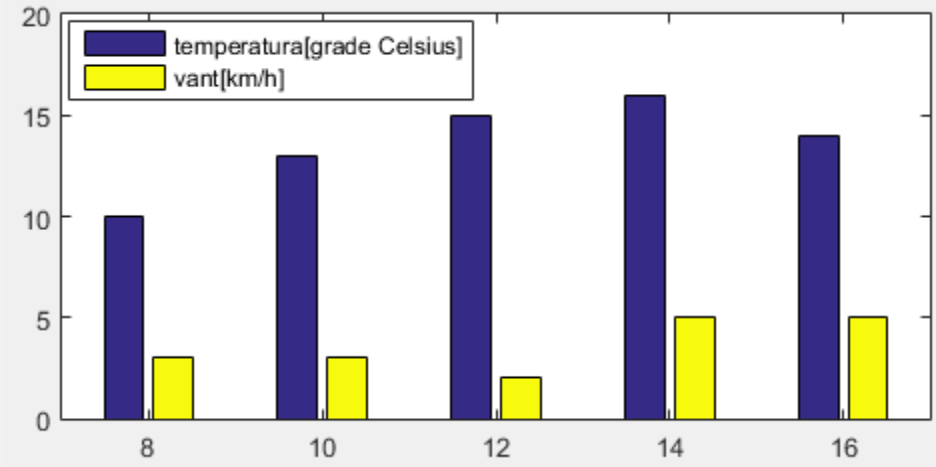
- Fie următoarele înregistrări meteorologice:

Ora	Temp[C°]	Vânt[km/h]
8	10	3
10	13	3
12	15	2
14	16	5
16	14	5

Să se reprezinte grafic pentru fiecare oră înregistrările obținute (temperatură, vânt)

```
X = [8 10 12 14 16];
Y = [10 3; 13 3; 15 2; 16 5; 14 5];
bar(X, Y)
legend('temperatura[grade Celsius]','vant[km/h]')
```

In *figure(1)* se va afișa următorul grafic.



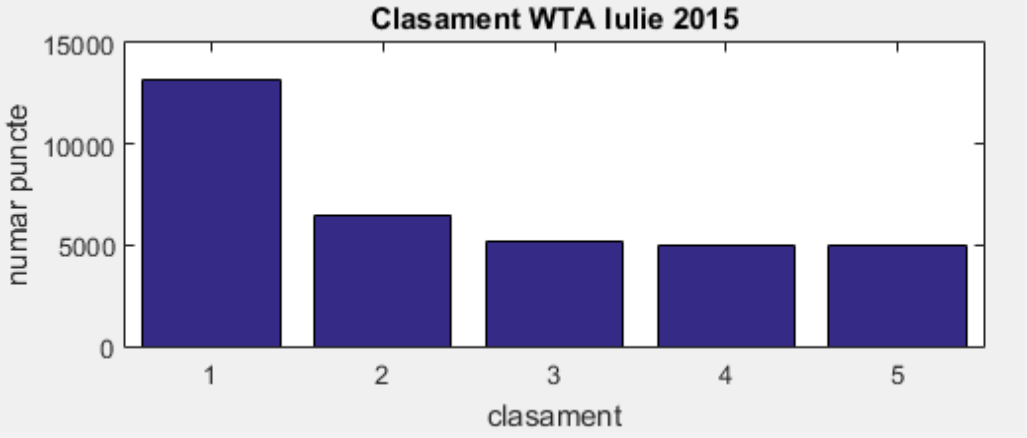
Dacă se folosește simplu sintaxa `bar(Y)`, se consideră că X = `[1, 2, 3,...]`.

- Folosind funcția `bar` să se reprezinte grafic următoarele date.

Poziție	Clasament WTA (Iulie 2015)	Număr puncte
1	Serena Williams (USA)	13161
2	Maria Sharapova (RUS)	6490
3	Simona Halep (ROU)	5151
4	Caroline Wozniacki (DEN)	5000
5	Petra Kvitova (CZE)	5000

```
puncte = [13161,6490,5151,5000,5000];
figure(1)
bar(puncte)
title('Clasament WTA Iulie 2015')
xlabel('clasament'), ylabel('numar puncte')
```

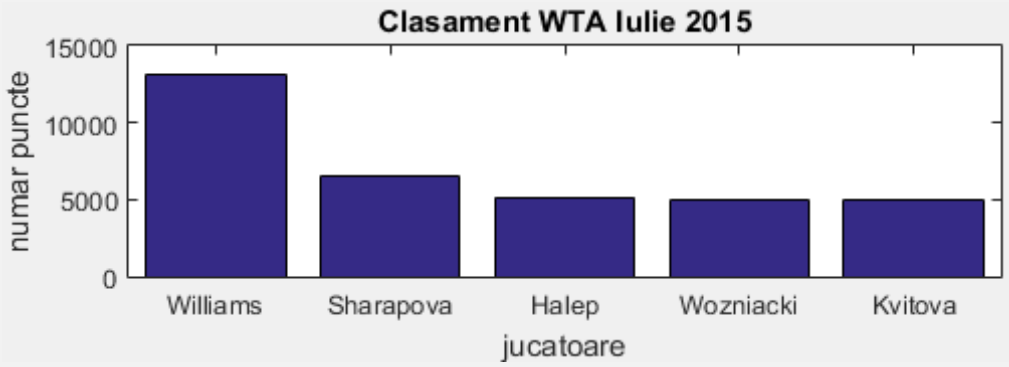
In *figure(1)* se va afișa următorul grafic.



Dacă se dorește ca pe axa *Ox* să se afișeze numele jucătoarelor, atunci se va folosi proprietatea `xticklabel`, astfel:

```
puncte = [13161,6490,5151,5000,5000];
nume = {'Williams','Sharapova','Halep','Wozniacki','Kvitova'};
figure(1)
bar(puncte);
set(gca,'xticklabel', nume)
title('Clasament WTA Iulie 2015')
xlabel('jucatoare'), ylabel('numar puncte')
```

In *figure(1)* se va afișa următorul grafic.



9.7 Reprezentarea histogramei folosind funcția `hist`

Funcția `hist` este utilizată pentru a reprezenta histograma unui semnal. Histograma este utilă pentru a vedea cum sunt distribuite valorile unui semnal și este des utilizată în statistică, procesarea imaginilor etc.

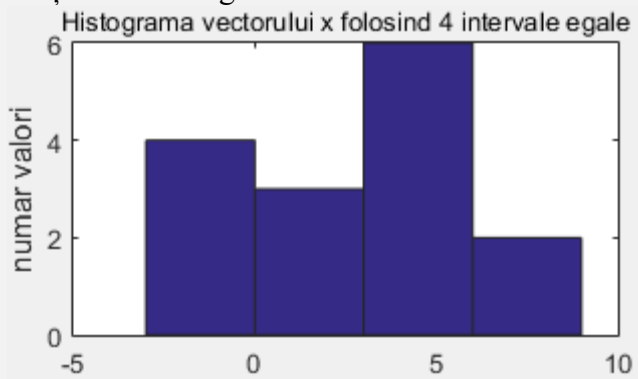
Sintaxă: `hist(x, M)`

Histograma împarte domeniul de valori al unui semnal *x* în *M* intervale egale și întoarce numărul de valori din fiecare interval. Cu alte cuvinte axa *Ox* a histogramei reflectă intervalul de valori al semnalului *x*, iar axa *Oy* reflectă numărul de elemente care intră în cadrul fiecărui interval. Reprezentarea grafică se face de obicei sub forma unor bare verticale.

Fie semnalul *x* = [-2, 1, 6, 4, 0, 2, 6, 4, 5, -2, 3, 4, -3, 8, 9]. Se dorește vizualizarea histogramei în cazul în care se împarte domeniul de valori în *M* = 4 intervale egale. Valoarea minimă a semnalului *x* este x_{min} = -3 iar valoarea maximă este x_{max} = 9. Împărțindu-se intervalul [-3, 9] în *M* = 4 intervale egale, vor rezulta intervalele: [-3, 0], (0, 3], (3, 6] și (6, 9]. În intervalul [-3, 0] sunt 4 valori, în intervalul (0, 3] sunt 3 valori, în intervalul (3, 6] sunt 6 valori iar în intervalul (6, 9] sunt 2 valori.

```
M = 4;
x = [-2, 1, 6, 4, 0, 2, 6, 4, 5, -2, 3, 4, -3, 8, 9];
figure(1), hist(x, M)
title('Histograma vectorului x pe 4 intervale egale')
ylabel('numar valori')
```

In *figure(1)* se va afișa următorul grafic.



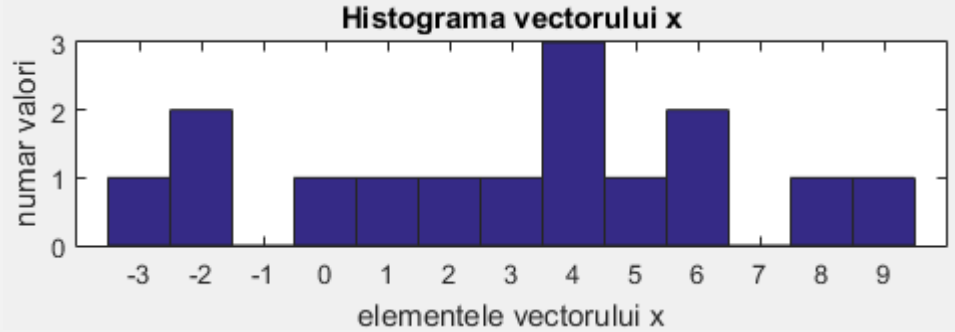
Dacă *M* este un vector, numărul de intervale ale histogramei va fi egal cu lungimea vectorului *M*, iar valorile de referință ale intervalelor vor fi valorile din vectorul *M*.

Particularizând, pentru un vector ce conține numai elemente numere întregi, histograma poate fi utilizată și pentru a afla de câte ori se repetă fiecare valoare a semnalului *x*, folosind sintaxa `hist(x, min(x):max(x))`

- Fie semnalul *x* = [-2, 1, 6, 4, 0, 2, 6, 4, 5, -2, 3, 4, -3, 8, 9]. Să se afișeze de câte ori apare fiecare valoare.

```
x = [-2, 1, 6, 4, 0, 2, 6, 4, 5, -2, 3, 4, -3, 8, 9];
figure(1), hist(x, min(x):max(x))
title('Histograma vectorului x')
xlabel('elementele vectorului x'), ylabel('numar valori')
```

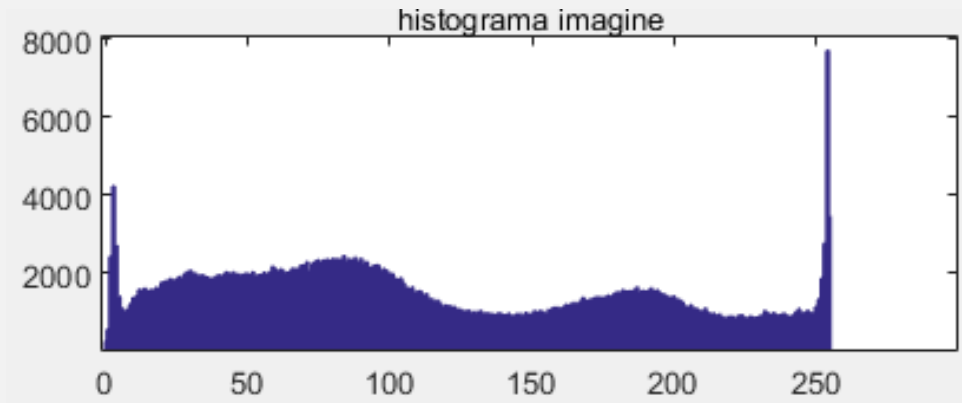
In *figure(1)* se va afișa următorul grafic.



- Să se afișeze histograma unei imagini grayscale

```
imag = imread('catedrala.jpg');
figure(1),hist(imag(:), 0:255)
title('histograma imagine')
```

In *figure(1)* se va afișa următorul grafic.



Observație: pentru histograma unei imagini se poate folosi funcția `imhist` (a se vedea *12.2.3 Histograma unei imagini*)

9.8 Reprezentare grafică în spațiul 3-D

În MATLAB, se folosesc cel mai adesea funcțiile `mesh` și `surf` pentru reprezentarea grafică a unei funcții de forma *z = f(x, y)*.

Funcțiile `mesh` și `surf`

Sintaxă : `mesh(X, Y, Z), surf(X, Y, Z)`

`X`, `Y` și `Z` sunt matrice având aceleași dimensiuni. Funcția `mesh` trasează rețeaua de linii definită de punctele de coordonate (`x, y, z`). Funcția `surf`, în plus față de funcția `mesh`, reprezintă grafic și suprafața determinată de rețeaua de linii definită de punctele de coordonate (`x, y, z`).

Observație: se pot folosi și sintaxele `mesh(x,y,Z)` și `surf(x,y,Z)` unde `x` și `y` sunt vectori și `Z` este matrice cu proprietatea că `n = length(x)`, `m = length(y)` și `[m,n]=size(Z)`.

- Pentru a înțelege mai bine cum funcționează reprezentarea 3-D, vom începe cu un exemplu simplu. Fie vectorii: `x` = [1, 2, 3, 4, 5, 6, 7, 8, 9], `y` = [3, 4 ,5 ,6 ,7, 8, 9] și `Z = f(x,y)`, unde $f(x,y) = 25 - (x-5)^2 - (y-6)^2$.

Pentru `x = 1` și `y = 3` ➔ `z = 0`
Pentru `x = 1` și `y = 4` ➔ `z = 5`
Pentru `x = 1` și `y = 5` ➔ `z = 8`
.....
Pentru `x = 1` și `y = 9` ➔ `z = 0`

În *Figura 9.1*, s-au reprezentat grafic toate perechile (`x,y`) în spațiul 2-D.

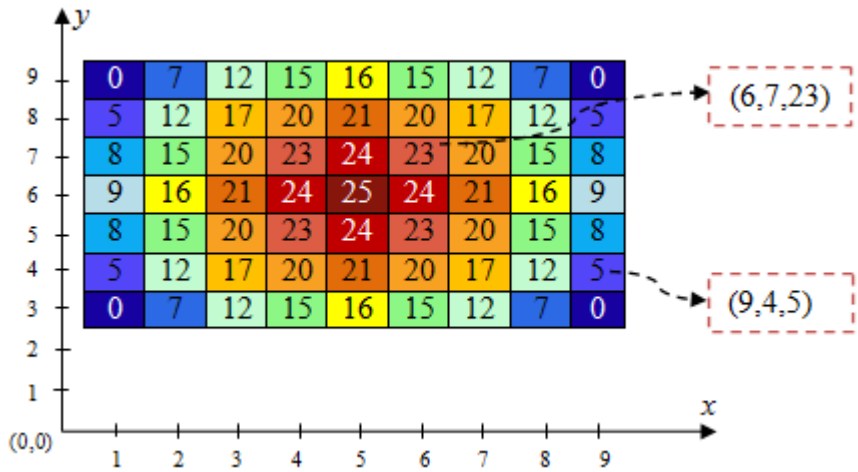


Figura 9.1. Reprezentarea grafică a funcției $f(x,y) = 25 - (x-5)^2 - (y-6)^2$

Pentru a reprezenta în spațiul 3-D, se va mai ridica și axa *Oz*, perpendiculară pe planul (*xOy*). În cazul de față pe axa *Oz* vor fi valori între 0 și 25. Pentru a putea realiza implementarea grafică în spațiul 3-D, folosind funcțiile `mesh` sau `surf` există două posibilități:

Caz 1. Matricele `X`, `Y` și `Z` să aibă aceleași dimensiuni. Pentru exemplul de față, `x` ia valorile [1, 2, 3, 4, 5, 6, 7, 8, 9]. Pentru fiecare valoare a lui `x`, `y` poate fi [3, 4 ,5 ,6 ,7, 8, 9]. Ca urmare:

- se va construi matricea `X` care va avea pe fiecare linie vectorul `x`, iar numărul de linii va fi egal cu numărul de elemente din `y`.
- se va construi matricea `Y` care va avea pe fiecare coloană vectorul `y`, iar numărul de coloane va fi egal cu numărul de elemente din `x`.

Pentru exemplul de față, matricele `X` și `Y` vor avea 7 linii și 9 coloane.

Matricea Z va avea aceleași dimensiuni ca și X și Y. Elementele matricei Z se calculează cu formula: `Z(m,n) = f(X(m,n), Y(m,n))`, unde `m` și `n` reprezintă indicele liniei respective coloanei.

`Z(3,2) = f(X(3,2),Y(3,2))=` $25-(2-5)^2-(5-6)^2=$ `15`

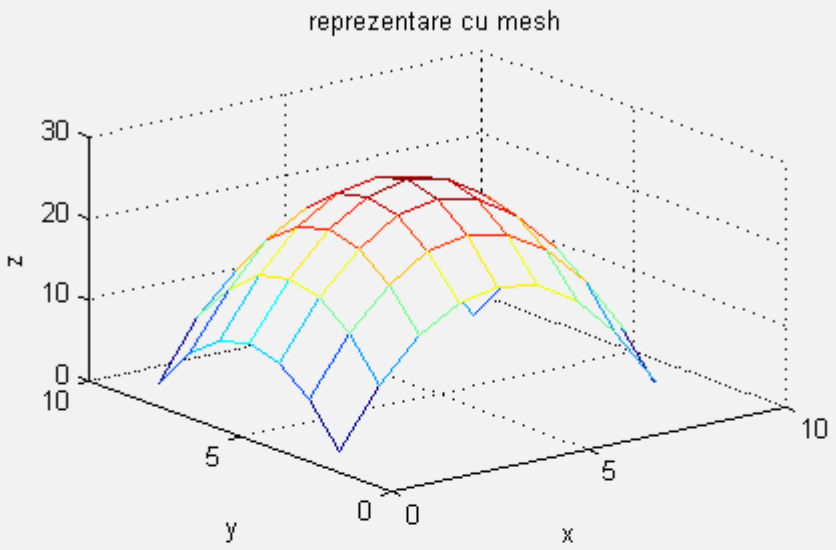
$$X=\begin{bmatrix}1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\\1&2&3&4&5&6&7&8&9\end{bmatrix}\quad Y=\begin{bmatrix}3&3&3&3&3&3&3&3&3\\4&4&4&4&4&4&4&4&4\\5&5&5&5&5&5&5&5&5\\6&6&6&6&6&6&6&6&6\\7&7&7&7&7&7&7&7&7\\8&8&8&8&8&8&8&8&8\\9&9&9&9&9&9&9&9&9\end{bmatrix}$$

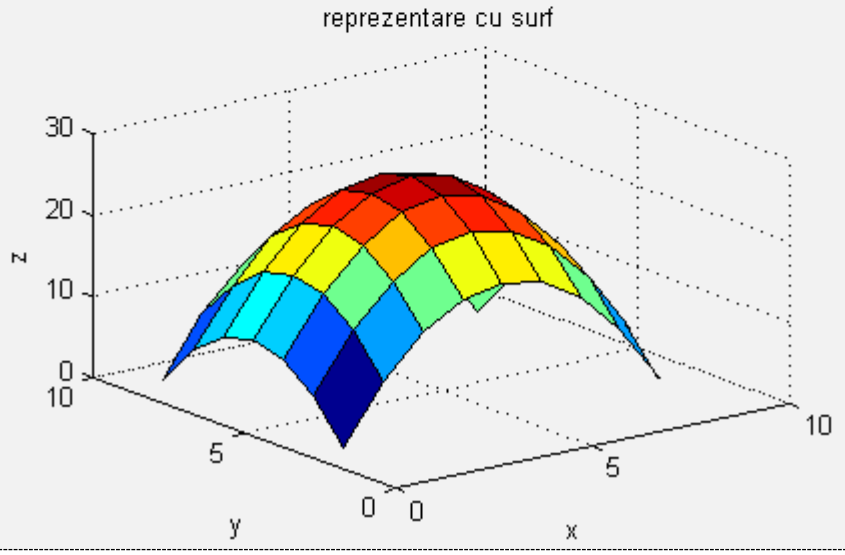
$$Z=\begin{bmatrix}0&7&12&15&16&15&12&7&0\\5&12&17&20&21&20&17&12&5\\8&15&20&23&24&23&20&15&8\\9&16&21&24&25&24&21&16&9\\8&15&20&23&24&23&20&15&8\\5&12&17&20&21&20&17&12&5\\0&7&12&15&16&15&12&7&0\end{bmatrix}$$

Implementare în MATLAB (matricele `X`, `Y` și `Z` au aceleași dimensiuni)

```
x = 1 : 9;
y = 3 : 9;
% se formeaza matricea X ce contine pe fiecare linie vectorul x
X = repmat(x, length(y), 1);
% matricea Y contine pe fiecare coloana vectorul y
Y = repmat(y',1, length(x));
% se calculeaza elemenetele matricei Z
Z = 25 - (X - 5).^2-(Y - 6).^2;
figure(1)
subplot(2,1,1), mesh(X,Y,Z), title('reprezentare cu mesh')
subplot(2,1,2), surf(X,Y,Z), title('reprezentare cu surf')
```

In *figure(1)* se vor afișa următoarele grafice.





Caz 2. `x` și `y` sunt vectori și `Z` este matrice cu proprietatea că `n = length(x)`, `m = length(y)` și `[m,n] = size(Z)`.

În cazul de față `x` = [1, 2, 3, 4, 5, 6, 7, 8, 9] și `y` = [3, 4 ,5 ,6 ,7, 8, 9] deci `n` = 9 și `m` = 7. Matricea `Z` va avea 7 linii și 9 coloane. Elementele matricei `Z` se vor calcula pentru toate combinațiile elementelor vectorilor `x` și `y`.

`Z(i,j) = f(x(j), y(i))`

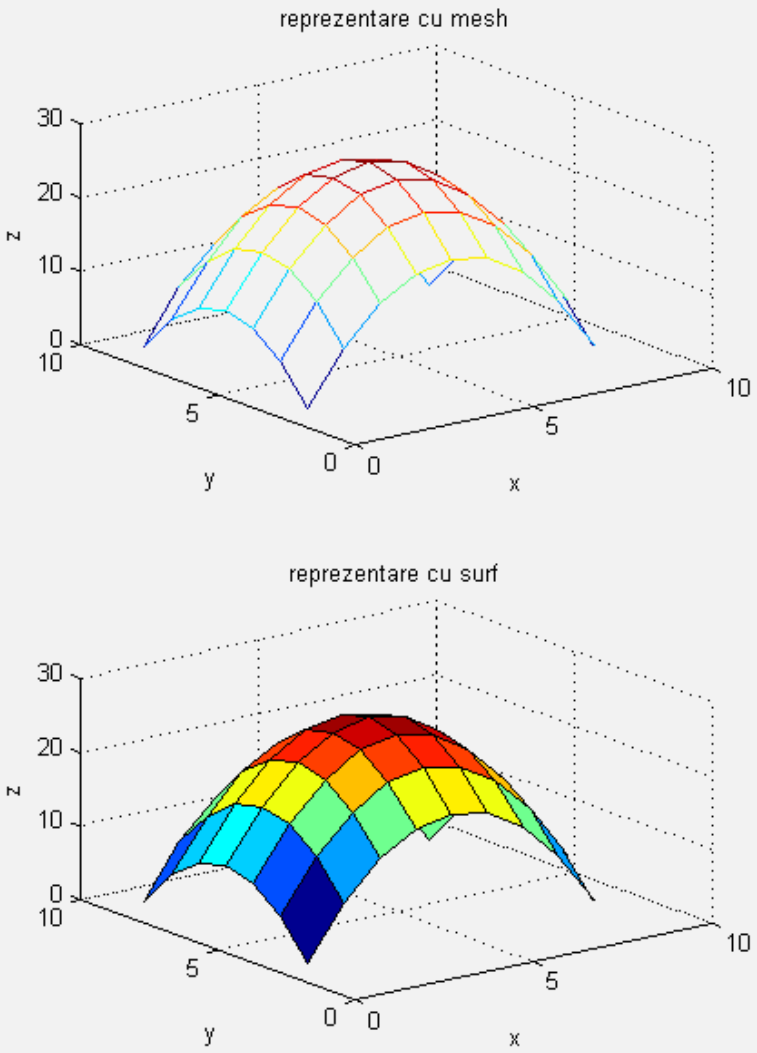
`Z(1,1) = f(x(1),y(1))=` $25-(1-5)^2-(3-6)^2=$ `0`

`Z(3,2) = f(x(2),y(3))=` $25-(2-5)^2-(5-6)^2=$ `15`

Matricea `Z` rezultată va fi identică cu cea din cazul anterior.

```
x = 1 : 9;
y = 3 : 9;
for i = 1:length(y)
    for j = 1: length(x)
        Z(i,j) = 25 - (x(j) - 5)^2-(y(i) - 6)^2;
    end
end
figure(1)
subplot(2,1,1), mesh(x,y,Z), title('reprezentare cu mesh')
subplot(2,1,2), surf(x,y,Z), title('reprezentare cu surf')
```

In *figure(1)* se va afișa următorul grafic.



9.8.1 Proprietăți ale funcției `surf`

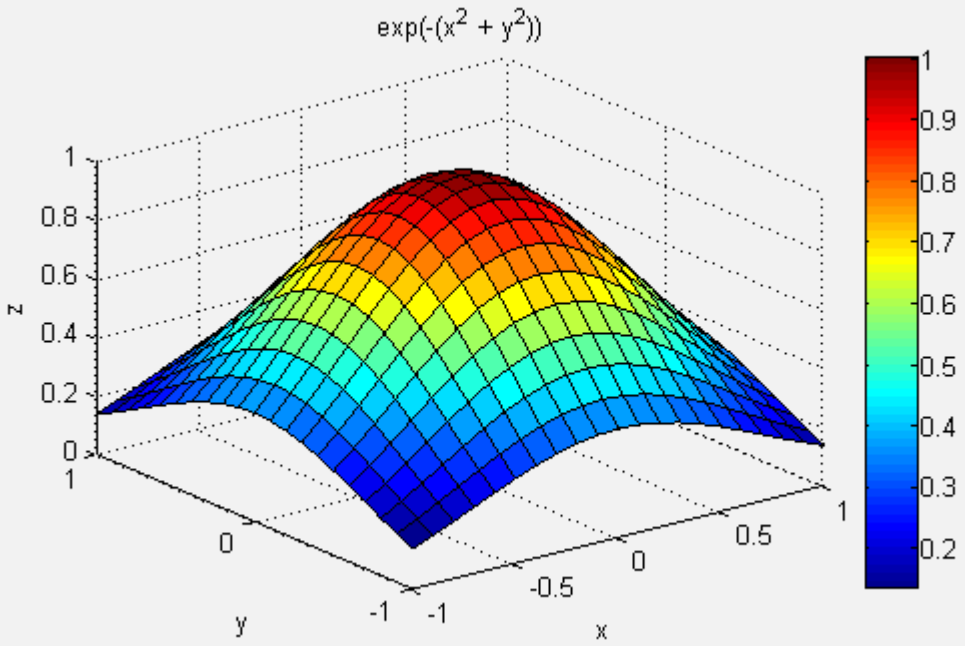
Funcția `surf` a fost folosită în exemplele anterioare cu parametri impliciți. Aceștia sunt: culoarea neagră pentru trasarea contururilor fețelor, suprafață opacă, culoarea suprafeței variază între albastru și vișiniu, în funcție de domeniul de valori etc.

- Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1, 1]$ și $y \in [-1, 1]$.

Se va folosi funcția `surf` cu parametri impliciți.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));

figure(1)
surf(X,Y,Z), title('exp(-(x^2 + y^2))'), colorbar
xlabel('x'), ylabel('y'), zlabel('z')
```

In *figure(1)* se va afișa următorul grafic.



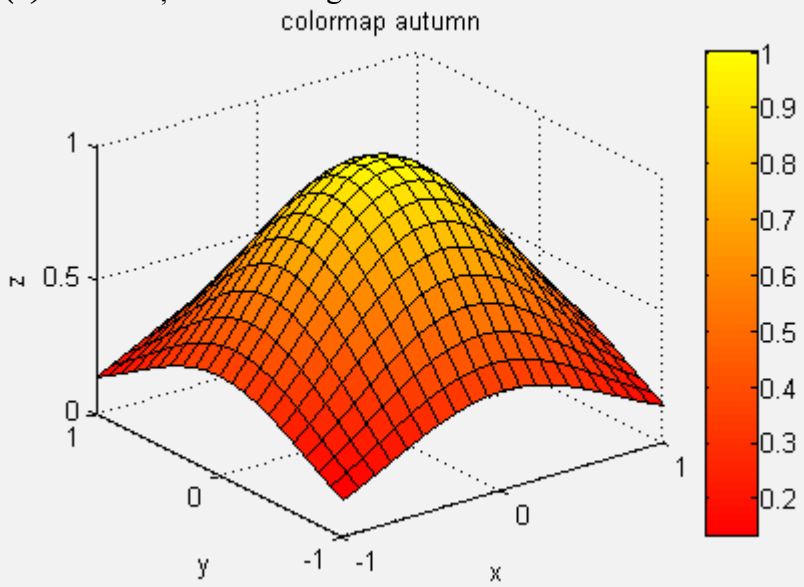
Funcția `surf` are însă numeroase proprietăți pentru culoarea suprafeței, transparență, iluminare etc. În continuare sunt date câteva exemple de folosire a acestor proprietăți.

- **Culoare.** O suprafață este formată din mai multe fețe rectangulare. Culoarea unei fețe este determinată de valorile matricei `Z` și de harta de culori `(colormap)`. Există mai multe hărți de culori predefinite cum ar fi `'default','spring','summer','autumn','winter','pink'` etc.

Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1,1]$ și $y \in [-1,1]$. Se va folosi funcția `surf` având harta de culori predefinită `'autumn'`.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));

figure(1)
surf(X,Y,Z), title('colormap autumn')
map = 'autumn';
colormap(map), colorbar
```

In *figure(1)* se va afișa următorul grafic.



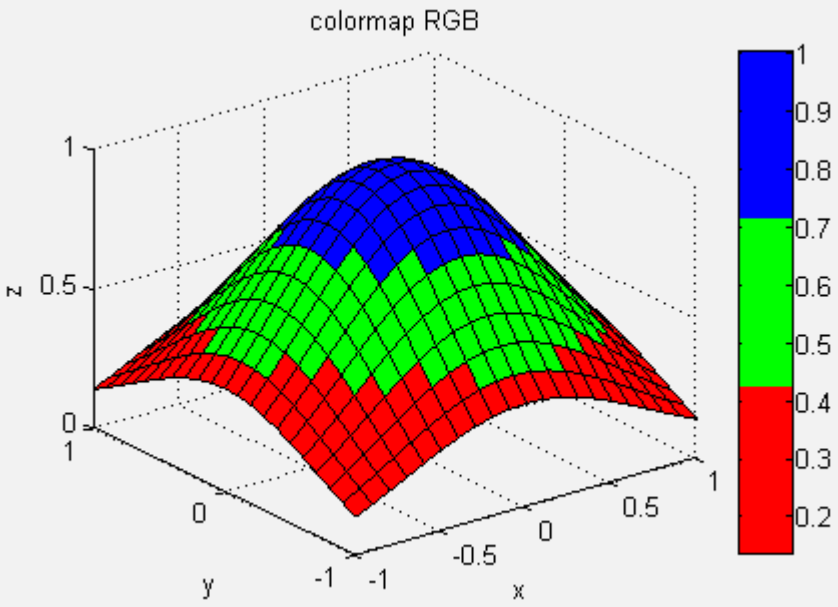
Se pot defini însă și hărți de culori personalizate, astfel: dacă se dorește folosirea a `N` culori, se va defini o matrice cu `N` linii și 3 coloane. Fiecare linie va conține codul RGB al culorii dorite. MATLAB-ul folosește această hartă de culori astfel: împarte domeniul de valori în `N` intervale egale și pentru cele mai mici valori ale lui z se va folosi culoarea definită pe prima linie, apoi pentru următoarele valori se folosește culoarea de pe linia a doua și tot așa, cele mai mari valori ale lui z fiind colorate cu culoarea definită pe ultima linie a hărții de culori.

- Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1, 1]$ și $y \in [-1, 1]$.

Se va folosi funcția `surf` având harta de culori formată din trei culori: roșu, verde și albastru.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));

figure(1)
surf(X,Y,Z), title('colormap RGB')
map = [1 0 0;
       0 1 0;
       0 0 1];
colormap(map), colorbar
```

In *figure(1)* se va afișa următorul grafic.



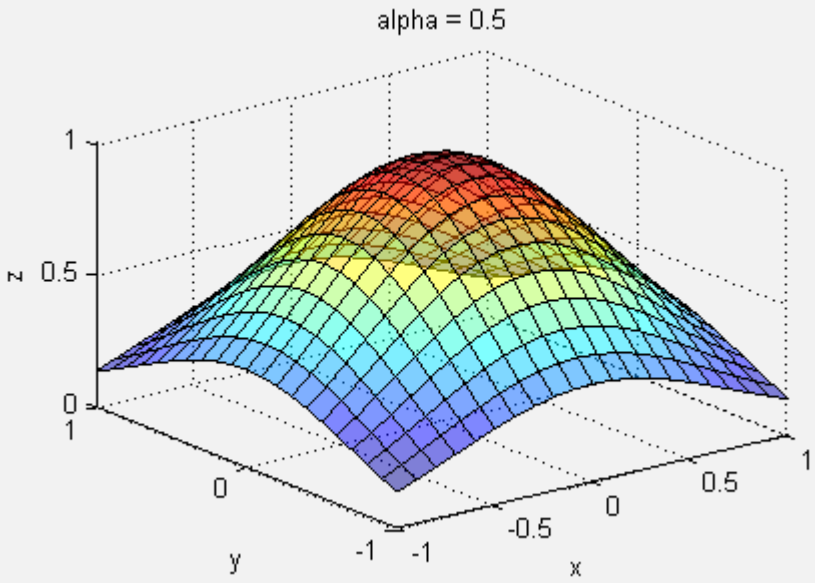
- **Transparență**. Pentru a seta transparența se folosește proprietatea *alpha*, ce poate avea o valoare între 0 și 1, unde 0 reprezintă transparență totală și 1 reprezintă opacitate totală.

- Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1, 1]$ și $y \in [-1, 1]$.

Se va folosi funcția `surf` cu transparență 0.5.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));

figure(1)
surf(X,Y,Z), title('alpha = 0.5')
alpha(0.5)
xlabel('x'), ylabel('y'), zlabel('z')
```

In *figure(1)* se va afișa următorul grafic.



- **Efect de iluminare.** Acest efect poate fi aplicat unei suprafețe folosind proprietatea `camlight`.

Exemple: `camlight('right')` produce iluminare din dreapta

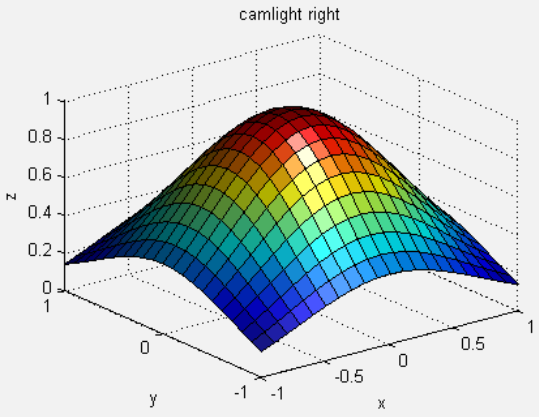
`camlight('left')` produce iluminare din stânga

- Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1,1]$ și $y \in [-1,1]$.

Se va folosi funcția `surf` cu efect de iluminare din dreapta.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));
```

```
figure(1)
surf(X,Y,Z), title('camlight right')
camlight('right')
```
In *figure(1)* se va afișa următorul grafic.



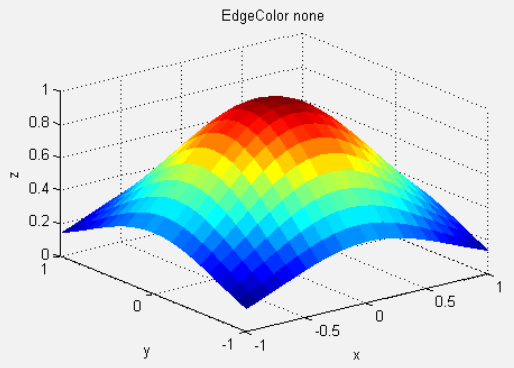
- **Culoarea liniilor ce determină suprafața.** Se poate seta folosind proprietatea `EdgeColor`.

Exemple: dacă `EdgeColor` este `'none'` atunci nu se vor trasa liniile
dacă `EdgeColor` este `'r'` atunci se vor trasa liniile cu roșu

Să se reprezinte funcția $z = e^{-(x^2+y^2)}$, pentru $x \in [-1,1]$ și $y \in [-1,1]$.

Se va folosi funcția `surf` fără a se trasa liniile fețelor rectangulare.

```
x = -1 : 0.1: 1;
y = -1 : 0.1: 1;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = exp(-(X.^2+Y.^2));
figure(1)
surf(X,Y,Z, 'EdgeColor','none'), title('EdgeColor none')
xlabel('x'), ylabel('y'), zlabel('z')
```
In *figure(1)* se va afișa următorul grafic.

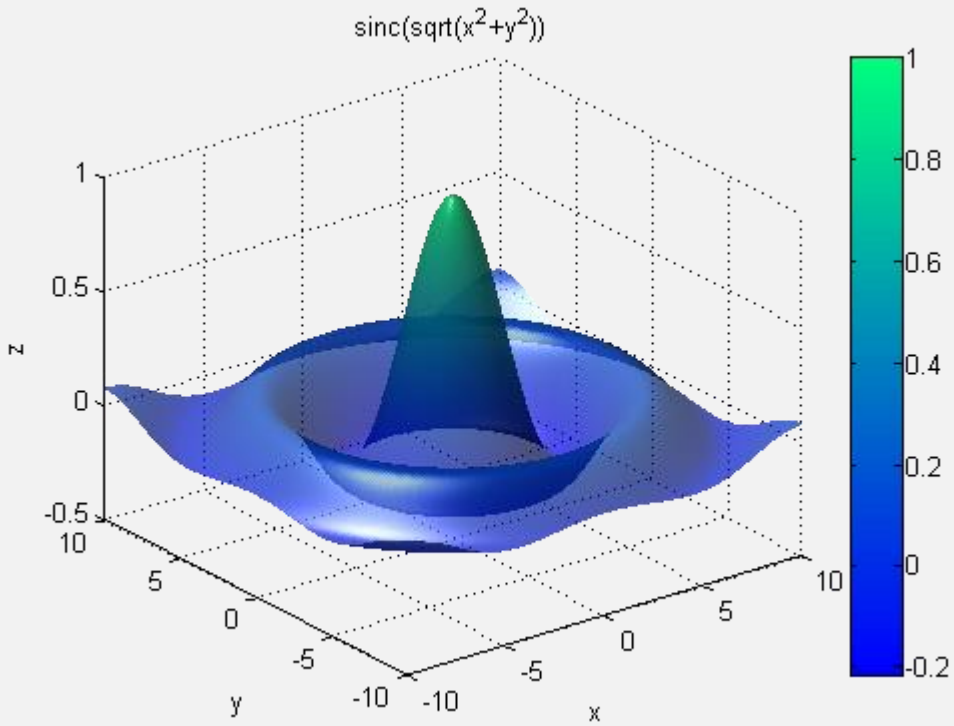


Să se reprezinte grafic funcția $z = \sin\left(\sqrt{x^2 + y^2}\right) / \sqrt{x^2 + y^2}$, pentru $x \in [-10, 10]$ și $y \in [-10, 10]$. Se va folosi funcția `surf` cu următorii parametri: transparență 0.7, iluminare din stânga, colormap `winter`, lipsă rețea contururi fețe rectangulare.

Obs: Pentru `x = 0` și `y = 0` ➔ `z = 1`.

```
x = -10 : 0.1: 10;
y = -10 : 0.1: 10;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
% se adauga valoarea eps pentru a evita impartirea la zero
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;

figure(1)
surf(X,Y,Z,'EdgeColor','none'), title('sinc(sqrt(x^2+y^2))')
xlabel('x'), ylabel('y'), zlabel('z')
camlight('left')
map = 'winter';
colormap(map), colorbar
alpha(0.7);
```

In *figure(1)* se va afișa următorul grafic.



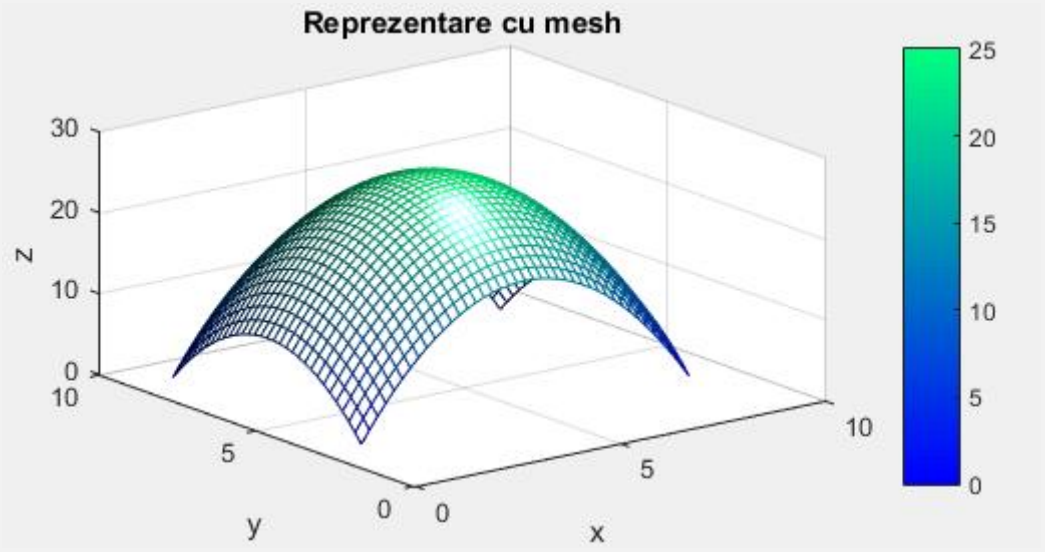
9.8.2 Proprietăți ale funcției `mesh`

Funcția `mesh` a fost folosită în exemplele anterioare cu parametri impliciți. Însă ca și în cazul funcției `surf` se poate schimba culoarea liniilor, dimensiunea liniilor, iluminarea etc.

Să se reprezinte grafic funcția $f(x,y) = 25 - (x-5)^2 - (y-6)^2$, pentru $x \in [1,\ 9]$ și $y \in [3,\ 9]$. Se va folosi funcția `mesh` cu următorii parametri: iluminare din dreapta, colormap `winter`, grosimea liniilor 0.5.

```
x = 1 : 0.2 : 9;
y = 3 : 0.2 : 9;
X = repmat(x, length(y), 1);
Y = repmat(y',1, length(x));
Z = 25 - (X - 5).^2-(Y - 6).^2;
figure(1)
mesh(X,Y,Z,'LineWidth',0.5), title('Reprezentare cu mesh')
xlabel('x'), ylabel('y'), zlabel('z')
camlight('right')
map = 'winter';
colormap(map), colorbar
```

In *figure(1)* se va afișa următorul grafic.



10. Interfață grafică în MATLAB
GUI - Graphical User Interface

Dezvoltarea unei interfețe grafice pentru un program permite modificarea parametrilor programului într-un mod mai accesibil, un program cu o interfață grafică având avantajul unei utilizări mai simple. Utilizatorul poate folosi interfața fără a fi nevoit să cunoască limbajul în care a fost dezvoltată aplicația.

Pentru a deschide o interfață grafică în MATLAB se tastează comanda *guide* în fereastra *Command Window* (sau din *Home* se selectează *New/ Graphical User Interface*). În continuare sunt două opțiuni:

- *Create New GUI/Blank GUI (Default)*: pentru a porni o nouă interfață
- *Open Existing GUI:* pentru a deschide o interfață deja existentă.

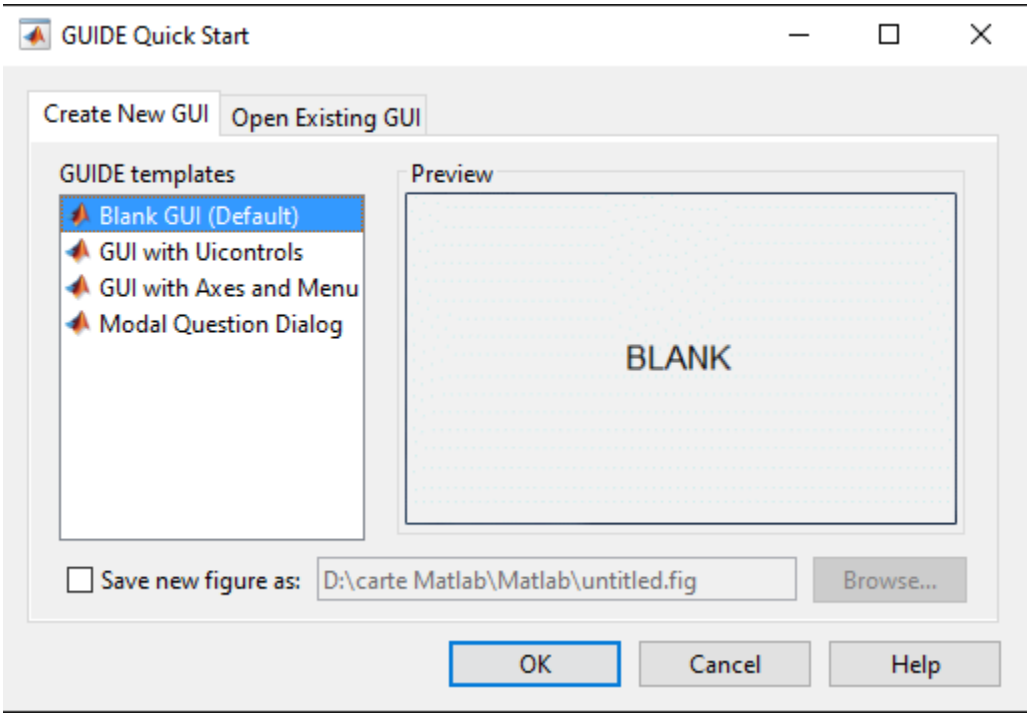


Figura 10.1. Deschiderea unei interfețe grafice

O interfață conține o zonă de lucru (centrală) și o zonă cu obiecte grafice (în stânga).

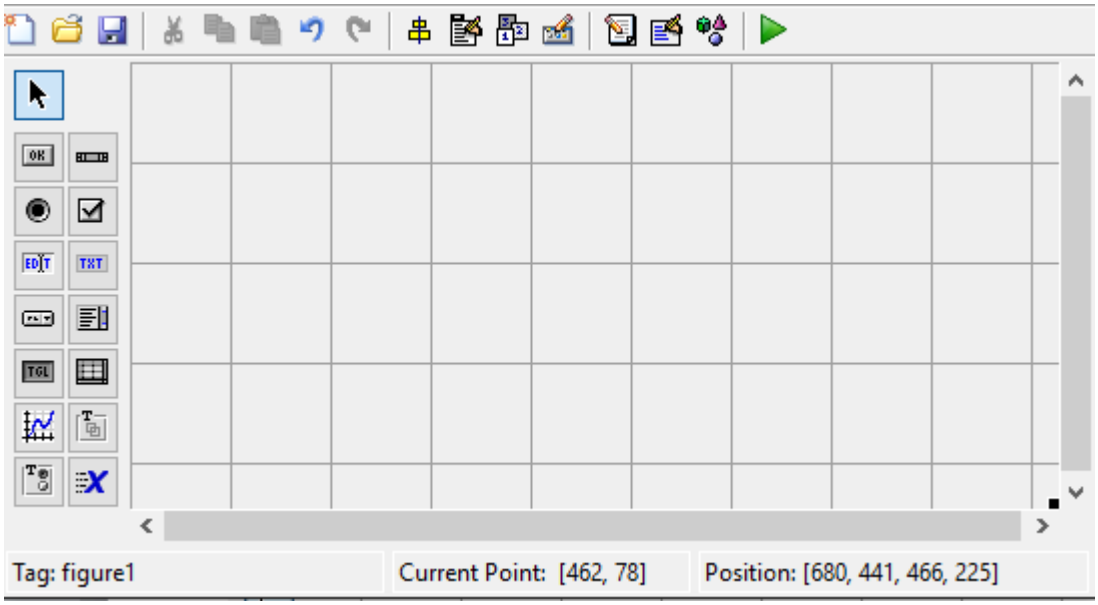


Figura 10.2. Interfață grafică în MATLAB

La salvarea unei interfețe se vor crea două fișiere cu același nume dar cu extensii diferite: un fișier **.fig* (în care se află obiectele grafice și zona de lucru) și un fișier **.m* (ce conține codul MATLAB din care se programează butoanele din interfață).
Observații:

- cele două fișiere trebuie să fie în același folder.
- ambele fișiere trebuie să aibă același nume (diferă doar extensiile). Dacă se dorește redenumirea fișierelor, indicat ar fi să se salveze mai întâi fișierul **.fig* cu noul nume dorit, fișierul **.m* generându-se apoi automat cu noua denumire.
- de fiecare dată când se fac modificări în fișierul **.fig* acesta trebuie salvat, pentru a se actualiza și codul (fișierul **.m*).
- fișierul *M*-file conține următoarele funcții:
    - o funcție principală având aceeași denumire cu cea a fișierului *M*-file.
    - funcția `OpeningFcn` precedată de numele fișierului *M*-file. Această funcție se apelează imediat ce s-a rulat programul, înainte de a accesa orice obiect. Aici se inițializează variabilele.
    - funcția `OutputFcn` precedată de numele fișierului *M*-file.
    - funcții asociate obiectelor introduse în interfață.

Pentru a introduce un obiect grafic în zona de lucru, se dă click pe obiectul grafic dorit și apoi se trasează în zona de lucru o regiune de dimensiunea pe care dorim să o aibă obiectul. Obiectele grafice disponibile sunt cele din *Tabel 10.1*.

Tabel 10.1. Obiectele grafice disponibile într-o interfață MATLAB

Componentă	Iconiță	Descriere
Push Button		*Push Button* este un buton care generează o acțiune atunci când este apăsat. De exemplu, apăsarea unui buton poate deschide o fereastră de dialog.
Slider		*Slider*-ul permite parcurgerea cu pas constant a unui interval de valori. Utilizatorul poate deplasa *Slider*-ul dând *click* și apoi trăgând de *Slider*, sau prin apăsarea săgeților de la capetele *Slider*-ului.
Check Box		*Check Box* permite selectarea uneia sau a mai multor opțiuni.
Radio Button		*Radio Button* permite selectarea unei opțiuni. Este similar cu *Check Box*. Dacă se dorește folosirea mai multor *butoane radio* care să se excludă reciproc (adică la selectarea unui *Radio Button*, butonul selectat anterior să se deselecteze automat) se va folosi *Button Group*.
Edit Text		*Edit Text* permite utilizatorului să introducă și să afișeze date care sunt interpretate ca șiruri de caractere.
Static Text		*Static Text* afișează linii de text.
Pop-Up Menu		*Pop-up Menu* deschide pentru afișare o listă de opțiuni atunci când se dă click pe săgeată și permite utilizatorului să selecteze o singură opțiune.
List Box		*List Box* afișează o listă de opțiuni și permite utilizatorului să selecteze una sau mai multe opțiuni. Implicit se poate selecta o singură opțiune. Pentru a putea selecta mai multe opțiuni trebuie ca *Max* – *Min* > 1.
Axes		*Axes* permite reprezentarea graficelor și imaginilor.
Panel		*Panel* aranjează componentele unui GUI în grupuri, pentru o mai bună organizare a interfeței.
Button Group		Permite gruparea *butoanelor radio* într-un grup astfel încât la selectarea unui *Radio Button*, butonul selectat anterior să se deselecteze automat
Table		Permite inserarea unui tabel în interfață.
ActiveX Control		Permite introducerea controalelor ActiveX înregistrate pe calculator.
Toggle Button		Este un buton care spre deosebire de *Push Button* are două stări: *apăsat*(în jos) și *neapăsat*(în sus).

Toate proprietățile asociate unui obiect pot fi accesate și modificate atât din interfață (fișierul **.fig*) cât și din cod (fișierul **.m*) . Pentru a vedea în interfață proprietățile unui obiect și a le modifica, se dă dublu-click pe obiectul respectiv. De exemplu, pentru un *Slider*, lista de proprietăți arată astfel:

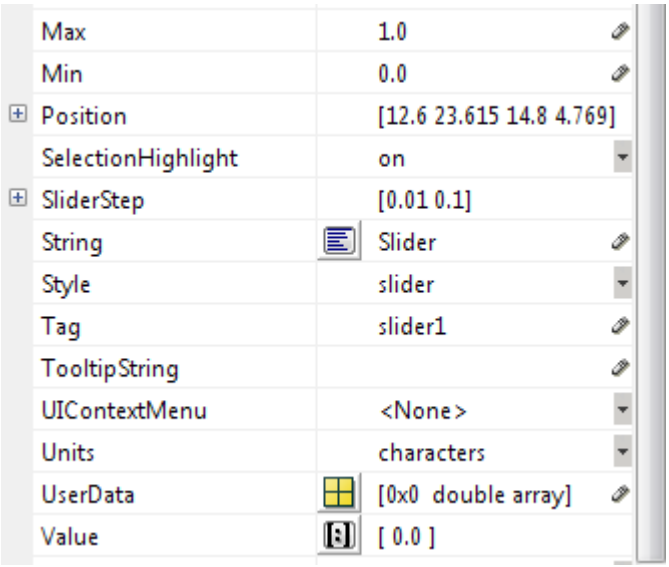


Figura 10.3. Câteva dintre proprietățile asociate unui *Slider*

Pentru a accesa din cod (din fișierul *M*-file) un obiect grafic, se folosește variabila `hObject` sau *Tag*-ul obiectului precedat de `handles.`

- pentru a citi valoarea proprietății unui obiect se folosește funcția `get`.

```
variabila = get(handles.tag_obiect,'proprietate')
```

- pentru a seta valoarea proprietății unui obiect se folosește funcția `set`.

```
set(handles.tag_obiect,'proprietate',valoare)
```

Obiectul grafic *Slider*

Slider-ul permite parcurgerea cu pas constant a unui interval de valori *Min÷Max*. Pentru a introduce un *Slider* în interfață, se dă click pe butonul având *tooltip*-ul *Slider* și apoi se trasează un dreptunghi în zona de lucru, având dimensiunea *Slider*-ului dorit. Dacă se dă dublu click pe *Slider* se va deschide o listă de proprietăți ca cea din *Figura 10.3*. Dintre proprietățile asociate unui *Slider*, ne interesează în mod special:

- *Tag*: reprezintă numele asociat *Slider*-ului (este de preferat ca *Tag*-ul să fie sugestiv. *Exemplu*: dacă se realizează un *Slider* din care se modifică frecvența unui semnal, atunci ar putea fi numit *sliderF*).
- *Min*: valoarea minimă de la care pornește *Slider*-ul.
- *Max*: valoarea maximă până la care merge *Slider*-ul.

- *SliderStep*: pasul cu care se modifică *Slider*-ul. Pentru ca *Slider*-ul să meargă din 1 în 1, parametrul *SliderStep* se calculează ca 1/(*Max* - *Min*).
- *Value*: reprezintă valoarea indicată de *Slider*. *Value* trebuie să fie tot timpul în intervalul [*Min*, *Max*]. *Atenție:* dacă selectați *Min* = 5 și *Max* = 20, atunci parametrul *Value* trebuie să fie setat în intervalul [5, 20]. Implicit *Value* = 0, iar în acest caz MATLAB-ul va semnala eroare.

Toate proprietățile pot fi accesate și modificate dinamic din cod (fișierul *M*-file).

Dacă *Tag*-ul *Slider*-ului este *sliderF*, în momentul în care se salvează fișierul **.fig* se vor crea în cod 2 funcții: `sliderF_CreateFcn` și `sliderF_Callback`. La fiecare mișcare a *Slider*-ului se apelează funcția `sliderF_Callback`. În această funcție putem scrie codul prin care se citește valoarea selectată cu *Sliderul* `sliderF`. Pentru a salva într-o variabilă `F` valoarea selectată cu *sliderF*, se folosește sintaxa: `F = get(handles.sliderF,'Value')`.
Pentru a seta valoarea lui *sliderF* cu o anumită valoare (de exemplu 5), se folosește sintaxa `set(sliderF,'value',5)`.

Obiectul grafic *Static Text*

Obiectul grafic *Static Text* este utilizat pentru afișarea unei valori. Dintre toate proprietățile asociate unui *Static Text*, următoarele ne interesează în mod special:

- *Tag*: reprezintă numele asociat obiectului grafic *Static Text*
- *String*: valoarea scrisă în acest câmp este cea care se va afișa.

Pentru a scrie din cod o anumită valoare (de exemplu 2) într-un câmp *Static Text* având *Tag*-ul *valF*, se folosește sintaxa `set(handles.valF,'String',2)`.

Obiectul grafic *Edit Text*

Obiectul grafic *Edit Text* este utilizat pentru preluarea și afișarea unei valori. Dintre toate proprietățile asociate unui *Edit Text*, ne interesează în mod special:

- *Tag*: reprezintă numele asociat obiectului grafic *Edit Text*
- *String*: în acest câmp se scrie ceea ce se dorește a se afișa și tot din acest câmp se și preia informația scrisă în *Edit Text*. *Observație*: valoarea preluată din interfață este de tip *string*. Dacă se dorește ca valoarea preluată să fie de tip numeric, de exemplu *double*, trebuie să se facă conversia `str2double`.

Pentru a scrie din cod o anumită valoare (de exemplu 2) într-un câmp *Edit Text* având *Tag*-ul *valF*, se folosește sintaxa `set(handles.valF,'String',2)`.
Dacă se dorește să se preia o valoare numerică din interfață se folosește sintaxa:

```
F = str2double(get(handles.valF, 'String'))
```

Obiectul grafic *Axes*

Acest obiect permite afișarea graficelor și imaginilor într-un GUI. Modul de utilizare este foarte simplu. Se selectează obiectul *Axes* și apoi se trasează în zona de lucru o suprafață în care se dorește reprezentarea grafică. Pentru *Axes* este importantă proprietatea *Tag*, care reprezintă numele asociat obiectului.

- Să se realizeze o interfață grafică (GUI) în care să se afișeze o sinusoidă cu următorii parametri:
  - Amplitudinea se modifica dintr-un câmp *Edit Text*
  - Frecvența sinusoidei se modifică dintr-un *Slider* și poate avea valori în intervalul 0÷10Hz, cu pas constant de 1Hz.

Ceilalți parametri ai sinusoidei se vor inițializa în cod (frecvența de eșantionare de 100Hz, durata semnalului de 2 secunde, faza inițială nulă).
Se va implementa un GUI conținând următoarele obiecte:

- *Slider* cu proprietățile: *Tag* = *sliderF*, *Min* = 0, *Max* = 10, *SliderStep* = 0.1
- *Static Text* în care se afișează valoarea frecvenței selectată cu *Slider*-ul, având *Tag* = *valF*. Se vor mai adăuga două componente de tip *Static Text* în care se va scrie "*F=*" și *"Hz"*. *Tag*-urile acestor două obiecte nu sunt importante, deoarece nu vor fi accesate din fișierul *M*-file.
- Un obiect *Edit Text* din care se va modifica amplitudinea. *Tag*-ul va fi *valA*. Se va mai adăuga o componentă de tip *Static Text* în care se va scrie "*A=*".
- O zonă în care se afișează graficul, având *Tag* = *axes1*.

Interfața se va salva cu denumirea *sinusoida.fig*. Automat se va crea și fișierul M-file *sinusoida.m*.

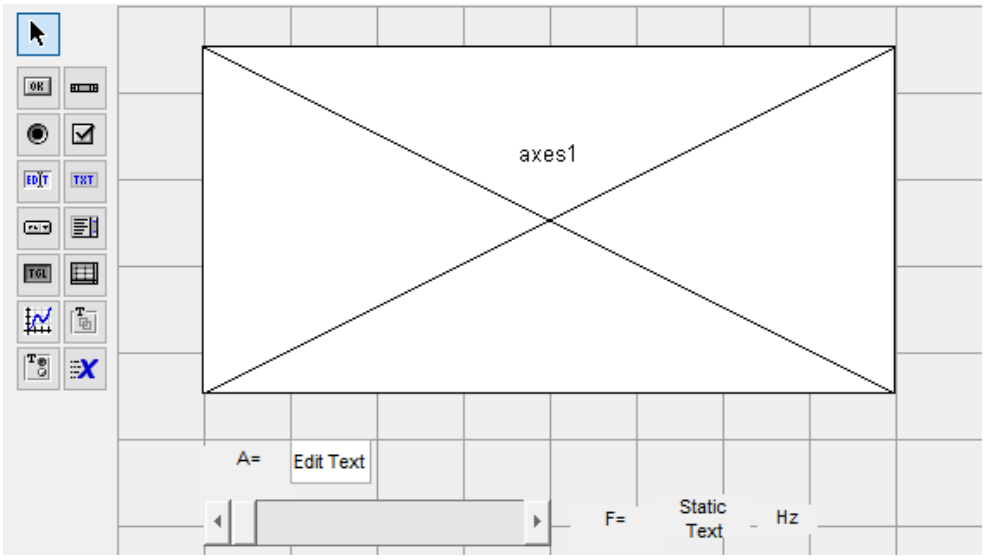


Figura 10.4. Fișierul *sinusoida.fig*

Conținutul fișierului *sinusoida.m* este următorul:

```
function varargout = sinusoida(varargin)
function sinusoida_OpeningFcn(hObject, eventdata, handles, varargin)
function varargout = sinusoida_OutputFcn(hObject, eventdata, handles)
function valA_Callback(hObject, eventdata, handles)
function valA_CreateFcn(hObject, eventdata, handles)
function sliderF_Callback(hObject, eventdata, handles)
function sliderF_CreateFcn(hObject, eventdata, handles)
```

La fiecare mișcare a *Slider*-ului se apelează funcția `sliderF_Callback`. În această funcție se poate scrie codul prin care se citește valoarea selectată cu *Slider*-ul `sliderF` și se afișează în câmpul de *Static Text* având *Tag*-ul `valF`. Deoarece variabila în care se salvează valoarea frecvenței trebuie să fie recunoscută și în alte funcții, aceasta se declară variabilă globală (`global F`) în toate funcțiile în care se folosește. Pentru o bună structurare a codului este indicat să se genereze și să se reprezinte grafic sinusoida într-o funcție separată, de exemplu funcția `grafic`. După citirea valorii frecvenței se apelează funcția `grafic(handles)`. În aceste condiții, funcția `sliderF_Callback` este următoarea:

```
function sliderF_Callback(hObject, eventdata, handles)
global F
% se salveaza in F valoarea selectata cu sliderul
F = get(handles.sliderF,'Value');
% se scrie in campul Static Text valoarea frecventei F
set(handles.valF,'String',F);
grafic(handles) % se apeleaza functia grafic
```

Conținutul funcției `valA_Callback` asociată câmpului *Edit Text* este:

```
function valA_Callback(hObject, eventdata, handles)
global A
% se salveaza in variabila A valoarea scrisa in Edit Text
% valoarea este convertita din string in double
A = str2double(get(handles.valA, 'String'));
grafic(handles)  % se apeleaza functia grafic
```

Funcția `grafic(handles)` în care se generează și se reprezintă grafic sinusoida poate fi scrisă la sfârșitul programului. Conținutul funcției este:

```
function grafic(handles)
global F
global A
Fs = 100; durata = 2;
t = 0:1/Fs:durata;
x = A*sin(2*pi*F*t);
axes(handles.axes1)
plot(t,x)
```

Pentru ca programul să pornească cu valori implicite (de exemplu `F=2Hz` și `A=3`) se modifică funcția `sinusoida_OpeningFcn` astfel:

```
function sinusoida_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
global A
global F
A = 3;
F = 2;
set(handles.valA,'String',A);
set(handles.valF,'String',F);
set(handles.sliderF,'Value',F);
grafic(handles)
```

La rularea programului *sinusoida.m*, rezultatul este cel din *Figura 10.5*.

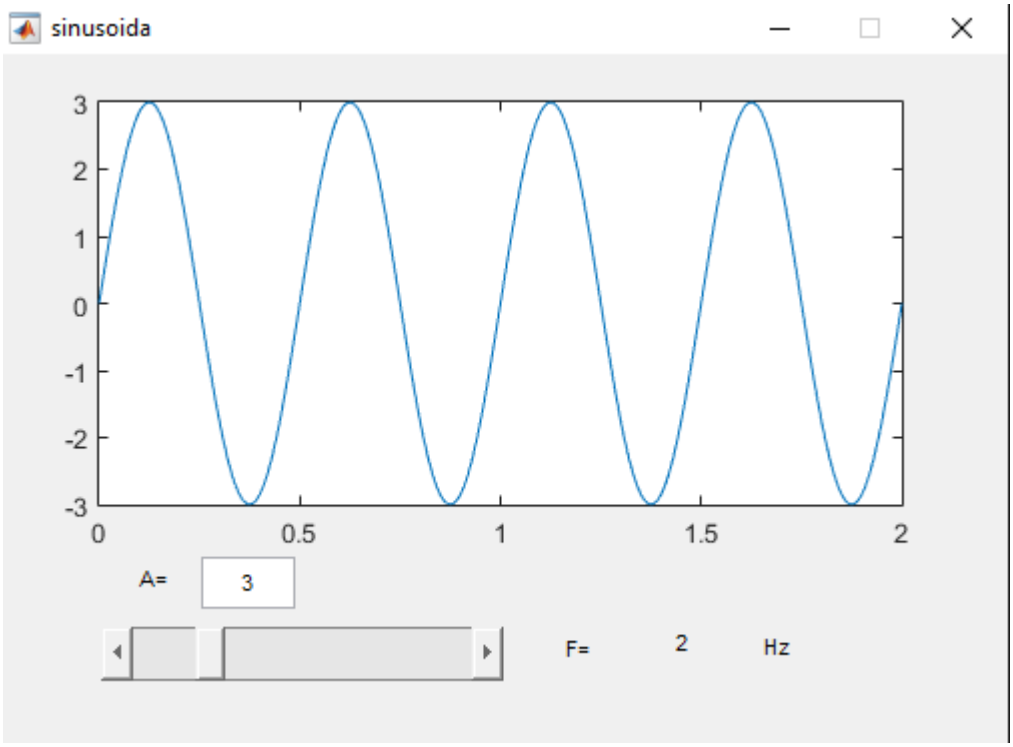


Figura 10.5. Sinusoidă cu parametri ce pot fi modificați din interfață

Observație: implementarea nu este unică, în MATLAB existând multe alte moduri de a realiza o aplicație cu aceeași funcționalitate.

Să se realizeze o interfață grafică numită *filtre* care să conțină un *Button Group* (câte un *Radio Button* pentru fiecare tip de filtru: FTJ, FTS, FTB și FOB). După selectarea filtrului se va afișa caracteristica ideală de amplitudine a filtrului ales. Într-un câmp de *Edit Text* se va introduce frecvența de tăiere *Ft*. Dacă filtrul ales este FTB sau FOB trebuie să se poată introduce într-un *Edit Text* și cea de-a doua frecvență de tăiere *Ft2*; acest *Edit Text* este vizibil doar dacă se alege FOB sau FTB. Implicit aplicația pornește cu filtru FTJ și *Ft* = 1000Hz. Frecvența de eșantionare este *Fs* = 10000Hz.

Fișierul *filtre.fig* va arăta astfel:

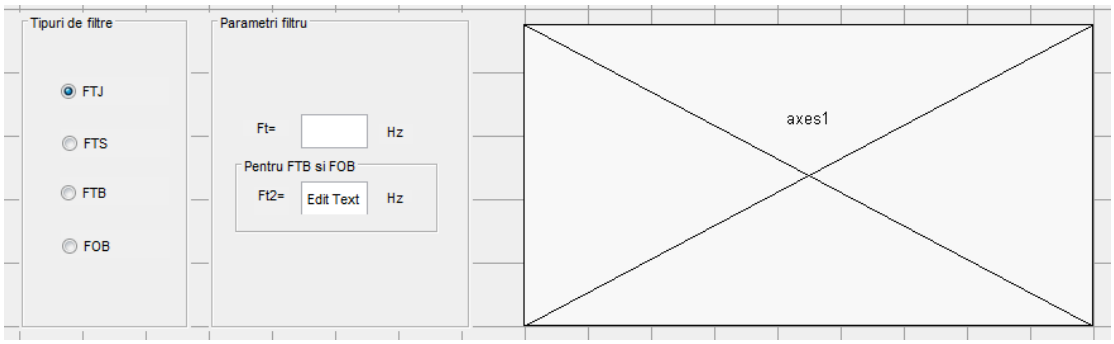


Figura 10.6. Interfața *filtre.fig*

Dacă *Button Group*-ul are *Tag*-ul *uipanel_filtre*, atunci funcția asociată acestui obiect grafic va fi `uipanel_filtre_SelectionChangedFcn`. Dacă această funcție nu se află în fișierul *filtre.m*, atunci dați click dreapta pe grupul de butoane din fișierul *filtre.fig*, selectați *View Callbacks* și apoi *SelectionChangedFcn*. În acest moment, funcția `uipanel_filtre_SelectionChangedFcn` a fost adăugată în fișierul *filtre.m*.

```
function uipanel_filtre_SelectionChangedFcn(hObject, eventdata, handles)
global content
content = get(hObject,'Tag');
switch(content)
    case 'FTJ'
          set(handles.uipanelFt2, 'Visible','Off')
          FTJ(handles)
    case 'FTS'
          set(handles.uipanelFt2, 'Visible','Off')
          FTS(handles)
    case 'FTB'
          set(handles.uipanelFt2, 'Visible','On')
          FTB(handles)
    case 'FOB'
          set(handles.uipanelFt2, 'Visible','On')
          FOB(handles)
end
```

Observații:

- Câmpul de *Edit Text* în care se introduce frecvența de tăiere `Ft2`, face parte din *Panel-ul* cu *Tag*-ul *uipanelFt2*. Acest *Panel* este vizibil doar dacă se selectează filtru FTB sau FOB; pentru FTJ și FTS acest *Panel* este invizibil.
- *Tag*-urile butoanelor radio sunt *FTJ, FTS, FTB* și *FOB*. În funcție de butonul selectat se apelează funcția corespunzătoare.

Dacă *Tag*-ul câmpului de *Edit Text* în care se introduce frecvența de tăiere `Ft` este *Ft*, atunci funcția asociată acestui *Edit Text* va fi `Ft_Callback`.

```
function Ft_Callback(hObject, eventdata, handles)
global Ft
global content
Ft = str2double(get(handles.Ft,'string'));
switch(content)
    case 'FTJ'
          FTJ(handles)
    case 'FTS'
          FTS(handles)
    case 'FTB'
          FTB(handles)
    case 'FOB'
          FOB(handles)
end
```

După ce se modifică frecvența de tăiere `Ft` se apelează din nou funcția asociată tipului de filtru selectat cu *Radio Button* pentru a se actualiza reprezentarea grafică cu noua frecvență de tăiere `Ft`.

Dacă *Tag*-ul câmpului de *Edit Text* în care se introduce frecvența de tăiere `FT2` este *Ft2*, atunci funcția asociată acestui *Edit Text* va fi `Ft2_Callback`.

```
function Ft2_Callback(hObject, eventdata, handles)
global Ft2
global content
Ft2 = str2double(get(handles.Ft2,'string'));
switch(content)
    case 'FTB'
          FTB(handles)
    case 'FOB'
          FOB(handles)
end
```

După ce se modifică frecvența de tăiere `Ft2` se apelează din nou funcția asociată tipului de filtru selectat cu *Radio Button* pentru a se actualiza reprezentarea grafică cu noua frecvență de tăiere `Ft2`.

Pentru ca aplicația să pornească cu parametrii impliciți specificați în enunț, funcția `filtre_OpeningFcn`, va fi modificată astfel:

```
function filtre_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
global Ft
global Ft2
global Fs
global content
content = 'FTJ';
Fs = 10000;
Ft = 1000; Ft2 = 2000;
set(handles.Ft,'string',Ft);
set(handles.Ft2,'string',Ft2);
set(handles.uipanelFt2, 'Visible','Off')
FTJ(handles)
```

Funcțiile ce se apelează la selectarea butoanelor radio sunt:

- Funcția `FTJ(handles)` atunci când se apasă butonul radio cu *Tag*-ul *FTJ*

```
function FTJ(handles)
global Ft
global Fs
% H_ideal = caracteristica ideala de amplitudine
H_ideal(-Fs/2+Fs/2+1:-Ft+Fs/2+1) = 0;
H_ideal(-Ft+Fs/2+1:Ft+Fs/2+1) = 1;
H_ideal(Ft+Fs/2+1:Fs/2+Fs/2+1) = 0;
axes(handles.axes1)
cla
hold on
plot(linspace(-Fs/2, Fs/2, length(H_ideal)), H_ideal,'r')
plot([0 0], [0 1.2], 'g')
title('Caracteristica ideala de amplitudine pentru FTJ')
xlabel('Frecventa [Hz]')
hold off
```

- Funcția `FTS(handles)` atunci când se apasă butonul radio cu *Tag*-ul *FTS*

```
function FTS(handles)
global Ft
global Fs
% H_ideal = caracteristica ideala de amplitudine
H_ideal(-Fs/2+Fs/2+1:-Ft+Fs/2+1) = 1;
H_ideal(-Ft+Fs/2+1:Ft+Fs/2+1) = 0;
H_ideal(Ft+Fs/2+1:Fs/2+Fs/2+1) = 1;
axes(handles.axes1)
cla
hold on
plot(linspace(-Fs/2, Fs/2, length(H_ideal)), H_ideal,'r')
plot([0 0], [0 1.2], 'g')
title('Caracteristica ideala de amplitudine pentru FTS')
xlabel('Frecventa [Hz]')
hold off
```

- Funcția `FTB(handles)` atunci când se apasă butonul radio cu *Tag*-ul *FTB*

```
function FTB(handles)
global Fs
global Ft
global Ft2
% Ft trebuie sa fie mai mica decat Ft2
if(Ft2 < Ft)
    temp = Ft2;
    Ft2 = Ft;
    Ft = temp;
end
H_ideal(-Fs/2+Fs/2+1:-Ft+Fs/2+1) = 0;
H_ideal(-Ft2+Fs/2+1:-Ft+Fs/2+1) = 1;
H_ideal(-Ft+Fs/2+1:Ft+Fs/2+1) = 0;
H_ideal(Ft+Fs/2+1:Ft2+Fs/2+1) = 1;
H_ideal(Ft2+Fs/2+1:Fs/2+Fs/2+1) = 0;
axes(handles.axes1)
cla
hold on
plot(linspace(-Fs/2, Fs/2, length(H_ideal)), H_ideal,'r')
plot([0 0], [0 1.2], 'g')
title('Caracteristica ideala de amplitudine pentru FTB')
xlabel('Frecventa [Hz]')
hold off
```

- Funcția `FOB(handles)` atunci când se apasă butonul radio cu *Tag*-ul *FOB*

```
function FOB(handles)
global Fs
global Ft
global Ft2
% Ft trebuie sa fie mai mica decat Ft2
if(Ft2 < Ft)
    temp = Ft2;
    Ft2 = Ft;
    Ft = temp;
end
H_ideal(-Fs/2+Fs/2+1:-Ft+Fs/2+1) = 1;
H_ideal(-Ft2+Fs/2+1:-Ft+Fs/2+1) = 0;
H_ideal(-Ft+Fs/2+1:Ft+Fs/2+1) = 1;
H_ideal(Ft+Fs/2+1:Ft2+Fs/2+1) = 0;
H_ideal(Ft2+Fs/2+1:Fs/2+Fs/2+1) = 1;
axes(handles.axes1)
cla
hold on
plot(linspace(-Fs/2, Fs/2, length(H_ideal)), H_ideal,'r')
plot([0 0], [0 1.2], 'g')
title('Caracteristica ideala de amplitudine pentru FOB')
xlabel('Frecventa [Hz]')
hold off
```

În final, în funcție de filtrul ales (FTJ, FTS, FTB, FOB), aplicația va arăta astfel:

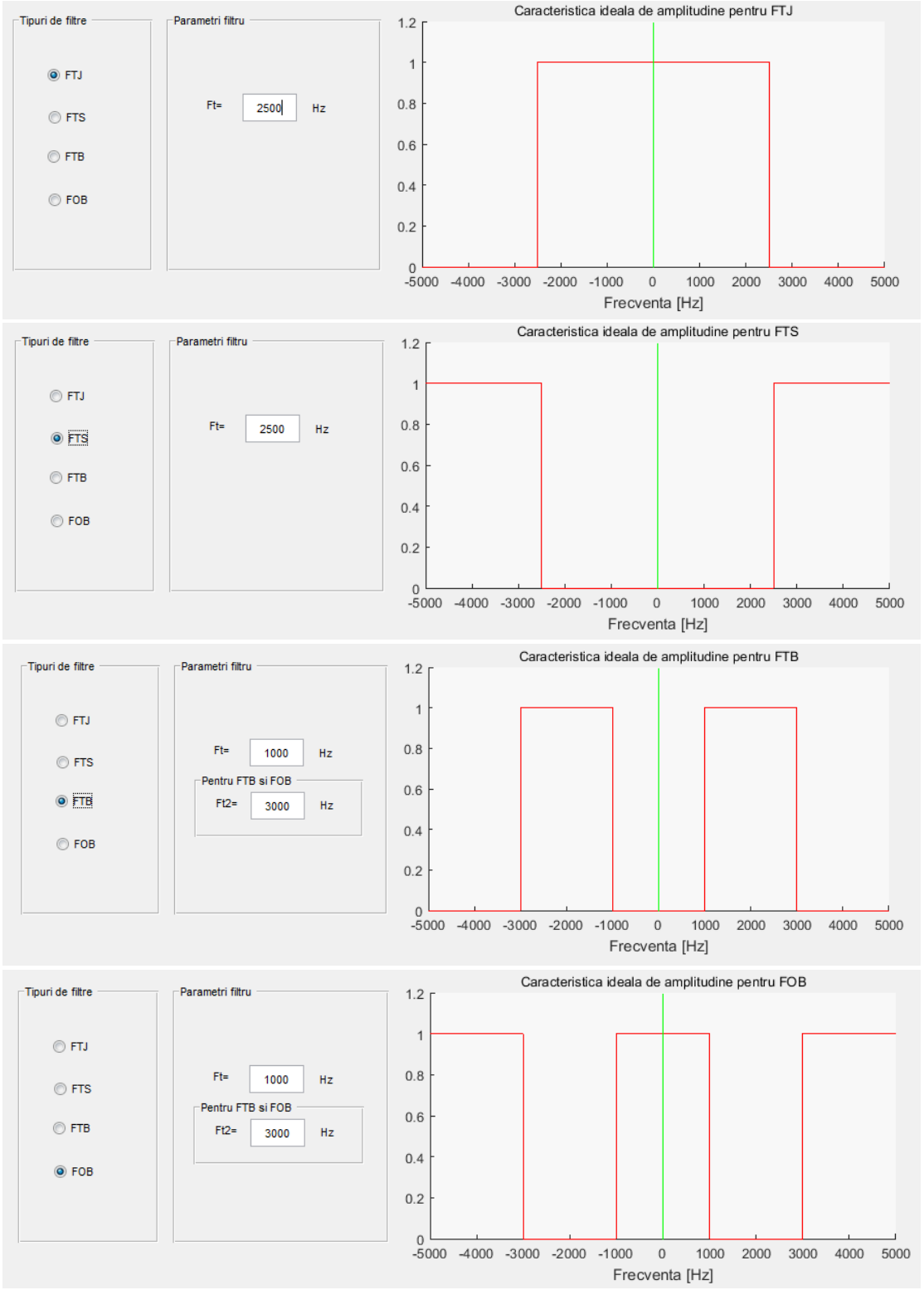


Figura 10.7. Interfața în funcție de tipul de filtru ales

Să se realizeze o interfață care să permită încărcarea unui semnal audio prin apăsarea unui buton numit *incarca semnal audio*. După ce semnalul este încărcat: se va afișa calea către semnal; se va afișa frecvența cu care semnalul a fost eșantionat; se va reprezenta spectrul semnalului; va deveni vizibil un buton numit *play* care atunci când este apăsat va reda semnalul audio. Va exista și un buton numit *clear* care atunci când este apăsat va aduce toate obiectele grafice la parametrii inițiali.

- Butonul din care se încarcă semnalul audio va avea *Tag*-ul *incarca_audio*.
- Calea către semnal se va afișa într-un câmp *Static Text* cu *Tag*-ul = *cale*.
- Frecvența de eșantionare se va afișa într-un câmp *Static Text* cu *Tag*-ul = *Fs*.

Interfața va arăta astfel:

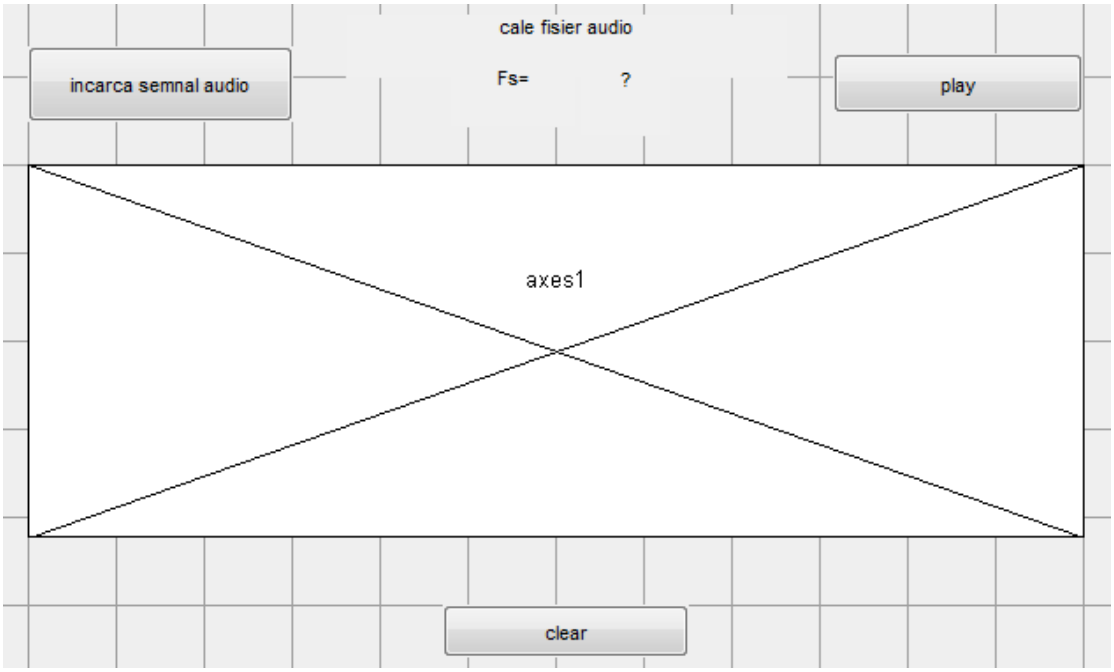


Figura 10.8. Interfață ce permite încărcarea unui semnal audio

Funcția `pushbutton_incarca_audio` asociată butonului de încărcare a semnalului audio va avea următorul conținut.

```
function pushbutton_incarca_audio_Callback(hObject, eventdata, handles)
global semnal
global Fs
% se selecteza fisierul audio
[nume, cale] = uigetfile('*wav','Alege un semnal wav');
cale_nume  = strcat(cale,nume);
% se afiseaza calea catre semnalul audio
set(handles.cale,'string',cale_nume);
% se citeste semnalul audio
[semnal,Fs] = audioread(cale nume);
% se afiseaza frecventa de esantionare
```

```
set(handles.Fs,'string',Fs);
% se afiseaza spectrul semnalului audio
axa_fft = linspace(-Fs/2, Fs/2, length(semnal));
axes(handles.axes1)
plot(axa_fft, fftshift(abs(fft(semnal))));
% se face vizibil butonul de play
set(handles.play,'visible','on')
```

Funcția `clear_Callback` asociată butonului care aduce toate obiectele grafice la parametrii inițiali are următorul conținut:

```
function clear_Callback(hObject, eventdata, handles)
axes(handles.axes1)
cla
clear playsnd % opreste redarea smnalului
set(handles.cale,'string','cale fisier audio')
set(handles.Fs,'string','?')
% se face invizibil butonul de play
set(handles.play,'visible','off')
```

Funcția `play_Callback` asociată butonului de redare a semnalului audio are următorul conținut:

```
function play_Callback(hObject, eventdata, handles)
global semnal
global Fs
sound(semnal,Fs)
```

În urma încărcării unui semnal audio, interfața arată astfel:

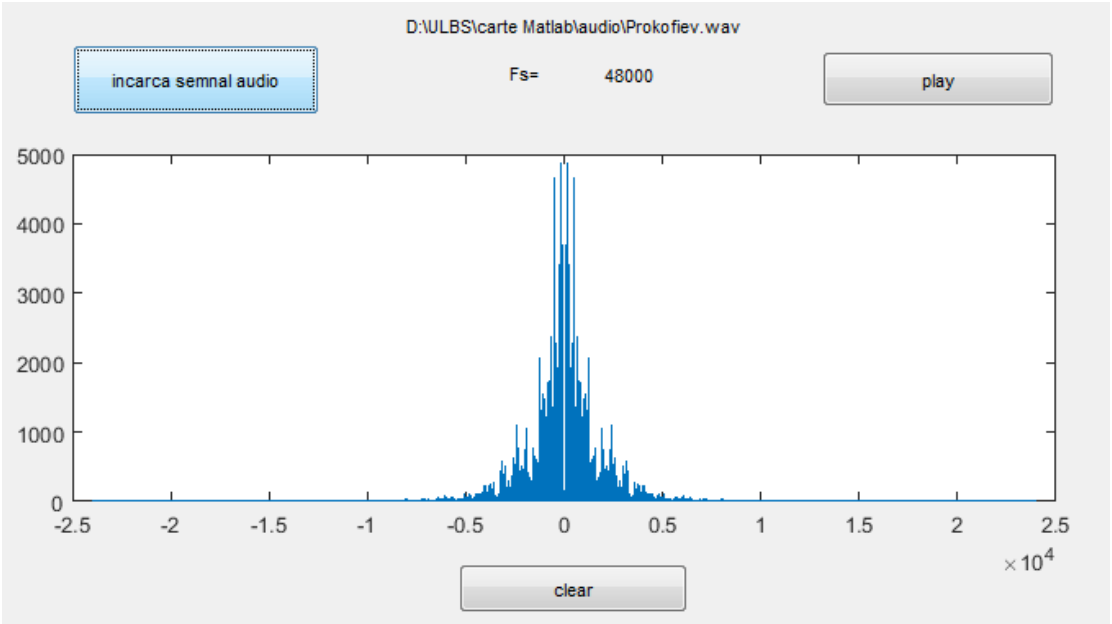


Figura 10.9. Reprezentarea în GUI a spectrului semnalului audio încărcat

Să se realizeze o interfață grafică care să permită încărcarea unei imagini. Să se afișeze imaginea iar după afișare să devină vizibil un buton prin apăsarea căruia să se poată selecta o zonă de interes din imagine (ROI). Să se afișeze zona de interes selectată. Fișierul se va numi *incarca_imagine*.

Funcția `pushbutton_incarca_imag_Callback` asociată butonului de încărcare a imaginii va avea următorul conținut.

```
function pushbutton_incarca_imag_Callback(hObject, eventdata, handles)
global imagine
[nume, cale] = uigetfile('*jpg','Alege o imagine');
cale_nume  = strcat(cale,nume);
imagine = imread(cale_nume);
axes(handles.axes1)
imshow(imagine)
set(handles.pushbutton_ROI,'visible','on')
```

Funcția `pushbutton_ROI_Callback` asociată butonuluinde selectare a regiunii de interes va avea următorul conținut.

```
function pushbutton_ROI_Callback(hObject, eventdata, handles)
global imagine
BW = uint8(roipoly(imagine));
BW = repmat(BW,1,1,3);
imag_ROI = imagine.*BW;
axes(handles.axes2), imshow(imag_ROI)
```

Observație: inițial, butonul pentru selecția regiunii de interes trebuie să fie invizibil. Pentru a realiza acest lucru sunt 2 posibilități:

- În funcția `incarca_imagine_OpeningFcn` se va scrie:
  `set(handles.pushbutton_ROI,'visible','off')`
- În fișierul *incarca_imagine.fig* se dă dublu-click pe butonul respectiv, iar la proprietatea *Visible* se deselectează *On*.

În urma încărcării unei imagini, interfața arată astfel:

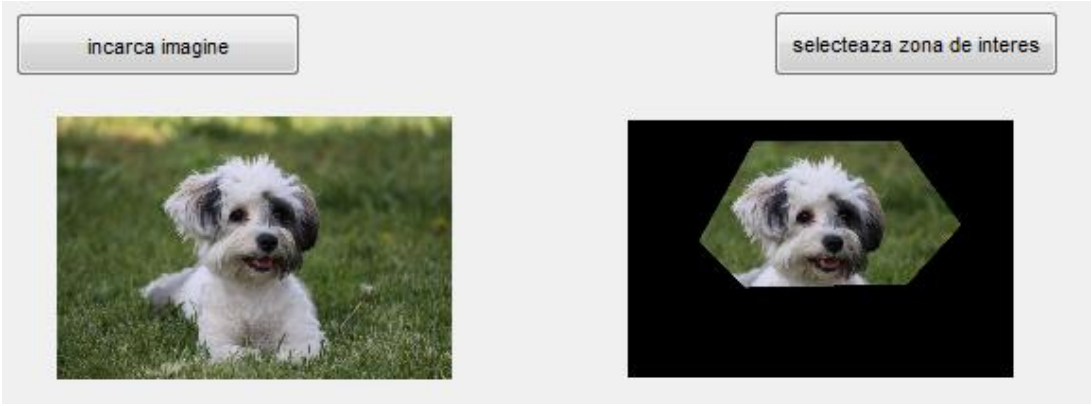


Figura 10.10. Interfață pentru încărcarea unei imagini și selecția ROI

11. Funcții MATLAB utile în Prelucrarea Semnalelor

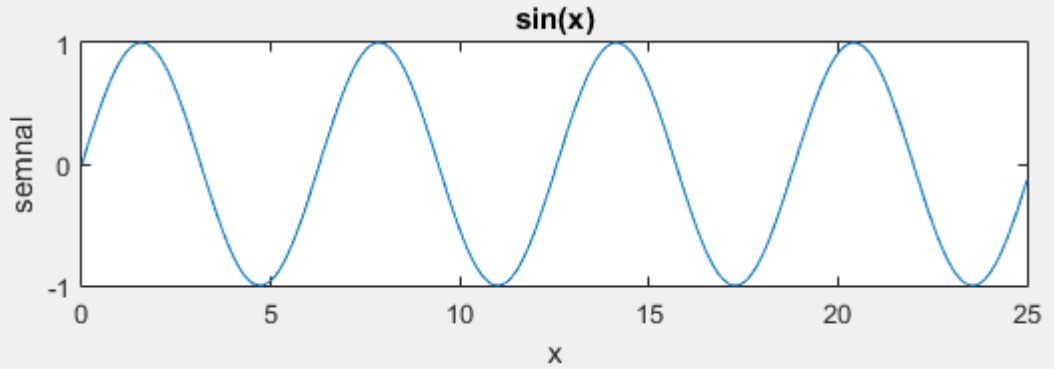
Prelucrarea numerică a semnalelor, fie ele semnale audio, imagini sau semnale generice (de exemplu semnale electrice, etc.), este facilitată de existența mai multor funcții ale MATLAB-ului. In acest capitol vor fi prezentate câteva funcții utile pentru generarea și prelucrarea semnalelor. Deși exemplele vor conține semnale unidimensionale, multe dintre aceste funcții pot fi generalizate și aplicate semnalelor multidimensionale.

11.1 Generarea de semnale

Limbajul MATLAB pune la dispoziție funcții de generare a semnalelor de bază. Funcția `sin(pi/2)` generează valoarea sinusului pentru argumentul `pi/2`. Însă dacă argumentul este un vector de valori, funcția `sin` va genera un semnal numeric.

- Să se calculeze și să se reprezinte grafic $f(x) = sin(x)$ pentru $x \in [0, 25]$.

```
x = 0:0.01:25;
semnal = sin(x);
figure(1)
plot(x, semnal), xlabel('x'), ylabel('semnal')
title('sin(x)')
```



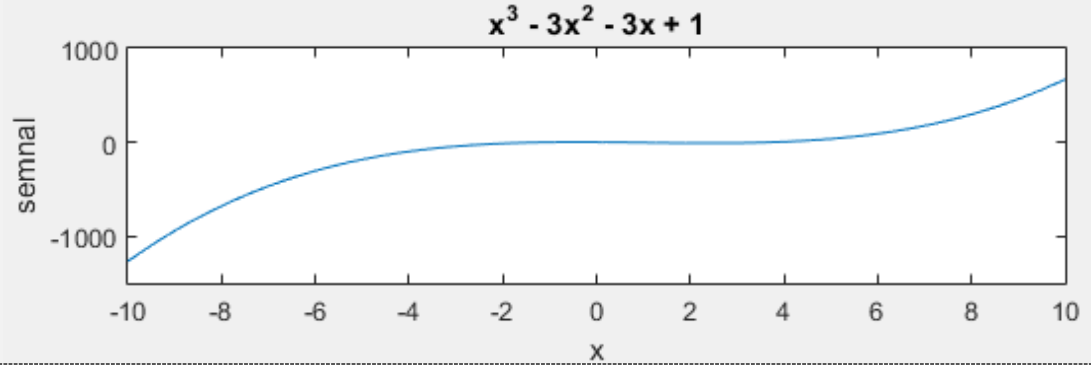
In mod similar, următoarele funcții pot fi folosite pentru a genera semnale:

- *funcții uzuale*: exp, log, abs, sign, square, rectpuls, sawtooth, tripuls
- *funcții trigonometrice*: sin, cos, tan, cot, asin, acos, atan, acot, sinc
- *semnale aleatoare*: random, unifrnd, normrnd, randn
- *alte funcții*: în această categorie pot intra multe funcții, depinzând de aplicație.

MATLAB-ul are predefinite funcții precum `chirp` (o sinusoidă cu frecvența variabilă în timp), `tansig` (sigmoidală bipolară, folosită în rețelele neurale artificiale, de exemplu rețeaua *Perceptron Multistrat*), `gamma` (generalizarea factorialului) etc. O familie de funcții uzuale care nu apare în enumerarea de mai sus este familia *funcțiilor polinomiale*. Pentru generarea valorilor unui polinom (de exemplu pentru $f(x) = x^3 - 3x^2 - 3x + 1$) nu este necesară existența unei funcții speciale în MATLAB, ci se pot folosi ușor operațiile matematice de bază.

Să se calculeze și să se reprezinte grafic $f(x) = x^3 - 3x^2 - 3x + 1$ pentru $x \in [-10, 10]$.

```
x = -10:0.01:10;
semnal  = x.^3 - 3*x.^2 - 3*x + 1;
figure(1), plot(x, semnal),xlabel('x'), ylabel('semnal')
title('x^3 - 3x^2 - 3x + 1')
```

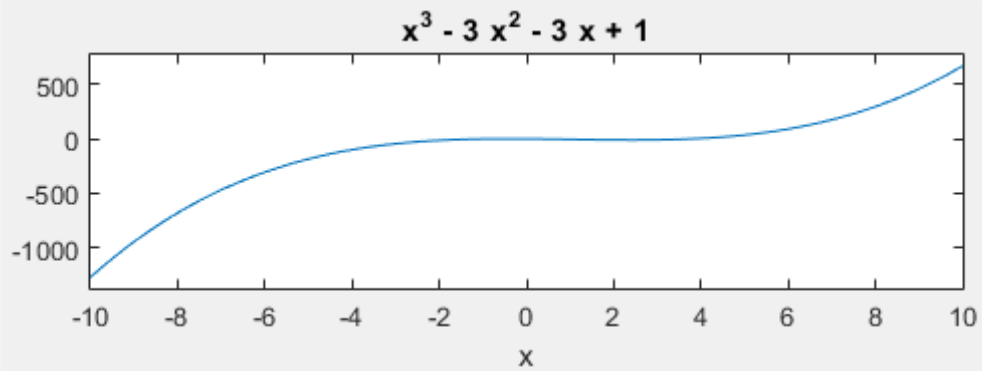


Pentru o reprezentare grafică mult mai rapidă a funcțiilor se poate folosi funcția `ezplot`.

Sintaxă: `ezplot(f, [a, b])` reprezintă grafic funcția `f(x)`, pentru $a < x < b$.

Să se afișeze graficul funcției $f(x) = x^3 - 3x^2 - 3x + 1$, $x \in [-10, 10]$.

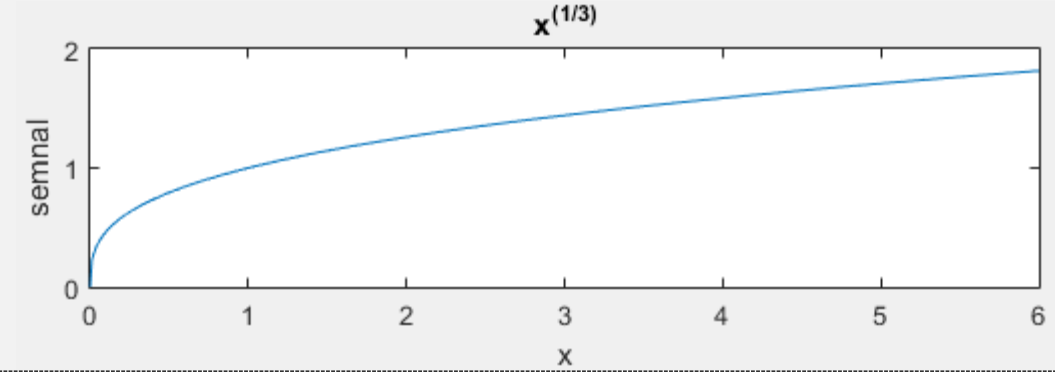
```
ezplot('x^3 - 3*x^2 - 3*x + 1',[-10, 10])
```



Alte funcții folositoare sunt rădăcinile de ordin N (de exemplu $f(x) = \sqrt[3]{x}$). Pentru generarea acestor funcții, se pot folosi din nou operațiile matematice de bază, cu mențiunea că operația de extragere a rădăcinilor de ordin N poate genera numere complexe.

- Să se calculeze și să se reprezinte grafic $f(x) = \sqrt[3]{x}$, $x \in [0, 6]$.

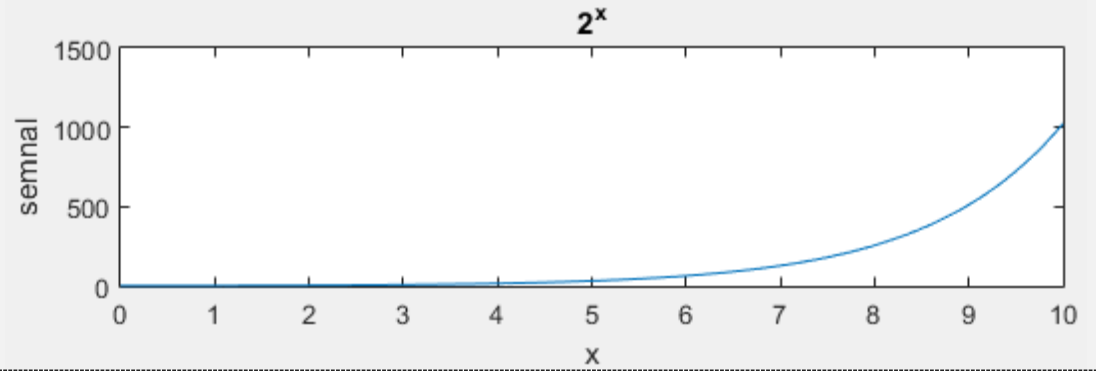
```
x = 0:0.01:6;
semnal  = x.^(1/3);
figure(1)
plot(x, semnal),xlabel('x'), ylabel('semnal')
title('x^{(1/3)}')
```



Funcțiile exponențiale pot fi obținute tot prin operații matematice de bază.

- Să se calculeze și să se reprezinte grafic $f(x) = 2^x$, $x \in [0, 10]$.

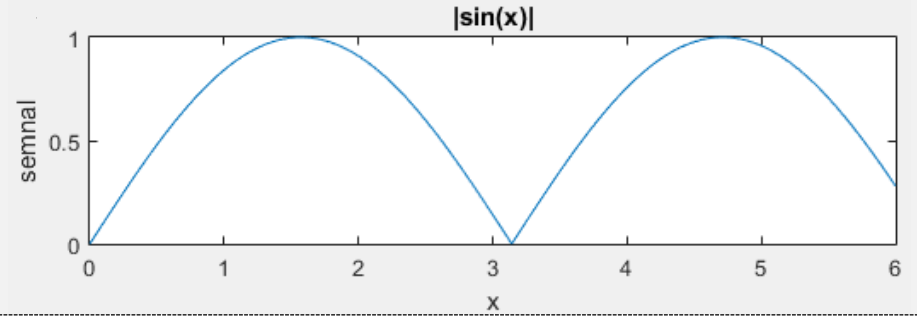
```
x = 0:0.01:10;
semnal  = 2.^x;
figure(1)
plot(x, semnal),xlabel('x'), ylabel('semnal')
title('2^x')
```



De altfel, orice combinație de funcții dintre cele menționate mai sus poate fi obținută în MATLAB, după cum o cer necesitățile aplicației. Mai jos sunt câteva exemple de funcții obținute prin combinații ale funcțiilor de mai sus.

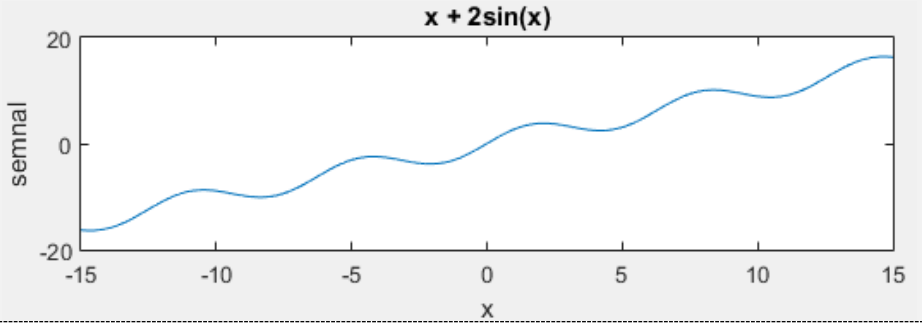
- Să se calculeze și să se reprezinte grafic $f(x) = |sin(x)|$, $x \in [0, 6]$.

```
x = 0:0.01:6;
semnal = abs(sin(x))
figure(1), plot(x, semnal), xlabel('x'), ylabel('semnal')
title('|sin(x)|')
```



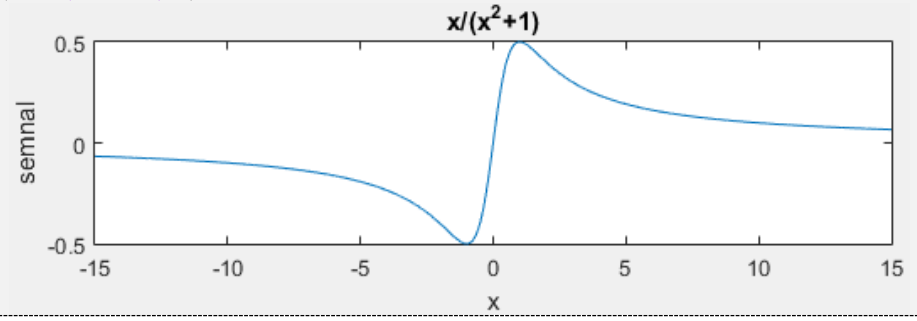
- Să se calculeze și să se reprezinte grafic $f(x) = x + 2 \cdot sin(x)$, $x \in [-15, 15]$.

```
x = -15:0.01:15;
semnal = x + 2*sin(x);
figure(1)
plot(x, semnal), xlabel('x'), ylabel('semnal')
title('x + 2sin(x)')
```



- Să se calculeze și să se reprezinte grafic $f(x) = \frac{x}{x^2+1}$, $x \in [-15, 15]$.

```
x = -15:0.01:15;
semnal = x./(x.^2+1);
figure(1), plot(x, semnal), xlabel('x'), ylabel('semnal')
title('x/(x^2+1)')
```



11.2 Transformarea și vizualizarea în domeniul frecvență

Analiza în domeniul frecvență este o unealtă esențială în domeniul prelucrării semnalelor. Dacă vizualizarea în domeniul timp oferă informații despre cum evoluează semnalele, analiza în domeniul frecvență oferă o altă perspectivă asupra semnalelor, indicând amplitudinile și fazele sinusoidelor în care se descompune semnalul.

Un semnal poate fi transformat din domeniul timp în domeniul frecvență cu ajutorul *Transformatei Fourier*. Pentru lucrul cu semnale discrete, MATLAB-ul pune la dispoziție funcțiile `fft` și `ifft`, implementări ale *Transformatei Fourier Discrete* directe și inverse, care transformă *N* eșantioane ale unui semnal din domeniul timp în *N* valori complexe din domeniul frecvență (și invers).

Datorită modului în care este definită TFD, spectrul semnalului conține *N* valori distincte, pe frecvențele $0 \cdot \frac{F_S}{N}, 1 \cdot \frac{F_S}{N}, 2 \cdot \frac{F_S}{N}, \dots, (N-1) \cdot \frac{F_S}{N}$, însă poate fi extins prin repetiție atât pentru frecvențe pozitive cât și pentru frecvențe negative. Cum în prelucrarea de semnal se preferă interpretarea valorilor în intervalul [-Fs/2... Fs/2), este necesară deplasarea valorilor obținute pe pozițiile [Fs/2 ... Fs) către intervalul [-Fs/2...0). Special pentru această operație, MATLAB-ul are funcția `fftshift` iar rezultatul este cel reprezentat grafic mai jos.

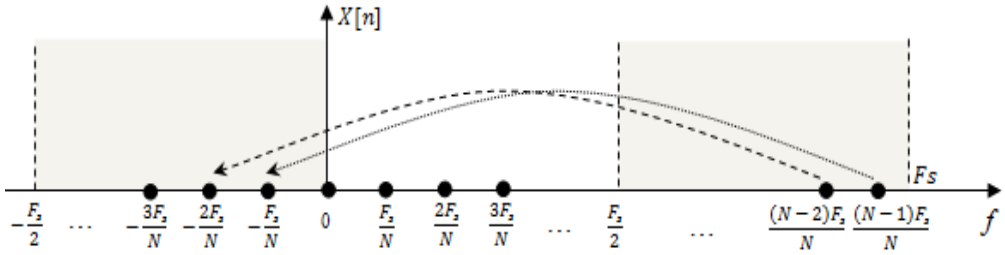


Figura 11.1. Domeniul frecvențelor în intervalul $-\frac{Fs}{2} \div \frac{Fs}{2}$

Generarea spectrului de amplitudine al unui semnal discret x

- Se calculează *Transformata Fourier Discretă* a semnalului *x*[*n*], utilizând funcția `fft` (FFT este acronim de la ***F**ast **F**ourier **T**ransform)*

```
>> x = [1, 0, -2, 1, 1];
>> fft(x)
ans =
1.0000 + 0.0000i    2.1180 + 2.7144i   -0.1180 - 2.2654i   -0.1180 +
2.2654i   2.1180 - 2.7144i
```

- Se calculează spectrul de amplitudine, folosind funcția `abs`

```
>> x = [1, 0, -2, 1, 1];
>> abs(fft(X))
ans =
     1.0000    3.4430    2.2685    2.2685    3.4430
```

- Se realizează shift-area semnalului obținut folosind funcția `fftshif`

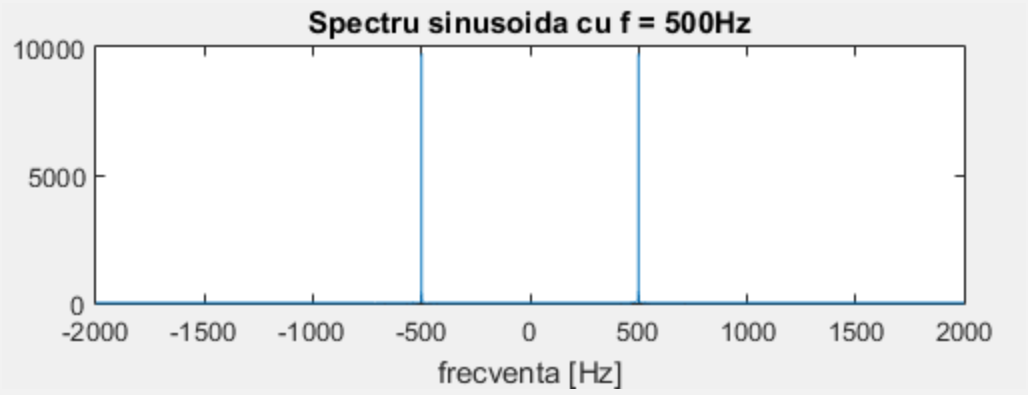
```
>> x = [1 0 -2 1 1];
>> fftshift(abs(fft(x)))
ans =
     2.2685    3.4430    1.0000    3.4430    2.2685
```

- pentru a genera cele *N* valori echidistante ale frecvențelor între `-Fs/2` și `Fs/2`, se folosește funcția `linspace`

```
>> x = [1 0 -2 1 1];
>> Fs = 100;
>> axa_fft = linspace(-Fs/2, Fs/2, length(x))
axa_fft =
   -50   -25     0    25    50
```

Fie semnalul sinusoidal $x(t) = \sin(2\pi ft)$, $f = 500Hz$, de durată 5 secunde, eșantionat cu $Fs = 4kHz$. Să se reprezinte spectrul de amplitudine al lui $x(t)$.

```
Fs = 4000;
f = 500;
t = 0:1/Fs:5;
x = sin(2*pi*f*t);
axa_fft = linspace(-Fs/2,Fs/2,length(x));
figure(1)
plot(axa fft, fftshift(abs(fft(x))))
title('Spectru sinusoida cu f = 500Hz')
```



Alături de TFD unidimensionale (directă `fft` și inversă `ifft`), MATLAB-ul conține implementări și pentru transformatele multidimensionale:

- `fft2` și `ifft2`, folosite de obicei în prelucrarea imaginilor grayscale
- `fftn` și `ifftn`, folosite pentru semnale cu mai mult de 2 dimensiuni

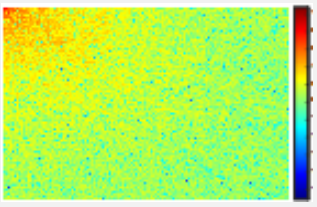
Transformata Fourier Discretă este definită pentru semnale discrete complexe, pentru care fiecare eșantion poate fi o valoare complexă. Insă în foarte multe cazuri, semnalele pe care se lucrează au eșantioane reale și pot fi interpretate ca parte a unor semnale pare fără a pierde semnificația originală. In aceste condiții, pentru a simplifica volumul de calcul și pentru a evita calculul cu valori complexe, se poate folosi *Transformata Cosinus Discretă*.

Ca și *Transformata Fourier Discretă*, *Transformata Cosinus Discreta* (DCT) este inclusă în MATLAB în varianta unidimensională (`dct`, `idct`) și bidimensională (`dct2`, `idct2`), folosită în special pentru prelucrări de imagini.

- În continuare este un exemplu de spectru bidimensional obținut cu ajutorul `dct2` și rezultatul aplicării unei filtrări trece-sus (prin îndepărtarea frecvențelor joase).

```
%citirea imaginii si transformarea ei intr-o imagine grayscale
imag = imread('papagal.jpg');
imag_gray = rgb2gray(imag);
figure(1), imshow(imag_gray);
%calculul spectrului bidimensional
imag_DCT2 = dct2(imag_gray);
figure(2), imshow(log(abs(imag_DCT2)),[]), colormap(jet), colorbar
%anularea frecventelor joase
[lin,col] = size(imag_DCT2);
imag_DCT2HF = imag_DCT2;
imag_DCT2HF(1:lin/8,1:col/8) = 0;
imag_IDCT2 = idct2(imag_DCT2HF);
%afisarea rezultatului
figure(3), imshow(imag_IDCT2/255);
```

Imaginea grayscale



Spectrul bidimensional

Imaginea prelucrată

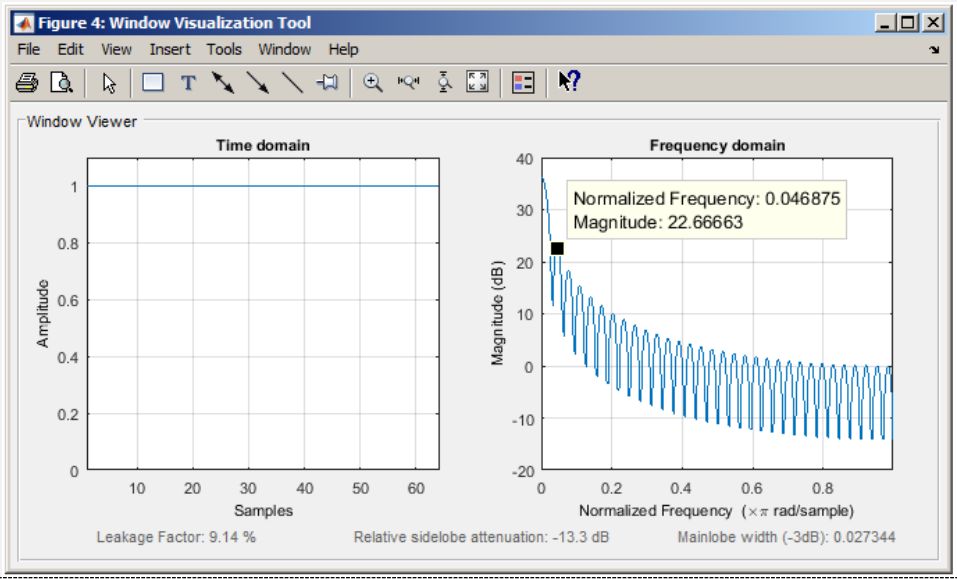
11.3 Ferestruirea

In prelucrarea semnalelor, un semnal este adesea împărțit în segmente în vederea prelucrării ulterioare. De asemenea, ponderea eșantioanelor poate să difere, eșantioanele din centru fiind de obicei cele mai semnificative pentru acea porțiune de semnal, pe când eșantioanele de margine au o pondere mai mică. Aceste ponderi, privite separat și independent de semnal, formează fereastra, iar operația de împărțire a semnalelor în segmente și ponderarea cu o anumită fereastră se numește *ferestruire*.

Cel mai ușor de imaginat este fereastra dreptunghiulară, generată cu funcția `rectwin`, care păstrează toate eșantioanele din fiecare interval cu ponderea 1. Vom folosi această fereastră pentru a ilustra interfața oferită de MATLAB pentru vizualizarea ferestrelor: `wvtool` (*Window Visualization Tool*).

Deschiderea interfeței *Window Visualization Tool* și vizualizarea unei ferestre dreptunghiulare formată din `L` = 64 elemente cu valoarea 1.

```
>> L = 64;
>> wvtool(rectwin(L))
```



In domeniul timp, fereastra este un vector de 64 de elemente cu valoarea 1, care poate fi la fel de bine generat cu funcția generică `ones`. Insă `wvtool` ne oferă și magnitudinea răspunsului în frecvență al ferestrei, care corespunde cu modulul *Transformatei Fourier* al acesteia, reprezentat în decibeli. Deoarece nu există informații asupra frecvenței de eșantionare pentru fereastră, axa frecvențelor reprezintă frecvențele normalizate, între 0 și π. In această interfață se pot investiga valorile în domeniul timp și magnitudinile în frecvență prin click-uri pe imagine.

Pentru generarea ferestrelor, există funcția MATLAB `window`, care acceptă ca argument denumirea ferestrei și parametrii acesteia (incluzând lungimea ferestrei). Lista de denumiri acceptate de funcția `window` este prezentată in *Tabelul 11.1* de mai jos, însă fiecare dintre aceste denumiri poate fi folosită ca funcție separată pentru generarea ferestrelor.

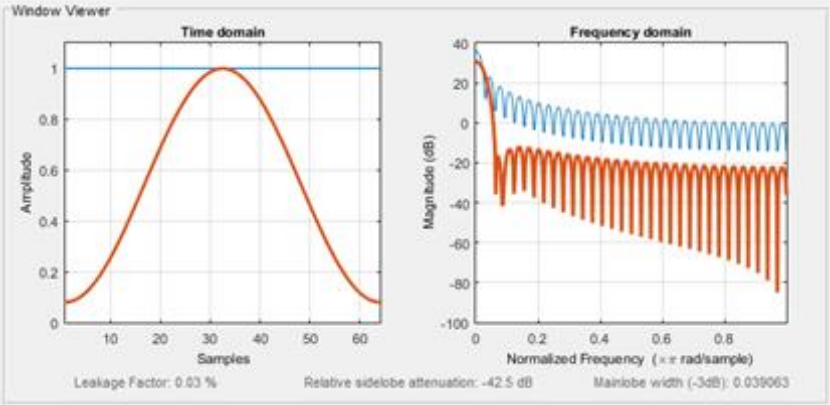
Tabel 11.1. Lista de ferestre acceptate de funcția `window`

Denumirea ferestrei	Denumirea completă a ferestrei
@bartlett	Bartlett
@barthannwin	Bartlett-Hanning
@blackman	Blackman
@blackmanharris	Blackman-Harris
@bohmanwin	Bohman
@chebwin	Chebyshev
@flattopwin	Flat Top
@gausswin	Gaussian
@hamming	Hamming
@hann	Hann
@kaiser	Kaiser
@nuttallwin	Nuttall
@parzenwin	Parzen
@rectwin	Dreptunghiulară
@taylorwin	Taylor
@tukeywin	Tukey
@triang	Triunghiulară

Pentru compararea performanțelor diverselor ferestre, funcția `wvtool` acceptă mai mult de un argument, reprezentând analiza mai multor ferestre în același grafic.

Să se genereze ferestrele *rectangulară* și *Hamming* și să se compare caracteristica de frecvență în `wvtool`

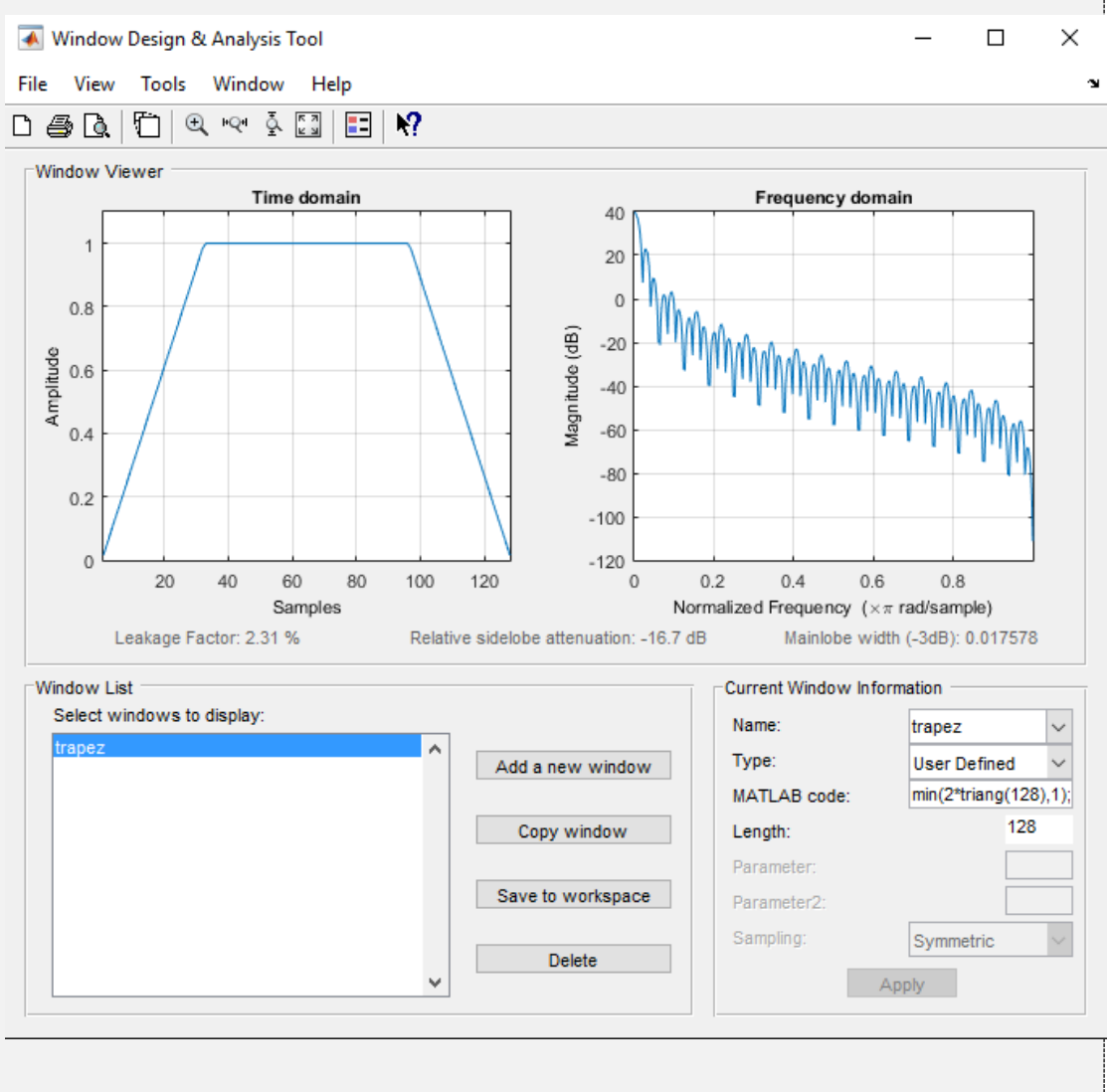
```
w1 = rectwin(64); % alternativ: w1 = window(@rectwin,64);
w2 = hamming(64); % alternativ: w2 = window(@hamming,64);
wvtool(w1,w2);
```



Dacă niciunul dintre tipurile de ferestre de mai sus nu satisface necesitățile aplicației, MATLAB-ul oferă și posibilitatea generării de ferestre proprii prin deschiderea interfeței de generare a ferestrelor (apelând `wintool` sau `window` fără parametri).

De exemplu, fereastra trapezoidală cu 128 de eșantioane (folosită uneori ca alternativă în locul fereastrelor *Bartlett* sau *Hamming*), poate fi generată astfel:

```
>> window
(în interfață) Name:  trapez
(în interfață) Type:  User Defined
(în interfață) MATLAB code:  min(2*triang(128),1);
(în interfață) Apply %pentru generarea ferestrei
(în interfață) Save to workspace %pentru incarcarea ei in workspace
```



- Să se ferestruiască semnalul *MaryPoppins.wav* în segmente de câte $L = 512$ eșantioane, cu suprapunere de $S = 128$ eșantioane.

```
[semnal,Fs] = audioread('MaryPoppins.wav');
L = 512; S = 128; pas = L-S;

segmente = [];
for i = 1:pas:length(semnal)-L+1
    segmente = [segmente, semnal(i:i+L-1,:)];
end
```

Fiecare coloană din matricea `segmente` conține câte un segment de lungime L din semnal, echivalent cu ferestruirea cu o fereastră dreptunghiulară (`rectwin`) . Dacă pentru prelucrările ulterioare este nevoie ca nu toate eșantioanele să fie la fel de semnificative, se va genera o fereastră de ponderi, iar eșantioanele fiecărui segment vor fi ajustate conform acesteia.

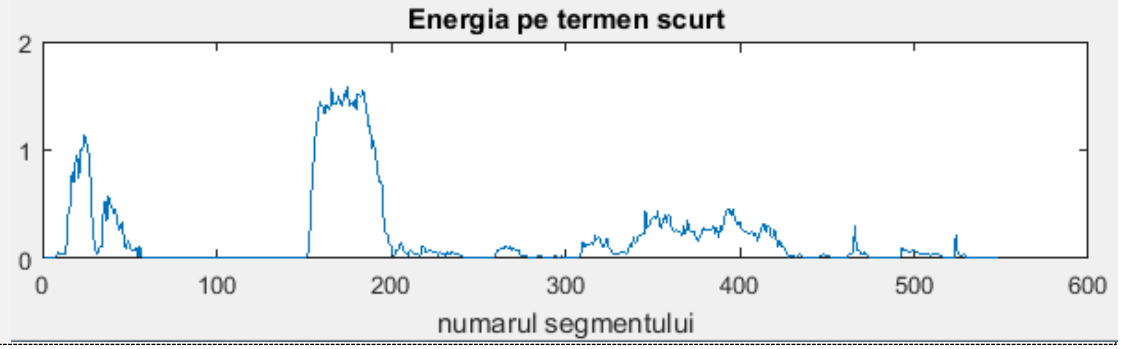
- Să se pondereze segmentele cu câte o fereastră Hann.

```
w = hann(L);
N = size(segmente,2); % numarul de segmente
ferestruite = segmente.* repmat(w,1,N);
```

Având semnalul împărțit în segmente de câte L eșantioane, ponderate conform necesităților, prelucrările de mai departe depind de natura aplicației. De exemplu, pentru identificarea segmentelor de liniște din semnal, se poate folosi calculul energiei pe termen scurt, definit ca suma pătratelor valorilor eșantioanelor din segment. Orice segment cu energie mai mică decât un prag poate fi considerat liniște. Folosirea unei ferestre, alta decât cea dreptunghiulară, reduce influența eșantioanelor marginale, astfel că un eșantion de valoare mare, dar aflat departe de centrul segmentului, nu va influența semnificativ decizia.

- Să se calculeze și să se afișeze energia pe termen scurt.

```
Energia = sum(ferestruite.^2,1);
figure, plot(Energia), title('Energia pe termen scurt')
xlabel('numarul segmentului')
```



11.4 Spectrograma

Transformata Fourier este importantă în prelucrarea semnalelor, însă nu suficientă pentru analiza semnalelor dinamice, a căror structură spectrală variază în timp. Pentru astfel de semnale (cum ar fi semnalele vocale), este mai utilă generarea de spectre folosind eșantioanele din segmente de semnal adiacente sau parțial suprapuse, astfel încât să se extragă informația spectrală dar să se și păstreze evoluția acesteia în timp. *Spectrograma* este construită prin alăturarea acestor spectre într-un grafic tridimensional care are pe o axă timpul, pe alta frecvența, iar pe cea de-a treia axă amplitudinea sau puterea spectrală. Pentru o vizualizare mai ușoară, spectrograma este adesea reprezentată ca o imagine bidimensională (timp-frecvență), componenta de amplitudine sau putere fiind indicată prin intensitatea culorii. In MATLAB, funcția de generare a spectrogramei are următorii parametri:

```
S = spectrogram(X,WINDOW,NOVERLAP,NFFT,Fs)
```

- `X` este semnalul pentru care este calculată spectrograma
- WINDOW este parametrul pentru ferestruire. Dacă acesta este o valoare, semnalul este împărțit în ferestre de lungime egală cu acea valoare și o fereastră Hamming este aplicată pentru îmbunătățirea reprezentării spectrale. Dacă WINDOW este un vector, semnalul este împărțit în segmente de lungime egală cu lungimea vectorului WINDOW, iar valorile acestuia sunt considerate eșantioane ale ferestrei.
- NOVERLAP este lungimea (numărul de eșantioane) pe care două segmente consecutive de semnal se suprapun.
- NFFT reprezintă numărul de puncte pentru calculul *Transformatei Fourier Discrete*.
- `Fs` reprezintă frecvența de eșantionare.
- Pentru a vizualiza axa frecvențelor pe verticală, se poate adăuga `'yaxis'`

Exemplul didactic pentru exemplificarea spectrogramei este semnalul *chirp*, deoarece pentru acest semnal este evident avantajul folosirii spectrogramei față de transformata Fourier aplicată întregului semnal, afișată mai jos.

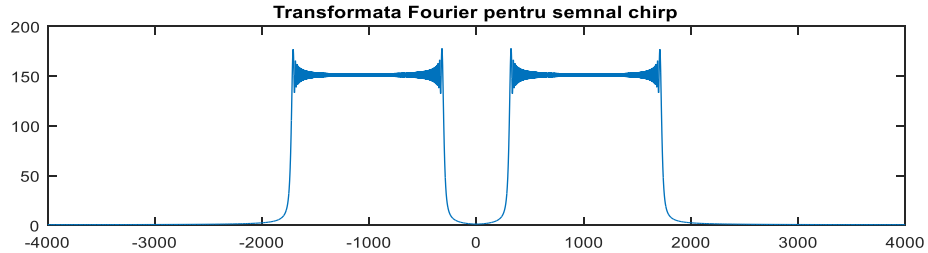
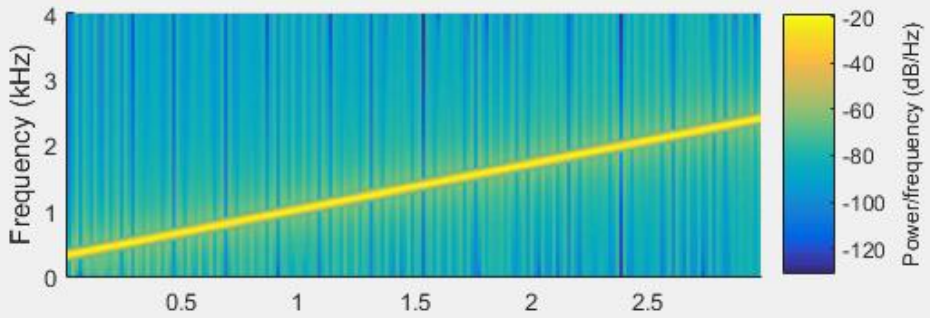


Figura 11.2. Transformata Fourier pentru semnalul *chirp*

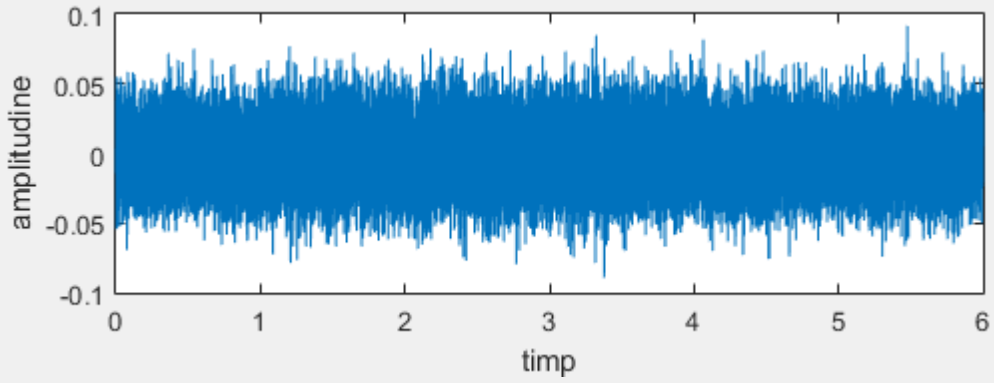
- Să se genereze spectrograma semnalului *chirp* liniar care pornește de la frecvența F1 = 300Hz, ajunge la frecvența F2 = 1000Hz la *t*=1s și este eșantionat cu frecvența de eșantionare Fs = 8kHz; spectrograma să folosească fereastra Hamming de 256 eșantioane cu suprapunere de 50% între segmente adiacente.

```
Fs = 8000;
t = 0:1/Fs:3;
semnal = chirp(t,300,1,1000);
spectrogram(semnal, 256, 128, 256, Fs, 'yaxis');
```



- Fie o sinusoidă eșantionată cu frecvența *Fs* = 8kHz, înecată în zgomot alb Gaussian. Să se afișeze grafic semnalul în timp și spectrograma, în care se pot identifica parametrii sinusoidei.

```
load('SinusoidaInZgomot');
n = 0:length(semnal)-1;
Fs = 8000;
figure(1)
plot(n/Fs,semnal); xlabel('timp'); ylabel('amplitudine');
```




```
L = 1024; w = hamming(L);
figure(2)
spectrogram(semnal,w,128,L,Fs,'yaxis');
```

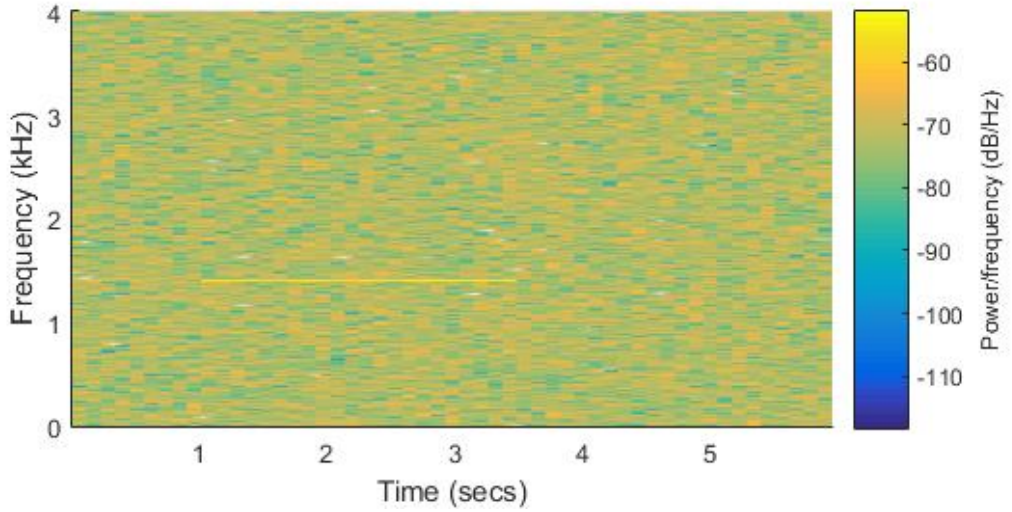


Figura 11.3. Spectrograma sinusoidei înecate în zgomot

După cum se poate observa pe spectrogramă, sinusoida este pe frecvența de 1400 Hz, începe la aproximativ 1s și se termină la aproximativ 3.5s (linia galbenă).

11.5 Analiza filtrelor

Filtrele numerice pot fi împărţite în două mari categorii:

- *filtre cu Răspuns Finit la Impuls* (RFI) sau FIR (**F**inite **I**mpulse **R**esponse)
- *filtre cu Răspuns Infinit la Impuls* (RII) sau IIR (**I**nfinite **I**mpulse **R**esponse).

Un filtru recursiv (*Infinite Impulse Response* – IIR) este un sistem numeric descris în domeniul timp de următoarea relație (*ecuație cu diferențe finite*):

$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k] - \sum_{i=1}^{M} a_i \cdot y[n-i] \tag{11.1}$$

- $x[n]$ reprezintă o secvență de date de intrare
- $y[n]$ reprezintă o secvență de date de ieșire
- b_k și a_i reprezintă coeficienții filtrului

Observație: în cazul în care toți coeficienții a_i sunt nuli, formula de mai sus va descrie un filtru FIR.

Aplicând transformata $\mathcal{Z}$ și folosind proprietatea de întârziere, se ajunge astfel la formula generală a *funcției de transfer* pentru un filtru IIR:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_M z^{-M}} \tag{11.2}$$

Deoarece MATLAB-ul pornește numerotarea coeficienților de la 1, normarea $a_0 = 1$ se traduce în documentația MATLAB prin `a(1) = 1`.
O modalitate de analiză a acestui filtru este cu ajutorul funcției MATLAB `freqz`.

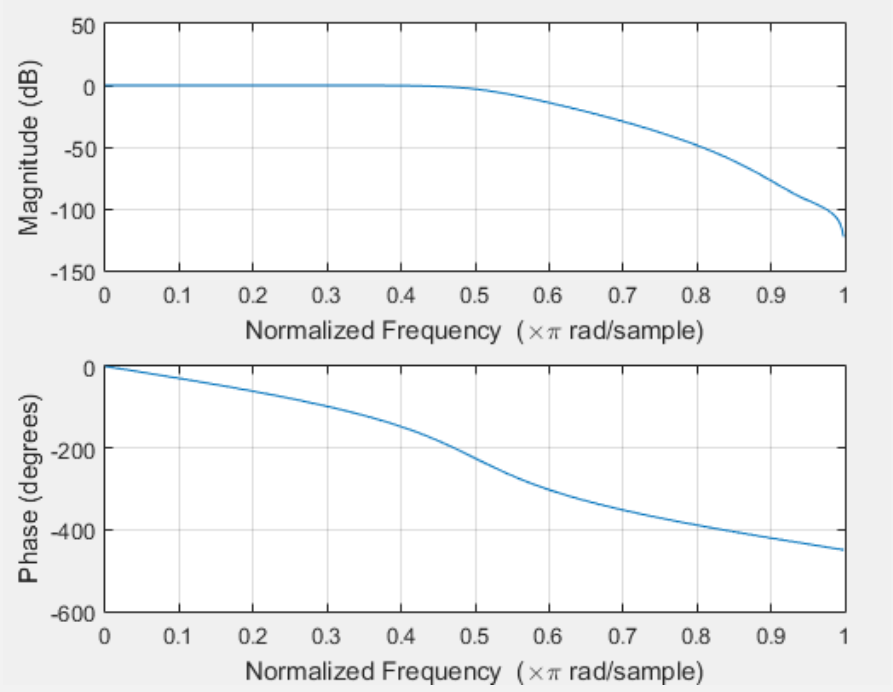
Sintaxă: `[H,W] = freqz(b,a,N)`
Se reprezintă comportamentul în frecvență al filtrului descris de coeficienții `a` și `b`, în `N` puncte. Daca `N` este omis, valoarea predefinita este `N` = 512. `H` este comportamentul în frecvență, un vector de valori complexe, iar `W` este vectorul de frecvențe normate, în intervalul [0 ... pi/2). Dacă funcția `freqz` este apelată fără parametri de ieșire, atunci se afișează magnitudinea (în db) și faza desfașurată (în grade) a filtrului.

- Un filtru are următorii coeficienți:

`b` = [0.0528 0.2639 0.5279 0.5279 0.2639 0.0528] și
`a` = [1.0000 -0.0000 0.6334 -0.0000 0.0557 -0.0000]

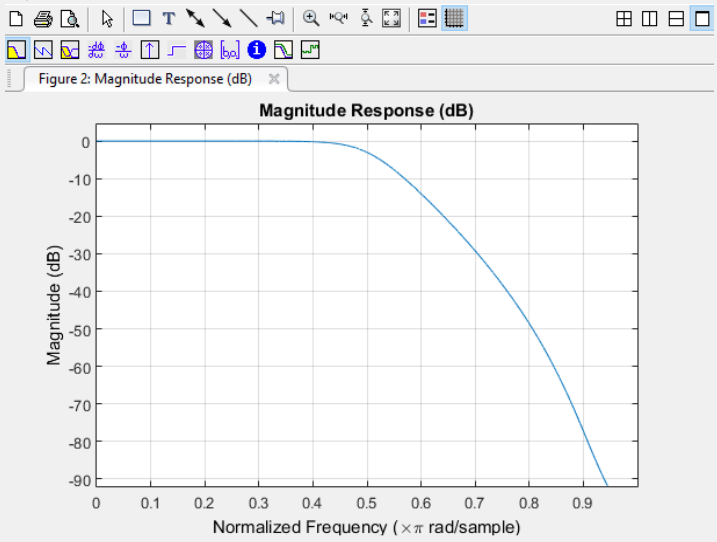
Să se analizeze acest filtru în frecvență.

```
>> b =  [0.0528    0.2639    0.5279    0.5279    0.2639    0.0528];
>> a =  [1.0000   -0.0000    0.6334   -0.0000    0.0557   -0.0000];
>> freqz(b, a)
```



Ca și în cazul ferestrelor, MATLAB facilitează analiza filtrelor printr-o interfață interactivă apelată cu `fvtool` *(Filter Visualization Tool)*, care permite utilizatorilor să aibă informații mai detaliate despre filtru.

```
>> fvtool(b,a)
```



Tabel 11.2. Descrierea opțiunilor din al doilea rând de butoane al interfeței `fvtool`

Descrierea opțiunilor din interfață	Funcție alternativă
Răspunsul filtrului (magnitudine) în *db*	
Răspunsul filtrului (faza desfășurată) în radiani	`phasez(b,a)`
Răspunsul de magnitudine și de fază în același grafic	
Intârzierea de grup a filtrului	`grpdelay(b,a)`
Intârzierea de fază a filtrului	`phasedelay(b,a)`
Răspunsul la impuls	`impz(b,a)`
Răspunsul la treapta unitate	`stepz(b,a)`
Diagrama poli-zerouri	`zplane(b,a)`
Valorile coeficienților `b` și `a`	
Sinteza informațiilor despre filtru și costurile de implementare (număr de operații necesare)	
Răspunsul filtrului ținând cont de erorile de rotunjire	
Mărimea erorilor de rotunjire (în *db*)	

Ca și în cazul `wvtool`, click-urile pe grafice afișează valoarea funcției în acel punct. Deoarece acesta este în primul rând un tutorial de MATLAB, în continuare vom folosi funcția `freqz` pentru a exemplifica tipurile de filtre pre-implementate în MATLAB, deși pentru aplicații practice ar putea fi utile și informațiile suplimentare accesibile din interfața `fvtool`.

11.6 Proiectarea filtrelor

MATLAB conține atât interfețe interactive cât și funcții pentru implementarea diverselor tipuri de filtre. Toolboxul ***Filter Design and Analysis*** oferă utilizatorilor o gamă largă de filtre și de parametri care pot fi ajustați în vederea construirii filtrului cel mai potrivit pentru aplicație. Insă cu toată flexibilitatea oferită de această interfață, există situații în care filtrul trebuie generat în timpul rulării unei simulări, cu caracteristici dependente de variabile calculate în timpul simulării. Pentru astfel de cazuri este necesară cunoașterea funcțiilor MATLAB care generează cele mai uzuale tipuri de filtre.

Pentru descrierea filtrelor, vor fi folosite următoarele notații:

- `B,A` sau `b,a` = coeficienții polinoamelor care descriu filtrul
- `N` = ordinul filtrului
- W_n = frecvența normată de tăiere; W_n trebuie să fie un număr în intervalul [0...1], iar frecvența reală de tăiere a filtrului este $F_n = W_n * F_s/2$.
- W_p = frecvența normată limită a benzii de trecere; W_p trebuie să fie un număr în intervalul [0...1], iar frecvența limită reală a benzii de trecere a filtrului este $F_{pass} = W_p * F_s/2$.
- W_s = frecvența normată limită a benzii de oprire; W_s trebuie să fie un număr în intervalul [0...1], iar frecvența limită reală a benzii de oprire a filtrului este $F_{stop} = W_s * F_s/2$.
- `Rp` = riplul maxim acceptabil (*db*) pentru banda de trecere
- `Rs` = atenuarea minimă (*db*) pentru banda de oprire
- OPT = parametru opțional care specifică tipul de filtru (trece-jos, trece-sus, trece-bandă sau oprește-bandă); dacă este omis, se generează filtre trece-jos.

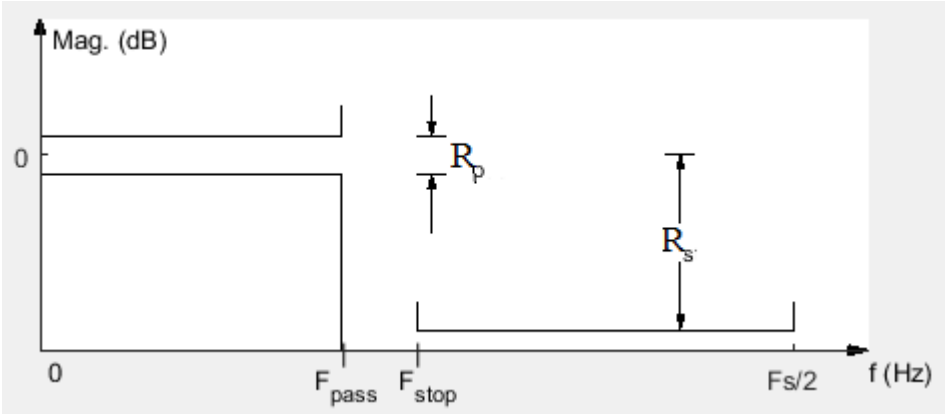


Figura 11.4. Exemplificarea constrângerilor pentru un filtru trece-jos

Tabel 11.3. Denumirea filtrelor, sintaxa de calibrare a filtrelor (funcții utile determinării ordinului minim al filtrelor pentru care sunt respectate constrângerile impuse de utilizator) și de generare a filtrelor pre-implementate.

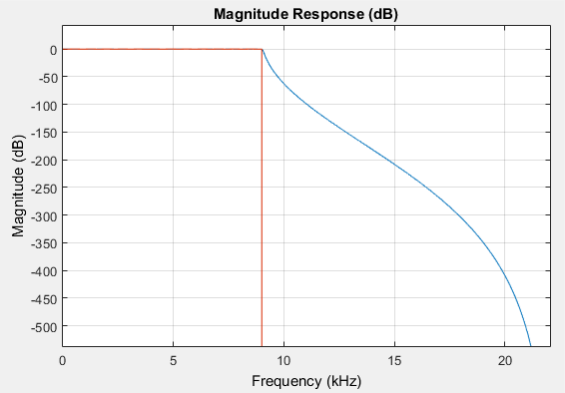
Denumiri	Determinarea ordinului filtrelor	Sintaxă generare filtre
Butterworth	[N, Wn] = **buttord**(Wp, Ws, Rp, Rs);	[B,A] = **butter**(N,Wn,OPT);
Chebyshev tip I si II	[N, Wp] = **cheb1ord**(Wp, Ws, Rp, Rs); [N, Ws] = **cheb2ord**(Wp, Ws, Rp, Rs);	[B,A] = **cheby1**(N,Rp,Wp,OPT); [B,A] = **cheby2**(N,Rs,Ws,OPT);
Eliptice	[N, Wp] = **ellipord**(Wp, Ws, Rp, Rs);	[B,A] = **ellip**(N,Rp,Rs,Wp,OPT);
Filtre FIR		B = **fir1**(N,Wn); B = **fir2**(N,F,M); B=**firpm**(N,F,M); B=**firls**(N,F,M);
Filtre speciale		[B,A] = **iirnotch**(Wo,BW,A); [B,A] = **iirpeak**(Wo,BW,A); [B,A] = **iircomb**(Wo,BW,A,TYPE);

Dacă niciunul dintre tipurile de filtre de mai sus nu satisface constrângerile necesare aplicației, există în MATLAB și o funcție generică de generare a filtrelor digitale, pentru care parametrii sunt chiar constrângerile: `designfilt`.

Funcția `designfilt`, apelată fără parametri, pornește o interfață de selecție a tipului de filtru, însă toți parametrii pot fi incluși în momentul apelării. Primul parametru trebuie să fie tipul de filtru, iar următorii parametri vin în perechi denumire-valoare. Variabila de ieșire a funcției `designfilt` este un obiect de tip *digital filter*, care nu poate fi vizualizat cu `freqz()`, ci doar cu `fvtool()`.

- Să se genereze și să se vizualizeze un filtru IIR trece-jos de ordin 17, cu banda de trecere până la frecvența de 9 kHz; frecvența de eșantionare *Fs = 44.1 kHz*.

```
lpFilt = designfilt('lowpassiir', 'FilterOrder', 17, ...
             'PassbandFrequency', 9000, 'PassbandRipple', 0.2,...
             'SampleRate', 44100);
fvtool(lpFilt)
```



11.7 Filtrarea semnalelor

Suportul matematic al filtrării este convoluția între semnalul de intrare `x` și răspunsul la impuls al filtrului `h`. MATLAB oferă posibilitatea realizării convoluției ca operație matematică, cu ajutorul funcțiilor `conv` și `conv2` (pentru semnale bidimensionale). Insă convoluția nu este cea mai potrivită operație pentru prelucrarea semnalelor, deoarece analiza filtrelor se face pornind de la coeficienții polinomiali care definesc filtrele, dar și pentru că filtrele IIR prin definiție au un răspuns la impuls infinit. De aceea, în MATLAB există funcția `filter`, care realizează filtrarea semnalelor unidimensionale pornind de la descrierea polinomială a filtrelor, iar semnalul de ieșire are aceeași lungime ca semnalul de intrare.

Sintaxa:

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

Funcția `filter` poate primi ca parametri de intrare, pe lângă semnalul `x` care se dorește a fi filtrat, atât vectorii de coeficienți `b` și `a` care descriu filtrul, cât și un obiect de tip *digitalFilter*, obținut prin apelarea funcției `designfilt`.

Funcția `filter` implementează ecuația cu diferențe finite standard, normalizând coeficienții prin împărțire la `a(1)` în cazul în care valoarea acestui coeficient este diferită de 1.

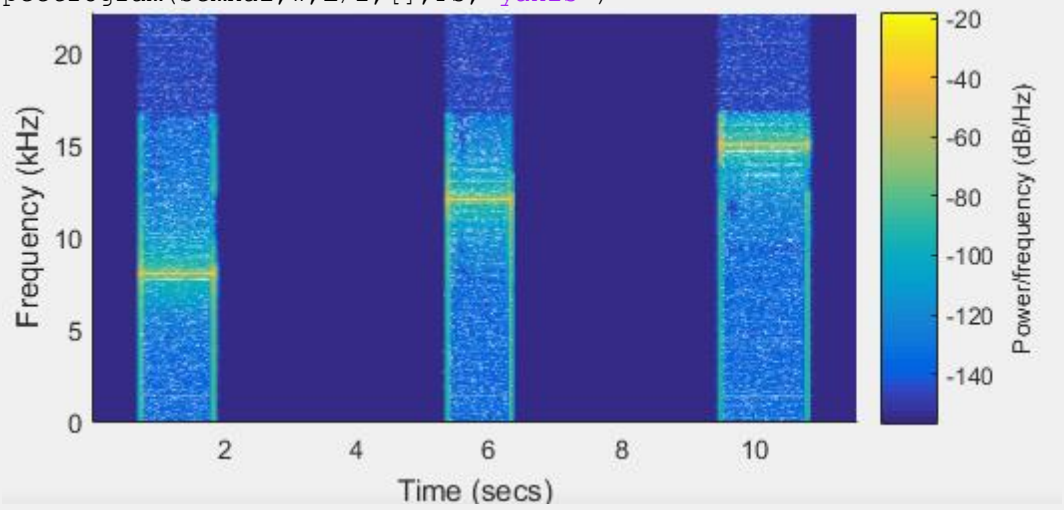
Filtrarea se face de-a lungul primei dimensiuni nesingulare. Cu alte cuvinte, indiferent dacă vectorul `x` este vector coloană sau vector linie, filtrarea va produce rezultate identice, iar vectorul-rezultat `y` va avea aceleași dimensiuni ca și vectorul `x`. In cazul în care se dorește filtrarea mai multor semnale în același timp, funcția `filter` permite acest lucru, daca vectorii de intrare $x_1 \ldots x_N$ sunt vectori coloană de aceeași lungime. In acest mod se poate realiza filtrarea tuturor canalelor audio ale unui semnal în același timp.

- Să se folosească filtrul `lpFilt` (generat anterior) pentru filtrarea ambelor canale din semnalul `zgomot.wav`

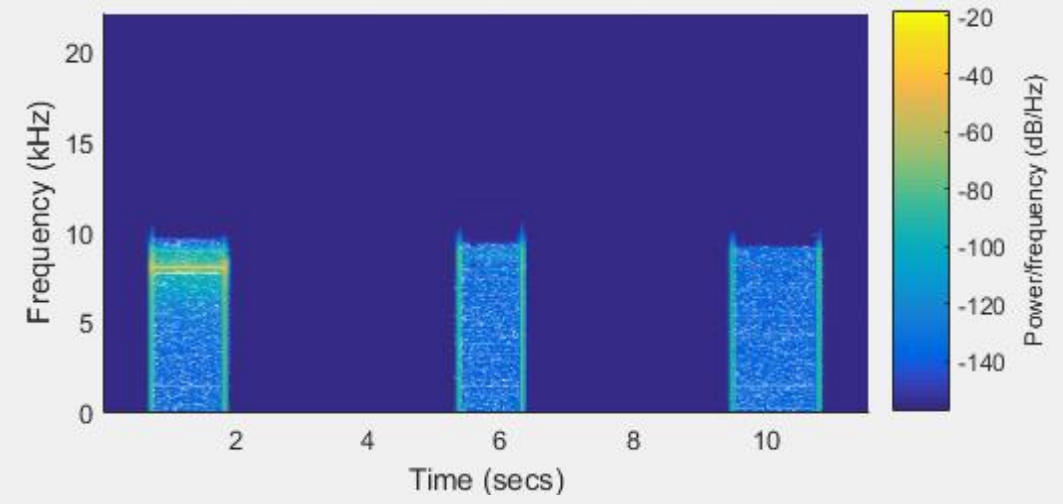
```
[semnal,Fs] = audioread('zgomot.wav');
y = filter(lpFilt,semnal);
% verificarea numarului de canale pentru semnalul de iesire
disp(size(y))
      510876           2
```

- Să se folosească filtrul `lpFilt` (generat anterior) pentru eliminarea sinusoidelor de frecvențe înalte din semnalul `zgomot.wav`

```
[semnal,Fs] = audioread('zgomot.wav');
semnal = semnal(:,1); % pastrarea unui singur canal

L = 4096; w = hann(L);
figure(1)
spectrogram(semnal,w,L/2,[],Fs,'yaxis')
```




```
y = filter(lpFilt,semnal);
figure(2)
spectrogram(y,w,L/2,[],Fs,'yaxis');
```



Pentru aplicațiile pentru care există constrângerea să nu fie distorsiuni de fază, o operație uzuală în procesarea de semnal este filtrarea dublă (înainte și înapoi prin filtru) realizată în MATLAB cu ajutorul funcției `filtfilt`. Această operație are efectul anulării diferențelor de fază și în consecință a întârzierii de grup. Din punct de vedere al atenuării, trecerea dublă prin filtru atenuează componentele spectrale cu pătratul atenuării originale.

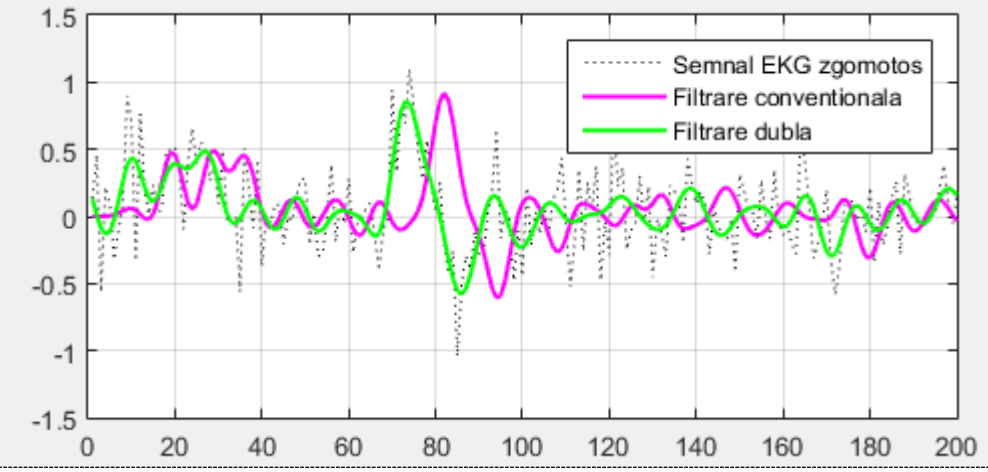
Sintaxa:

```
y = filtfilt(b,a,x);
y = filtfilt(digitalFilter,x);
```

Funcția `filtfilt` poate primi ca parametri de intrare, pe lângă semnalul `x` care se dorește a fi filtrat, atât vectorii de coeficienți `b` și `a` care descriu filtrul, cât și un obiect de tip *digitalFilter*, obținut prin apelarea funcției `designfilt`.

- Să se filtreze un semnal EKG cu ajutorul unui filtru Butterworth trece-jos de ordin 10 cu frecvența de tăiere normată la 0.25. Să se afișeze rezultatele filtrării simple și duble peste semnalul original.

```
x = ecg(200).'+0.25*randn(200,1);
[b,a] = butter(10,0.25);
y1 = filter(b,a,x);
y2 = filtfilt(b,a,x);
figure(1)
    plot(x,'k:'); grid on ;
    hold on
    plot(y1,'m','LineWidth',1.5);
    plot(y2,'g','LineWidth',1.5);
    hold off
    legend('Semnal EKG zgomotos','Filtrare conventionala','Filtrare
dubla');
```



- Fie un semnal audio monocanal eșantionat la 8kHz. Să se schimbe frecvența de eșantionare la 48kHz prin repetarea fiecărui eșantion de câte 6 ori.

```
>> [oaie1,Fs] = audioread('F:\Sheep.wav');
>> sound(oaie1, Fs)
>> Fs
Fs =
        8000
```

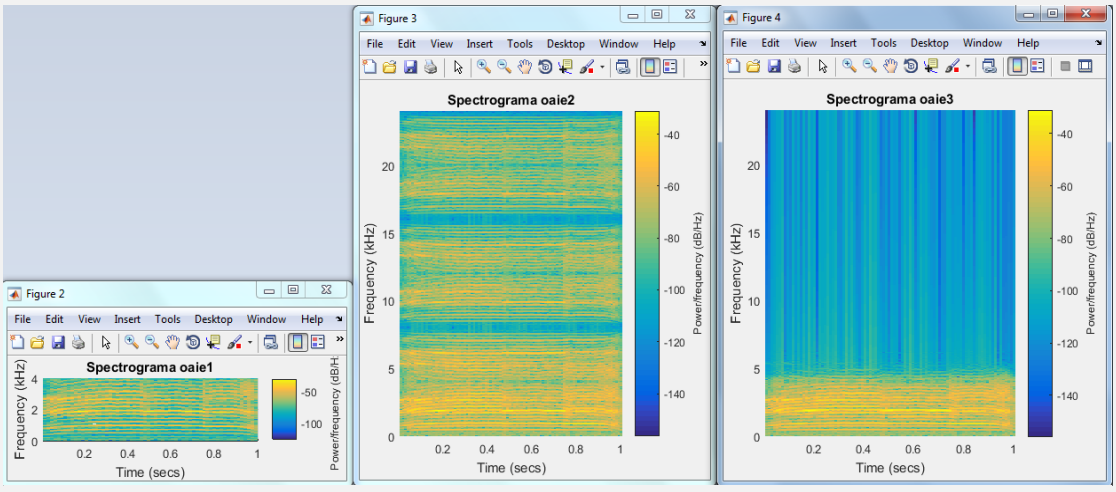
Cea mai rapidă implementare se face cu produs Kronecker între semnalul original `oaie` și vectorul `ones(6,1)`.

Produsul Kronecker, având în MATLAB sintaxa `C = kron(A,B)`, înmulțește fiecare element din `A` cu matricea `B`, apoi concatenează rezultatele în `C`. Atunci când matricea `A` este un vector linie/coloană, iar matricea `B` este un vector linie/coloană de ***b*** unități, rezultatul devine repetarea fiecărui element din `A` de câte ***b*** ori.

```
>> oaie2 = kron(oaie1,ones(6,1));
>> sound(oaie2, 6*Fs)
```

Semnalul conține artefacte audio datorită procedurii prin care s-a schimbat frecvența de eșantionare. Să se elimine artefactele audio prin filtrarea semnalului cu un FTJ de tip Chebyshev, de ordin 10, cu o atenuare de 0.1dB în banda de trecere și o frecvența de tăiere de 4kHz.

```
[b,a] = cheby1(10,0.1,1/6);
oaie3 = filter(b,a,oaie2);
sound(oaie3, 6*Fs)
```



12. Elemente de prelucrare și analiză a Semnalelor Medicale

Reprezentarea unei imagini digitale grayscale

Din punct de vedere matematic, o imagine grayscale este reprezentată ca un semnal bidimensional - în speță, o funcție `f(x,y)` de două variabile, unde variabila `x` reprezintă coordonata pentru linie, iar variabila `y` reprezintă coordonata pentru coloană. Funcția `f(x,y)` redă cantitatea de luminozitate pentru fiecare pereche de coordonate spațiale `(x,y)`. În spațiul continuu, suportul funcției `f(x,y)` este nelimitat, cu număr infinit de linii și coloane. În practică însă, se operează cu definiția mulțimii `f(x,y)` în plan discret - variabilele și valorile funcției luând doar valori discrete. Cu alte cuvinte, funcția este o matrice cu un număr finit de linii și coloane.

Așa cum a fost prezentat în capitolul 5, limbajul MATLAB este orientat pentru operarea cu structuri matriceale, având multe funcții integrate pentru procesarea directă, rapidă și optimizată a acestora. Astfel, o imagine grayscale digitală este asociată cu o matrice `A` de `m` linii si `n` coloane:

$$A = \begin{bmatrix} a_{11}\ a_{12}\ a_{13} \dots a_{1n} \\ a_{21}\ a_{22}\ a_{23} \dots a_{2n} \\ \dots \dots \dots \dots \dots \dots \dots \dots \\ a_{m1}\ a_{m2}\ a_{m3} \dots a_{mn} \end{bmatrix}$$

Elementul de bază al imaginii digitale se numește pixel (de la terminologia din literatura de specialitate *picture element*). Un pixel (a_{ij}) este definit prin coordonatele sale spațiale (linia `i`, coloana `j`) și valoarea de luminanță (valoarea nivelului de gri) asociată coordonatelor `i` și `j`. Valorile nivelurilor de gri se exprimă în mod uzual pe 8 biți, cu alte cuvinte sunt disponibile 256 de niveluri de gri pentru intensitatea fiecărui pixel. Dimensiunea imaginii (matricei) este redată prin produsul dintre numărul de pixeli dintr-o linie și numărul de pixeli dintr-o coloană, pentru exemplul de față dimensiunea fiind de *m*x*n* pixeli.

Rezoluția nivelurilor de gri

În urma procesului de digitizare, fiecare pixel din imagine are asignat pentru reprezentarea valorii un număr de biți. Această valoare este valoarea luminanței imaginii în pixelul respectiv. Uzual, în practică, sunt utilizate 2^8 niveluri de cuantizare (sau terminologia de 8 biți per pixel) sau niveluri de gri pentru redarea luminanței unui pixel. Intervalul dinamic al scalei de gri este intervalul în care pot lua valori pixelii din imagine: de exemplu, de la 0 la 255 pentru o imagine cu 8 biți/pixel.

Fiecare valoare din intervalul 0 ÷ 255 este o nuanță diferită de gri pe o scală de la negru (care are, prin convenție, valoarea 0) la alb (valoarea maximă de luminanță - 255). În unele aplicații însă, cum ar fi domeniul imagisticii medicale, se dorește o calitate bună a imaginii pentru a reda informația cu o acuratețe cât mai mare; de aceea, în aceste cazuri, sunt utilizate rezoluții de 10, 12, 16, respectiv 24 biți/ pixel.

Rezoluția spațială

Pentru ca o imagine să fie cât mai calitativă, este de dorit, ca efectul de eșantionare din cadrul procesului de digitizare al imaginii să afecteze într-o măsură cât mai mică detaliile dintr-o imagine; acest lucru presupune ca cel puțin două eșantioane să redea cel mai mic detaliu din imagine. Rezoluția spațială este redată, în practică, numeric, prin numărul maxim de dungi albe/negre, care se succed într-un milimetru, percepute de sistemul vizual uman fără ca acesta să le estompeze în niveluri/ nuanțe intermediare de gri. Unitatea de măsură utilizată este $lpmm^{-1}$ – perechi de linii/ milimetru. Pentru determinarea rezoluției spațiale a unei imagini este necesar să se determine corespondența pixel - unitate de măsură (milimetri), cu alte cuvinte dimensiunea unui pixel în milimetri. În imagistica medicală, această măsură este obținută în urma operației de calibrare.

12.1 Particularitățile imaginilor medicale

Întrucât acest capitol prezintă funcții de bază utile în procesarea și analiza imaginilor medicale, este de interes o prezentare succintă a particularităților acestor tipuri de imagini care va explica alegerea operațiilor de analiză următoare.

Imagistica medicală reprezintă ansamblul de tehnici/ procese utilizate pentru realizarea imaginilor interne ale corpului uman, procesul de sondare efectuându-se într-o manieră non-invazivă. În acest sens, imaginea medicală este o fereastră către interiorul organismului uman, dar această fereastră nu redă cu exactitate informația din interior, având la rândul ei diferite limitări. Metode diferite de imagistică medicală redau caracteristici diferite ale organismului investigat.

Calitatea imaginii și vizibilitatea structurii interne depind de mai mulți factori: caracteristicile și variabilele echipamentului de achiziție (cea mai mare parte a variabilelor sunt cantități fizice reglabile asociate cu procesul de achiziție aferent: *kilovoltaj* - în cazul radiografiei, *câștig* - în cazul ultrasunetelor și *timp de ecou* - în cazul metodei de rezonanță magnetică nucleară), abilitățile operatorului precum și limitările fizice ale fiecărei metode legate de timpul de expunere al achiziției, concentrația de radiație utilizată, etc. Calitatea unei imagini depinde de mai mulți factori esențiali: contrast, blur (grad estompare), zgomot, artefacte și distorsiuni.

Sintagma de *îmbunătățirea calității unei imagini medicale* presupune atât îmbunătățirea contrastului precum și minimizarea factorilor inerenți de blur, zgomot și alte artefacte/ distorsiuni care apar în procesul de achiziție al imaginii. Conform definiției de mai sus, imagistica medicală reprezintă procesul de transpunere al caracteristicilor țesutului într-o imagine. Cu alte cuvinte, rolul fiecărui sistem de imagistică este de a transpune o anumită caracteristică de țesut într-o imagine cu niveluri de gri sau color, respectiv o secvență video. Dacă contrastul imaginii obținute este adecvat, atunci obiectul va fi vizibil. Gradul de contrast al imaginii depinde atât de caracteristicile obiectului cât și de caracteristicile sistemului de achiziție. De aceea, creșterea contrastului este factorul esențial în îmbunătățirea calității imaginii medicale.

În figura de mai jos sunt câteva exemple de imagini medicale, care vor face subiectul aplicațiilor dezvoltate ulterior în acest capitol.

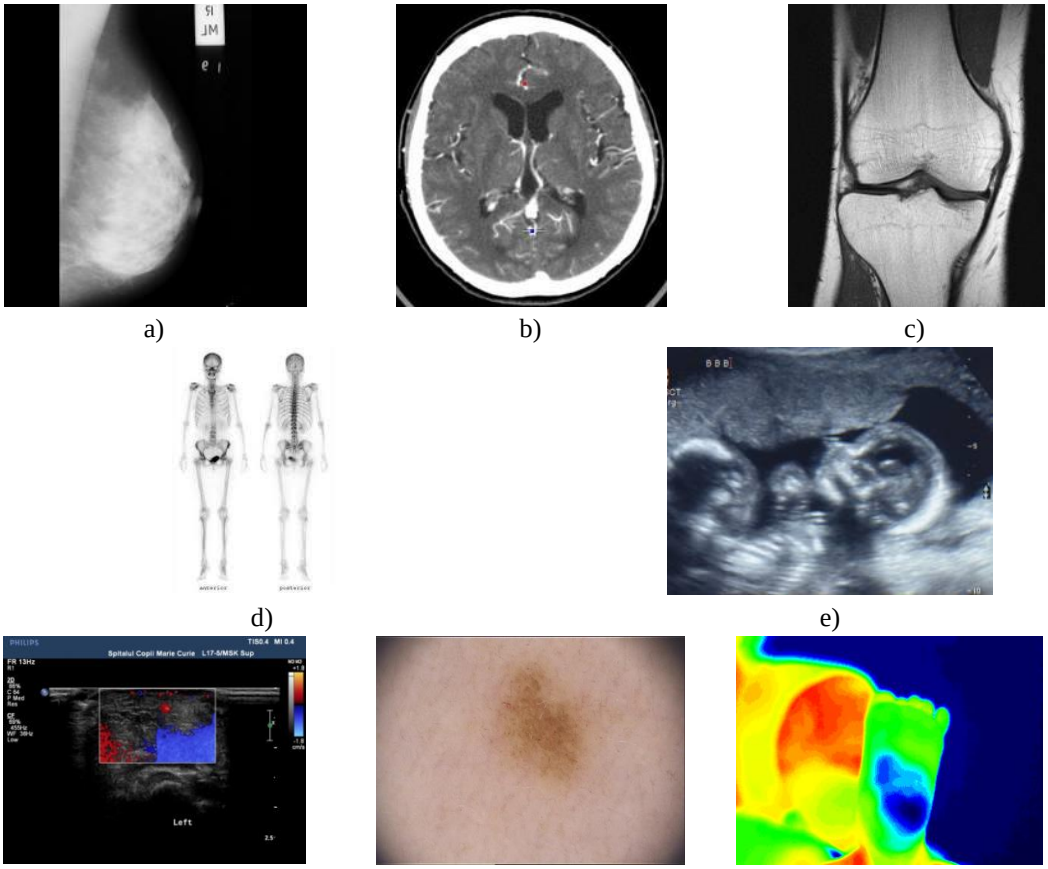


Figura 12.1. Exemple de imagini medicale: a) Mamografia (radiografia sânului) [7]; b) Tomografia computerizată craniană [8]; c) Rezonanța magnetică nucleară a genunchiului [9]; d) Scanare a întregului corp prin tehnica de medicină nucleară [10]; e) Ultrasonografie fetală [11]; f) Ecografie Doppler a unui hemangiom; g) Dermatoscopie a unui nev (metoda de microscopie electronică); h) Termografia unui nev

12.2 Funcții utile în procesarea imaginilor medicale

12.2.1 Îmbunătățirea contrastului

Operații punctuale de îmbunătățire a contrastului

Operațiile punctuale de modificare a contrastului (numite și transformări ale nivelului de gri) sunt asocieri (mapping, în engleză) ce leagă nivelul de gri original ***u*** de noua sa valoare ***v***. O asemenea asociere este o funcție conform ecuației de mai jos:

$$\mathbf{v = T(u)};\ \mathbf{u} \in [0;M-1]$$

Funcția **T** trebuie să îndeplinească următoarele condiții:

- Să păstreze gama admisibilă de valori a imaginii (dacă nivelurile de gri au fost reprezentate pe **M** niveluri de cuantizare, atunci $\mathbf{0 \le T(x) \le M-1}$, $\forall\ \boldsymbol{x} \in [0; M-1]$);
- Transformarea **T** să fie monoton crescătoare (sau descrescătoare) pentru a păstra ordinea între nivelurile de gri (componentele închise din imagine să rămână închise și după transformare).

12.2.1.1 Modificarea liniară a contrastului

Ajustarea valorilor de intensitate a imaginii este realizată în MATLAB cu funcția `imadjust`, al cărei grafic și ecuație le regăsim mai jos:

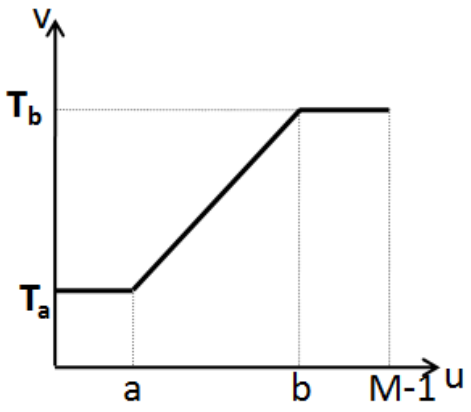


Figura 12.2. Graficul asociat funcției de modificare a contrastului `imadjust`

Ecuația funcției de modificare liniară a contrastului asociată funcției `imadjust`:

$$v = \begin{cases} T_a, 0 < u \le a \\ T_a + \dfrac{(T_b - T_a)}{(b-a)} \cdot (u-a), a < u \le b \\ T_b, b < u \le M-1 \end{cases}$$

Sintaxă:

```
J = imadjust(I, [low_in, high_in],[low_out, high_out])
```

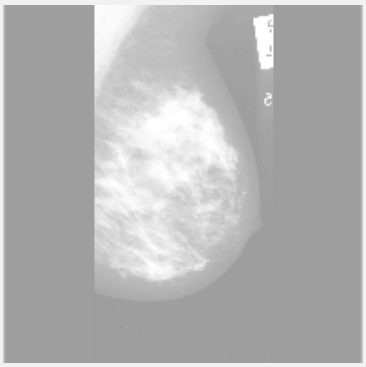
Funcția `imadjust` mapează valorile din imaginea `I` în noile valori `J` astfel încât valorile din intervalul `[low_in, high_in]` aferente imaginii inițiale corespund intervalului `[low_out, high_out]` din imaginea rezultat.

Observație: dacă în loc de [`low_out, high_out`] se scrie doar [ ], atunci se consideră intervalul [0, 1]

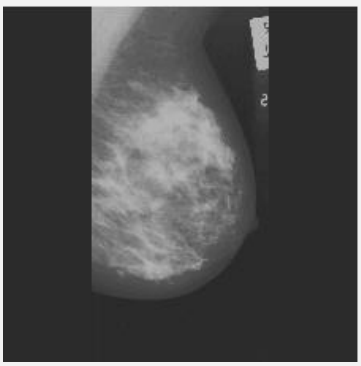
- Să se modifice contrastul unei imagini folosind funcția `imadjust`.

Se vor aduce toate valorile din intervalul `[0.5 1]` în intervalul `[0 0.7]`.

```
I = imread('eh4.jpg');
figure(1), imshow(I) % afisare imagine originala
% apelarea functiei imadjust
J = imadjust(I, [0.5 1], [0 0.7]);
% afisarea imaginii rezultat
figure(2), imshow(J)
```



Imaginea originală [7]



Imaginea după modificarea liniară a contrastului

Cazuri particulare

1. Binarizarea (Thresholding)

În cazul în care `a = b`, $T_a = 0$ și $T_b = M - 1$, se va obține o operație de binarizare; în acest caz, toți pixelii din imaginea originală mai mici decât pragul `a` vor fi 0 (corespunzător lui negru), iar pixelii cu valori de gri mai mari decât pragul `a` vor fi redați cu alb.

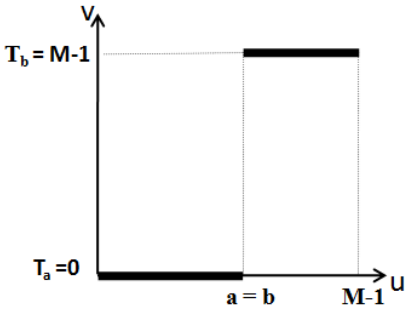
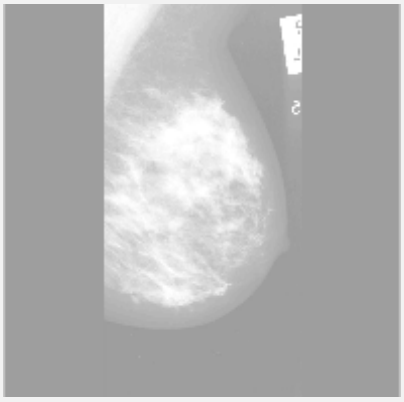


Figura 12.3. Operația de binarizare

- Să se binarizeze o imagine.

```
I = imread('eh4.jpg');
% apelarea functiei imadjust
J = imadjust(I, [0.68 0.68+eps], [ ]);
% afisarea imaginii rezultat
figure(1), imshow(I) % afisare imagine originala
figure(2), imshow(J)
```



Imaginea originală [7]

Imaginea după binarizare

2. Întinderea maximă a contrastului (Contrast Stretching)

Dacă pentru un interval `[a, b]` din imaginea originală, valorile corespunzătoare pragurilor T_a, respectiv T_b din imaginea rezultat sunt $T_a = 0$ și $T_b = M - 1$, atunci operația de modificare a contrastului poartă numele de întindere maximă a contrastului.

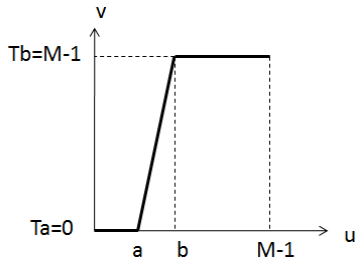
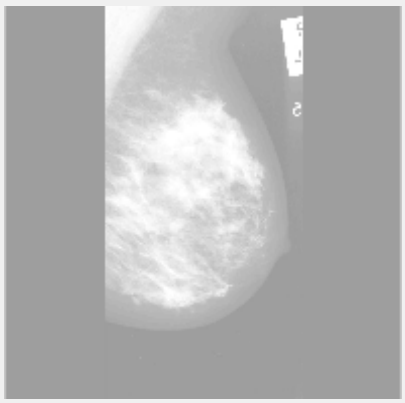


Figura 12.4. Operația de întindere maximă a contrastului

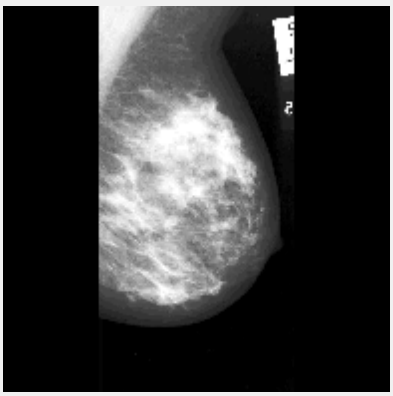
Observație: funcția `stretchlim(I)` returnează un vector de două elemente ale cărui valori redau limita inferioară și limita superioară a nivelurilor de gri dintr-o imagine grayscale. Aceste valori pot fi folosite mai departe pentru operația de îmbunătățire a contrastului.

- Exemplu pentru întinderea maximă a contrastului.

```
I = imread('eh4.jpg');
% apelarea functiei imadjust
J = imadjust(I, stretchlim(I), [ ]);
% afisarea imaginii rezultat
figure(1), imshow(I) % afisare imagine originala
figure(2), imshow(J)
```



Imaginea originală [7]

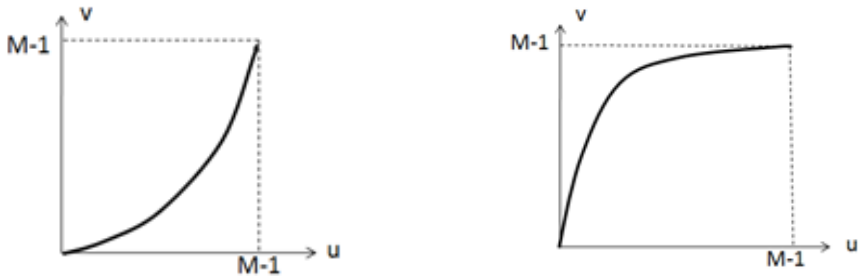


Imaginea obținută după întinderea maximă a contrastului

12.2.1.2 Modificarea neliniară a contrastului

Tehnicile de modificare liniară a contrastului acționează pe pantele intervalelor nivelurilor de gri menționate. În practică, avem însă uneori nevoie de o operație de contrastare diferită a unor intervale de niveluri de gri în raport cu altele. Acest lucru este posibil prin aplicarea unei funcții neliniare de modificare a contrastului.

În cele ce urmează vom da două exemple de operații de modificare neliniară a contrastului bazate pe *operația exponențială*, respectiv *operația logaritmică*.



a) b)

Figura 12.5. Graficele operațiilor de modificare neliniară a contrastului
a) Transformarea exponențială b) Transformarea logaritmică

În MATLAB, implementarea contrastului neliniar se realizează pe baza funcției `imadjust`, iar neliniaritatea (alura curbei) este introdusă de parametrul `gamma`.

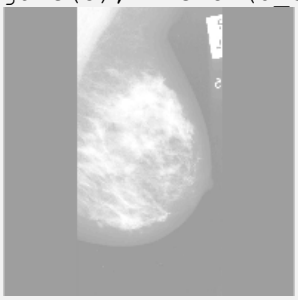
Sintaxă:

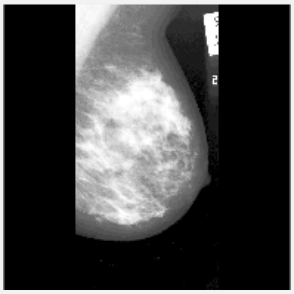
`J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)`

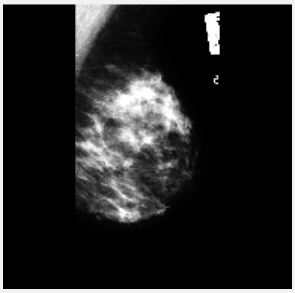
Dacă `gama` < 1 ➔ transformare logaritmică
Dacă `gama` > 1 ➔ transformare exponentiala
Dacă `gama` = 1 ➔ transformare liniară

Exemple:

```
I = imread('eh4.jpg');
% Apelare functie logaritmica: gamma < 1
J_log = imadjust(I, stretchlim(I), [], 0.7);
% Apelare func?ie exponentiala: gamma >1
J_exp = imadjust(I, stretchlim(I), [], 3);
figure(1), imshow(I)
figure(2), imshow(J_log)
figure(3), imshow(J_exp)
```


Imaginea originală [7]


Operația de logaritmare


Operația exponențială

12.2.2 Operația de negativare (Complementul unei imagini)

Această operație calculează negativa (complementul) unei imagini. Practic, pentru o imagine cu niveluri de gri, valoarea pixelului din imaginea rezultat este egală cu diferența dintre nivelul maxim de gri (valoarea 1) și valoarea pixelului considerat din imaginea inițială. Astfel, zonele întunecate se vor transforma în zone luminoase, iar zonele deschise din imaginea inițială vor deveni zone întunecate.

- Să se calculeze complementul unei imagini.

```
I = imread('eh2.tiff');
I_complement = imcomplement(I);
figure(1), imshow(I)
figure(2), imshow(I_complement)
```

Imaginea inițială [12]

Complementul imaginii inițiale

12.2.3 Histograma unei imagini și operația de egalizare de histogramă

Histograma unei imagini redă distribuția nivelurilor de gri din imagine. Este funcția care asociază fiecărui nivel de gri frecvența de apariție în imagine. Dacă se consideră o imagine inițială cu o rezoluție de 8 biți/pixel, pe axa abscisei graficului histogramei se regăsesc cele 256 niveluri de gri posibile, iar pe axa ordonatei sunt redate frecvențele de apariție ale fiecărui nivel de gri de pe axa abscisei. Practic, în urma realizării histogramei se obține un vector `h` cu 256 de poziții (`h(1)` reprezintă numărul de pixeli care au valoarea intensității egală cu 0, `h(2)` reprezintă numărul de pixeli care au valoarea intensității egală cu 1, ..., `h(256)` reprezintă numărul de pixeli care au valoarea intensității egală cu 255). Suma elementelor din `h` reprezintă numărul de pixeli din imagine.

În MATLAB, histograma unei imagini grayscale `I` se calculează folosind funcția `imhist(I)`.

Sintaxă: `h = imhist(I_grayscale)`, întoarce în vectorul `h` numărul de apariții în imaginea `I_grayscale` al fiecărui nivel de gri.

În continuare sunt prezentate histogramele câtorva imagini medicale.

```
I1 = imread('spine.bmp'); h1 = imhist(I1);
I2 = imread('radiog_falange.jpg'); h2 = imhist(rgb2gray(I2));
I3 = imread('pulmonar2.jpg'); h3 = imhist(rgb2gray(I3));
figure(1),plot(h1,'k','LineWidth',2);
figure(2),plot(h2,'k','LineWidth',2);
figure(3),plot(h3,'k','LineWidth',2);
```

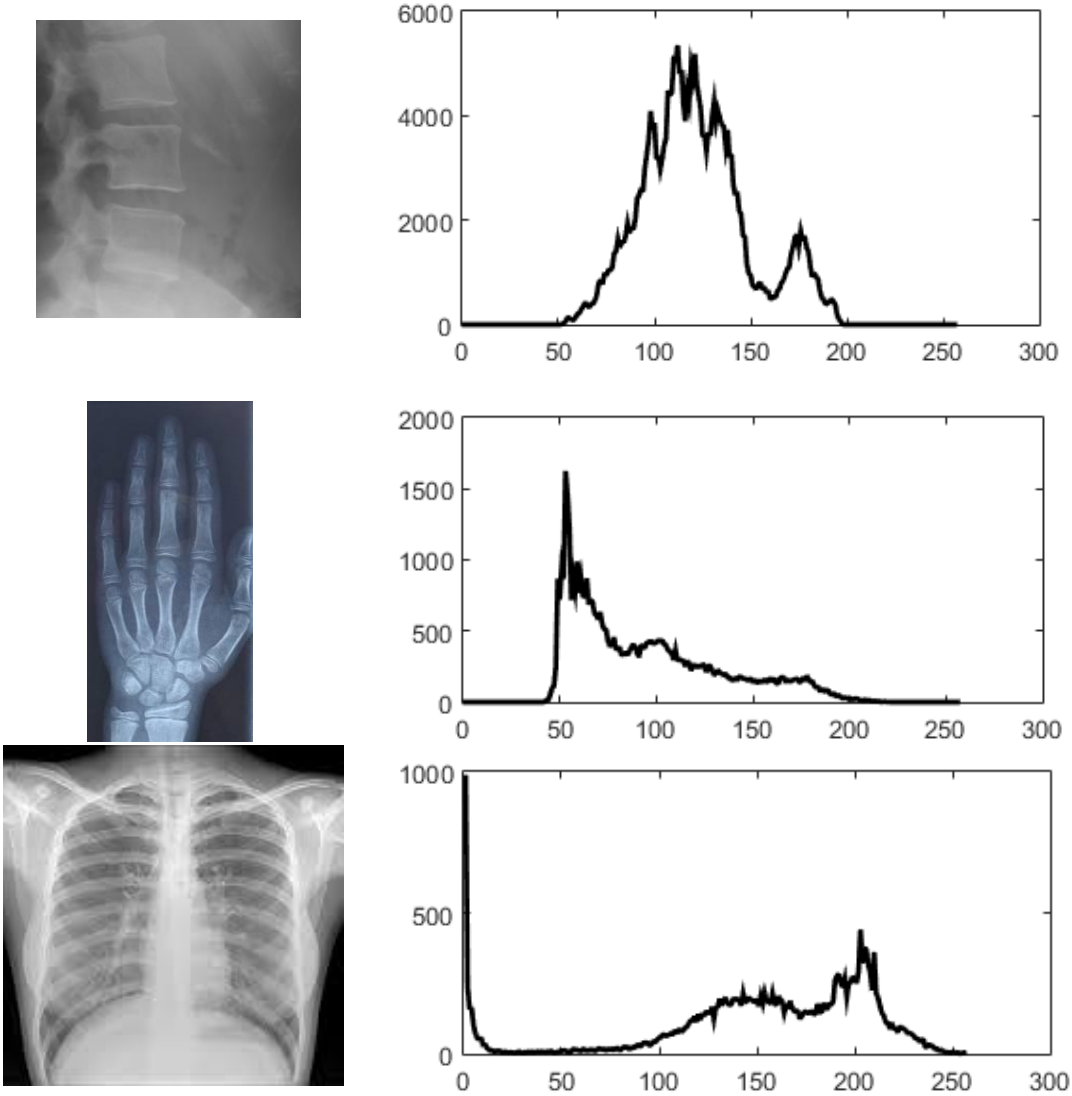


Figura 12.6. Exemple de histograme pentru diferite tipuri de radiografii.
În coloana din stânga sunt imaginile inițiale; în coloana din dreapta sunt graficele histogramelor asociate imaginilor din prima coloană

Observație: dacă se folosește funcția `imhist` fără parametru de ieșire se afișează direct histograma imaginii.

Egalizarea de histogramă

Dacă imaginea inițială este preponderent întunecată (sau luminoasă), atunci histograma asociată va avea o distribuție neuniformă, valorile concentrându-se în partea stângă a intervalului (în cazul imaginilor întunecate), sau în partea dreaptă a intervalului (în cazul imaginilor luminoase). Pe de altă parte, în practică, există și posibilitatea unei game restrânse a nivelurilor de gri. Toate aceste cazuri conduc la imagini cu un contrast redus și, implicit, la o calitate slabă a imaginii inițiale.

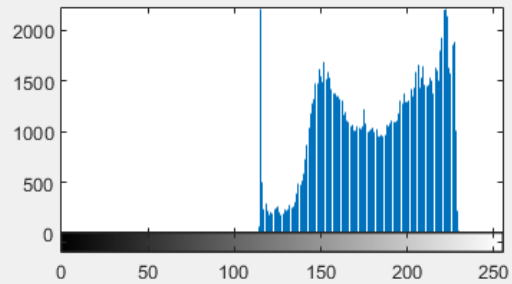
De aceea, în astfel de situații, este utilă operația de egalizare a histogramei. În MATLAB, egalizarea de histogramă se realizează cu funcția `histeq`.

Sintaxă: `J = histeq(I)`

- Exemplificare a egalizării de histogramă.

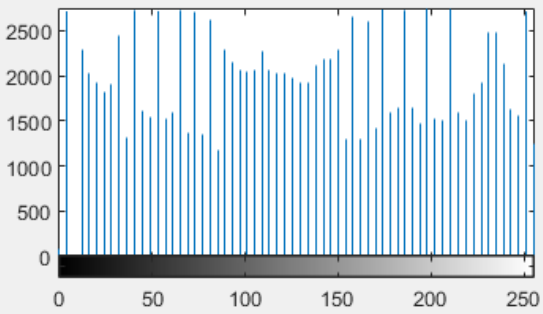
```
I1 = imread('low_contrast_pulmonar.tif');
I1_gray = rgb2gray(I1);
J1 = histeq(I1_gray);
% afisarea imaginii initiale
figure(1), imshow(I1_gray)
% afisarea histogramei initiale
figure(2), imhist(I1_gray)
% imaginea cu histograma egalizata
figure(3), imshow(J1)
% histograma egalizata
figure(4), imhist(J1)
```

Imagine inițială



Histograma imaginii inițiale

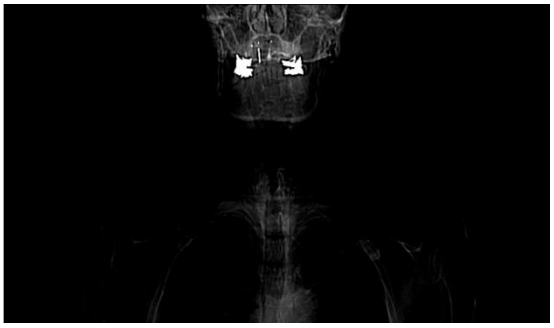
Imagine după egalizarea histogramei



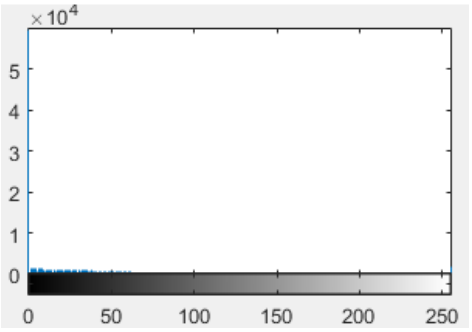
Histograma egalizată

Uneori, în practică, această uniformizare forțată a ponderilor nivelurilor de gri nu conduce neapărat la rezultate mai bune, întrucât prin acordarea de probabilități egale de apariție a nivelurilor de gri se amplifică în mod nedorit zgomotul din imagine și se minimizează și mai mult contrastul unor regiuni de interes subtile din imagine raportate la țesuturile din imediata vecinătate.

Fiind o tehnică de procesare globală, nu există niciun control în procesul de distribuire a nivelurilor de gri din imagine corespunzătoare regiunilor de interes. Se poate observa că atunci când imaginea inițială este extrem de dezechilibrată (în cazul de mai jos, imaginea este foarte întunecată), operația de egalizare de histogramă accentuează fundalul în detrimentul obiectelor de interes.

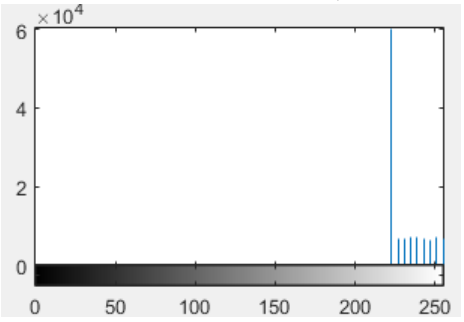


Imagine inițială



Histograma imaginii inițiale

Imagine după egalizarea histogramei



Histograma egalizată

Figura 12.7. Exemplu de folosire ineficientă a egalizării de histogramă

Pentru corectarea efectului uneori prea abrupt al egalizării de histogramă, se utilizează operația de **egalizare de histogramă adaptivă**. Această tehnică adaptivă acționează independent asupra diverselor regiuni din imagine, așa cum se poate observa în *Figura 12.7*. În mod ideal, o histogramă este generată într-o fereastră centrată pe fiecare pixel din imaginea originală, egalizată și utilizată, ulterior, pentru calcularea unei noi valori pentru pixelul respectiv.

În MATLAB, egalizarea adaptivă de histogramă pentru o imagine grayscale `I`, se calculează folosind funcția `adapthisteq`.

Sintaxă: `J = adapthisteq(I)`

- Să se aplice operatorul de egalizare de histogramă adaptivă.

```
I1 = imread('low_contrast_pulmonar.tif');
I2 = imread('mdb016.jpg');
J1 = histeq(rgb2gray(I1));
J1_adapt = adapthisteq(rgb2gray(I1));
J2 = histeq(I2);
J2_adapt = adapthisteq(I2);
figure(1),imshow(I1,[])
figure(2),imshow(J1,[])
figure(3),imshow(J1_adapt,[])
figure(4),imshow(I2,[])
figure(5),imshow(J2,[])
figure(6),imshow(J2_adapt,[])
```

Mai jos, sunt redate rezultatele obținute în urma rulării codului de mai sus.

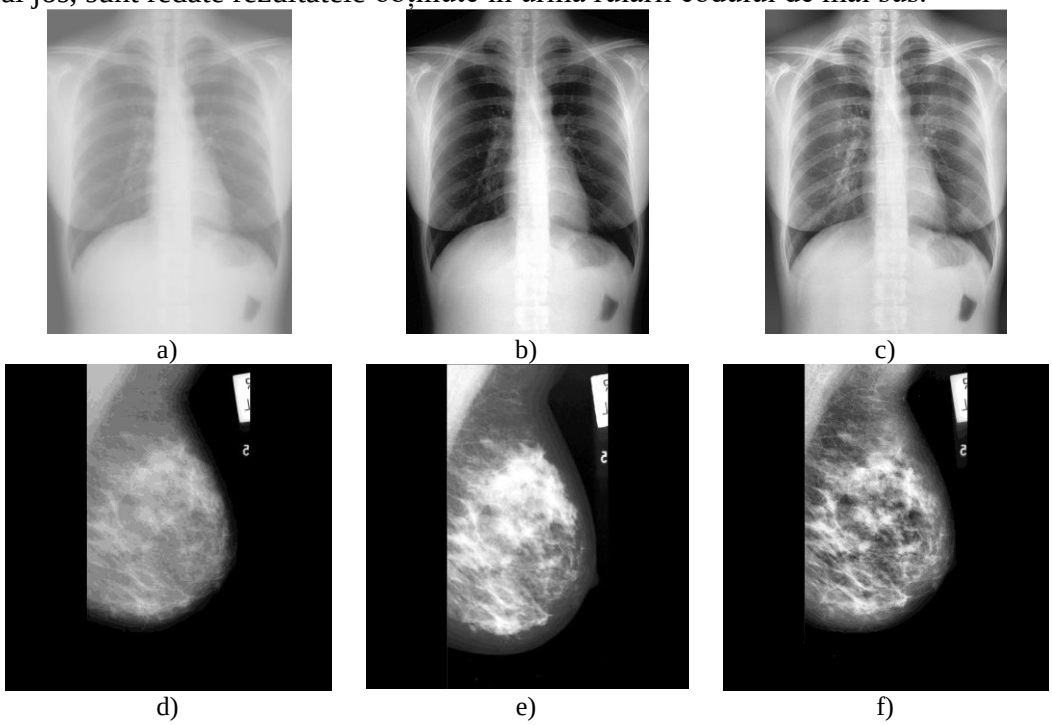


Figura 12.8. Comparație între egalizarea de histogramă neadaptivă și egalizarea de histogramă adaptive; a) și d) imagini originale [7]; b) și e) rezultate obținute în urma egalizării neadaptive de histogramă; c) și f) rezultate obținute în urma egalizării adaptive de histogramă

După cum s-a putut observa în cazurile prezentate anterior, radiografiile au un interval dinamic global mare și variații foarte mici ale nivelurilor de gri pentru componentele din imagine. Acest lucru face ca metoda de egalizare de histogramă adaptivă să fie o metodă eficientă pentru aceste cazuri. Însă, și această metodă prezintă câteva limitări: 1) este o metodă care implică o putere mare de calcul și 2) metoda adaptivă presupune alegerea unei ferestre variabile locale pentru calculul histogramei, care, implicit, diferă de la caz la caz, în funcție de aria obiectului de interes investigat.

12.2.4 Operația de accentuare a detaliilor

Claritatea detaliilor presupune contrastul între culori diferite. O tranziție rapidă de la alb la negru (respectiv, de la o culoare la alta) conduce la o imagine clară, cu detalii ascuțite (sharp). Invers, o tranziție graduală între alb și negru prin intermediul unor nuanțe de gri intermediare implică detalii neclare și o imagine globală încețoșată (blurată). Îmbunătățirea clarității detaliilor crește contrastul de-a lungul conturilor, a regiunilor de îmbinare cu culori, nuanțe diferite.
Funcția asociată operației de creștere a acuității detaliilor imaginii în MATLAB este `imsharpen`.

Sintaxă:

```
J = imsharpen(I, 'Radius',Value, 'Amount',Value, 'Threshold',Value)
```

Explicații:

- `I` este imaginea originală, cu niveluri de gri sau color

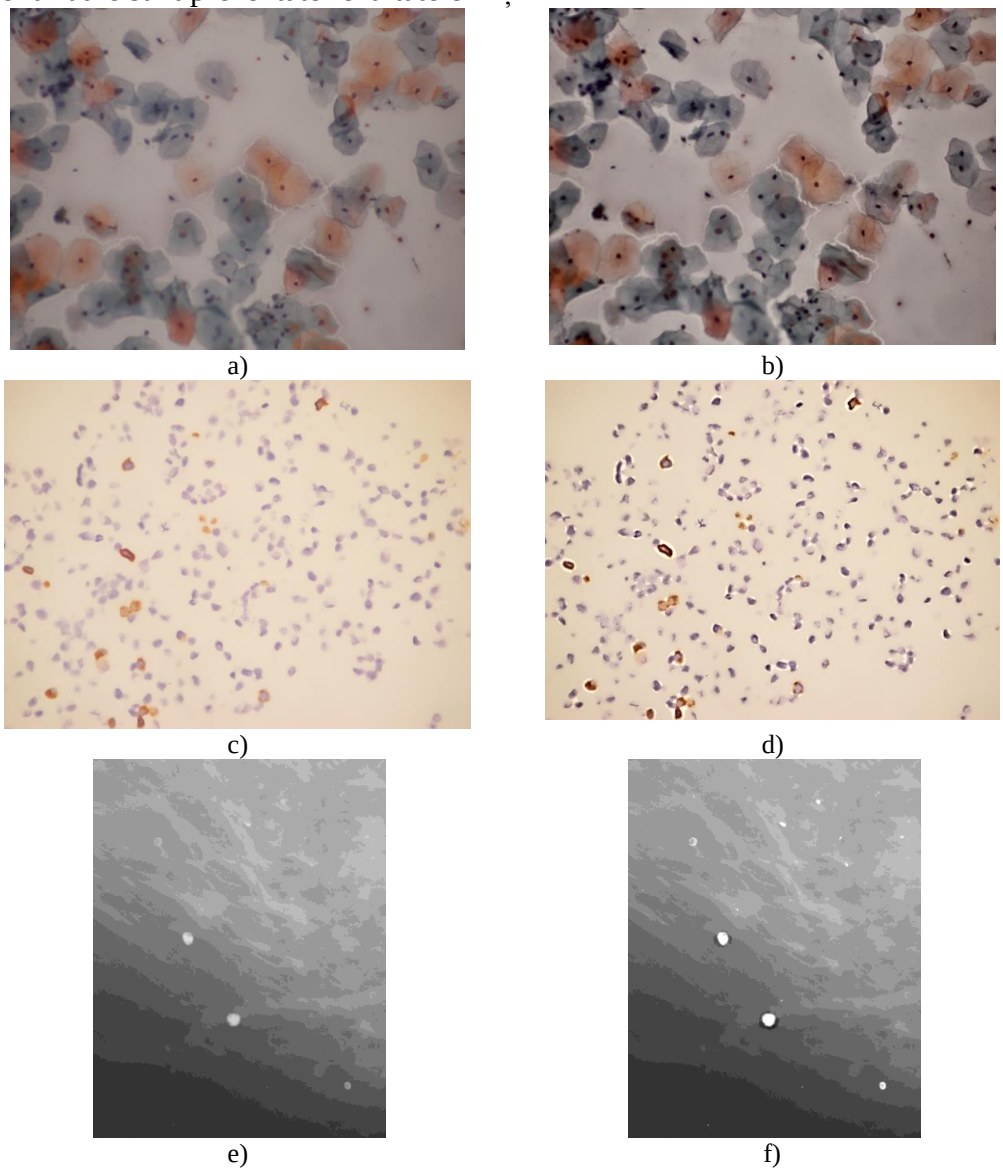
Următorii parametri sunt perechi de tipul `'Nume','Valoare'`

- Perechea `'Radius'`, `Value`: deviația standard a nucleului Gaussian a filtrului trece jos aplicat imaginii; valoarea predefinită este **1**.
- Perechea `'Amount'`, `Value`: intensitatea efectului de sharpening; valorile uzuale au loc în intervalul **[0, 2]**, efectul fiind cu atât mai puternic pentru o valoare mai mare.
- Perechea `'Threshold'`, `Value`: valoarea minimă de contrast pentru ca pixelul respectiv să fie considerat pixel contur (muchie), intervalul este **[0, 1]**; valorile mari permit numai muchiilor principale din imagine sa fie accentuate; implicit, cu cât pragul este redus, sunt accentuate și muchiile mai fine din imagine.

- Câteva exemple de accentuare a detaliilor folosind funcția `imsharpen`

```
I1 = imread('celule.tiff');
I2 = imread('celule2.tiff');
I3 = imread('cropl.tiff');
% apelare pentru I1
J1 = imsharpen(I1, 'Radius' ,20, 'Amount', 1.5, 'Threshold', 0.1);
% apelare pentru I2
J2 = imsharpen(I2,'Radius', 8, 'Amount', 2, 'Threshold', 0.15);
 % apelare pentru I3
J3 = imsharpen(I3, 'Radius', 10, 'Amount', 2 ,'Threshold', 0.15);
% afisare imagini rezultate
figure(1), imshow(J1)
figure(2), imshow(J2)
figure(3), imshow(J3)
```

În continuare sunt prezentate rezultatele obținute în urma rulării codului anterior.



a) b) c) d) e) f)

Figura 12.9. Exemple de utilizare a funcției `imsharpen`
a), c), e) imagini originale; b), d), f) imaginile rezultate

Se poate observa diferența de contrast dintre imaginile inițiale (prima coloană) și rezultatele obținute în urma îmbunătățirii contrastului cu funcția `imsharpen`. Efectul palid, cețos al detaliilor este corectat în imaginile rezultat. Se poate observa, de asemenea, un efect mai puternic asupra imaginilor color, nuanțele diferite fiind mai ușor de observat. Pentru ultimul caz, corespunzător imaginii cu microcalcificații, este însă evidentă accentuarea și vizibilitatea microcalcificațiilor mici din colțul dreapta – sus al imaginii.

12.3 Segmentarea imaginilor

În continuare se vor prezenta câteva dintre funcțiile MATLAB de bază ce se pot utiliza pentru segmentarea imaginilor în spațiul de culoare.

12.3.1 Operația de prăguire (Thresholding)

Operația de prăguire (echivalentă operației de binarizare din secțiunea anterioară) constă în aplicarea unei valori de prag (**`T`**) față de care valorile de gri din imagine vor lua două valori {0, 1}: pentru valorile de gri mai mari decât pragul `T` nivelurile de gri devin **1**, iar pentru valorile mai mici decât pragul `T` nivelurile de gri vor deveni **0**.

$$g(x,y)=\begin{cases}1, daca\ f(x,y)\geq T\\ 0, daca\ f(x,y)<T\end{cases}$$

Pixelii cu valoarea **1** corespund regiunii de interes, pe când pixelii cu valoarea **0** sunt atribuiți zonei de fundal. Când pragul `T` este o constantă, atunci metoda de prăguire este globală. În MATLAB, pentru aplicarea unui prag dat de utilizator, există funcția `im2bw`. Aceasta convertește imaginea cu niveluri de gri sau color **`I`** într-o imagine binară **`BW`**.

Sintaxă: `BW = im2bw(I, LEVEL)`

Pragul `LEVEL` este un scalar în intervalul [0, 1].

- Să se aplice operatorul de prăguire folosind pragul `LEVEL` = 0.6.

```
I = imread ('mamo_init.tiff');
J = im2bw(I, 0.6);
figure(1), imshow(J)
```

a) b)

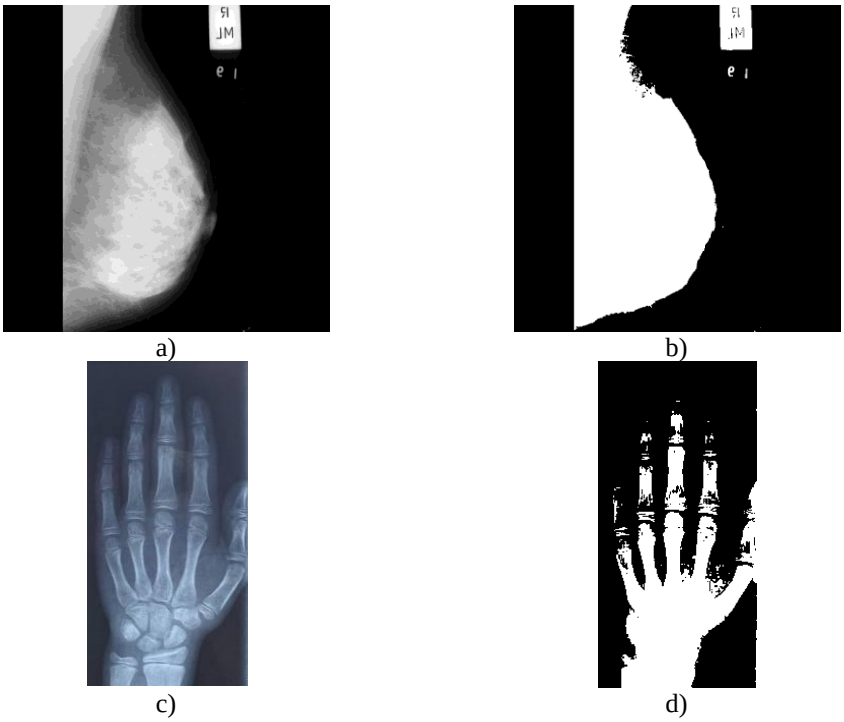
Figura 12.10. Segmentare cu prăguire.
a) imaginea originală [7] b) imaginea segmentată cu pragul T = 0.6

Dacă se dorește ca pragul să fie determinat automat, și nu introdus de utilizator, se poate folosi funcția `graythresh` bazată pe *metoda Otsu*; aceasta alege valoarea de prag ce maximizează varianța între două clase. Pragul va fi ales astfel încât cele două grupuri să fie cât mai "strânse", minimizând astfel suprapunerea. O măsură a omogenității grupului este varianța. Un grup cu omogenitate mare are o varianță mică, un grup cu omogenitate mică are varianță mare.

- Să se aplice operatorul de prăguire Otsu.

```
I1 = imread ('mamo_init.tiff');
I2 = imread ('radiog_falange.jpg');
% conversie imaginii color în imagini cu niveluri de gri
prag1 = graythresh(I1);
prag2 = graythresh(I2);
% aplicare operator praguire
J1 = im2bw(I1, prag1);
J2 = im2bw(I2, prag2);
figure(1), imshow(I1)
figure(2), imshow(J1)
figure(3), imshow(I2)
figure(4), imshow(J2)
```

În urma aplicării metodei Otsu de mai sus, pentru imaginea `I1` s-a obținut pragul `prag1` = 0.3569, iar pentru imaginea `I2` s-a obținut pragul `prag2` = 0.4667.



a) b) c) d)

Figura 12.11. Segmentare cu metoda Otsu
a)[7] și c) imagini originale; b) și d) imagini obținute în urma segmentării

12.3.2 Operatorul multi – prag (Multithresh)

MATLAB-ul dispune și de varianta de prăguire multi-prag, bazată pe *metoda Otsu*. Aceasta returnează un vector cu **N** valori de prag care pot fi utilizate ca argument de intrare al funcției `imquantize` pentru conversia imaginii de intrare **I** într-o imagine cu **N+1** praguri discrete.

Sintaxă: `thresh = multithresh(I, N)`

- Să se aplice operatorul de prăguire `multithresh` unei imagini grayscale.

```
I = imread ('prosthesis.bmp');
% apelare operator multithresh pentru cazul unei imagini grayscale
N = 2;
thresh = multithresh (I, N);
J = imquantize(I, thresh);
figure(1), imshow(I)
figure(2), imshow(J/(N+1))
```

Imagine originală

Imagine obținută în urma segmentării

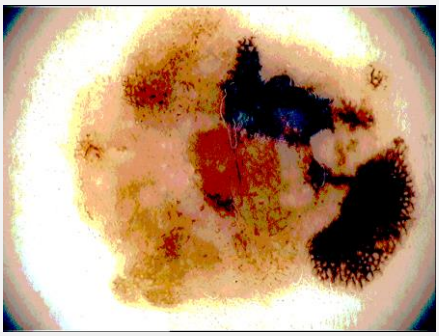
Pragurile determinate automat au fost **92** și **166**

- Să se aplice operatorul de prăguire `multithresh` unei imagini color.

```
I = imread (X.jpg');
N = 4;
% apelare operator multithresh pentru cazul unei imagini color
threshRGB = multithresh(I, N);
% se genereaza pragurile pentru fiecare plan al imaginii RGB
threshForPlanes = zeros(3, N);
 for i = 1 : 3
       threshForPlanes(i, :) = multithresh(I(:, :, i), N);
 end
 % proceseaza intreaga imagine cu setul de praguri determinati
 value = [0 threshRGB(2 : end) 255];
 quantRGB = imquantize(I, threshRGB, value);
% proceseaza fiecare plan RGB folosind vectorul de praguri din
fiecare plan.
% cuantizeaza fiecare plan RGB pe baza vectorului generat pentru
fiecare plan
```

```
 quantPlane = zeros(size(I));
for i = 1:3
       value = [0 threshForPlanes(i,2:end) 255];
       quantPlane(:,:,i) =
imquantize(I(:,:,i),threshForPlanes(i,:),value);
end
J = uint8(quantPlane);
imshow(J)
```

Imagine originală



Imagine obținută în urma segmentării

12.3.3 Metoda Fuzzy C-Means

Fuzzy C-Means este o metodă de clusterizare (grupare) a datelor în cadrul căreia fiecare punct aparține unei clase într-o anumită măsură care este dată de o pondere (grad de apartenență). Metoda arată posibilitatea de grupare a obiectelor care populează un spațiu multidimensional într-un număr specificat de clase diferite.

Inițial, se pornește cu o estimare a centrelor claselor și se atribuie în mod aleator fiecărui obiect un grad de apartenență la fiecare clasă. În final, prin actualizarea iterativă a centrelor claselor și a gradelor de apartenență, rezultă o distribuire a pixelilor din imagine într-un număr de clase.

Metoda *Fuzzy C-Means* este implementată în MATLAB prin intermediul funcției `fcm`.

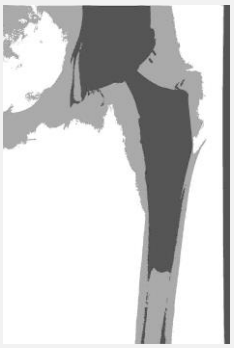
Sintaxă: `[CENTER, U, OBJ_FCN] = fcm(DATA, Nr_clase)`

- `DATA` reprezintă setul de date ce se dorește a fi împărțit în *clustere*
- `Nr_clase` reprezintă numărul de clase în care se dorește să se facă împărțirea
- `CENTER` conține coordonatele centrelor tuturor claselor
- `U` conține gradele de apartenență ale datelor la toate clasele
- `OBJ_FCN` reprezintă valoarea *funcția obiectiv* ce trebuie minimizată

În continuare, este prezentat un exemplu de segmentare a unor radiografii cu ajutorul metodei *Fuzzy C-Means*.

```
a = double(imread('proteza_sold2.jpg'));
c = 3; % setam numarul claselor c = 3
[center, u, obj_fcn] = fcm(a(:), c);
[out1, out2] = max(u);
out_seg = reshape(out2, size(a,1), size(a,2));
figure(1), image(out_seg), colormap(((1:c)' * ones(1,3))/ c)
```

Imagine originală



Imagine segmentată în 3 clase cu fcm

12.3.4 Metoda de segmentare bazată pe creșterea regiunilor (Region Growing)

Ideea de bază a metodei de segmentare bazată pe creșterea regiunilor este următoarea: se pornește, inițial, de la un pixel sau un grup de pixeli, denumiți pixeli semințe (*seeds*) și se examinează, pe rând, toți pixelii vecini ai acestora. Dacă unul dintre aceștia îndeplinește un anumit criteriu, dat de relația de vecinătate, atunci pixelul respectiv este adăugat grupului de pixeli deja format.

Acest proces este continuat, pentru toți pixelii noi adăugați, până când nu se mai respectă criteriul relației de vecinătate. Algoritmul depinde de masca pixelilor semințe inițiale, însă cel mai important este definirea criteriului de omogenitate pe baza căruia se adaugă noii vecini. În plus, prin natura sa, acest algoritm este unul recursiv ceea ce necesită resurse din partea sistemului de calcul.

MATLAB-ul nu conține o funcție predefinită de segmentare bazată pe algoritmul de creștere a regiunilor. Din acest motiv, vom reda algoritmul propus de Gonzalez și Woods [1].

```
function [g, NR, SI, TI] = regiongrow(f, S, T)
%REGIONGROW - aplicarea metodei de segmentare bazata pe cresterea
regiunilor.
%   [G, NR, SI, TI] = REGIONGROW(F, SR, T).  S poate fi un vector de
aceeasi dimensiune cu F) cu 1 in coordonatele fiecarei punct samanta
si 0 in rest; S  poate fi o singura valoare pentru obinterea
pixelului - samanta. Similar, T poate fi un vector (de aceeasi
dimensiune cu F) continand o valoare de prag pentru fiecare pixel din
F. T poate fi, de asemenea, un prag - o valoare globala pentru
intreaga imagine.
% La iesire, G este rezultatul metodei de cresterea regiunilor, cu
fiecare regiune etichetata cu un numar intreg, distinct. Parametrul
NR returneaza numarul de regiuni distincte determinate, SI este
imaginea finala cu pixeli-samanta utilizata in algoritm, iar
parametrul TI este imaginea care contine pixelii din F care satisfac
pragul de test.
%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.4 $  $Date: 2003/10/26 22:35:37 $
f = double(f);
% Daca S este un scalar, intoarce masca de pixeli - samanta
if numel(S) == 1
   SI = f == S;
   S1 = S;
else
   % S este un vector
   SI = bwmorph(S, 'shrink', Inf);
   J = find(SI);
   S1 = f(J); % Vectorul de pixeli - samanta
end

TI = false(size(f));
for K = 1:length(S1)
   seedvalue = S1(K);
   S = abs(f - seedvalue) <= T;
   TI = TI | S;
end
% Utilizarea functiei imreconstruct cu masca SI  pentru obtinerea
regiunilor corespunzatoare fiecarui pixel samanta S
[g, NR] = bwlabel(imreconstruct(SI, TI));
```

- În continuare se va testa funcția de mai sus pentru trei imagini medicale.

```
a = imread('proteza_sold2.jpg');
[g1, NR, SI, TI] = regiongrow(a,250,20);
b = imread('original.tiff');
[g2, NR, SI, TI] = regiongrow(b,190,10);
c = imread('radiog_falange.jpg');
c = rgb2gray(c);
[g3, NR, SI, TI] = regiongrow(c,120,40);
```

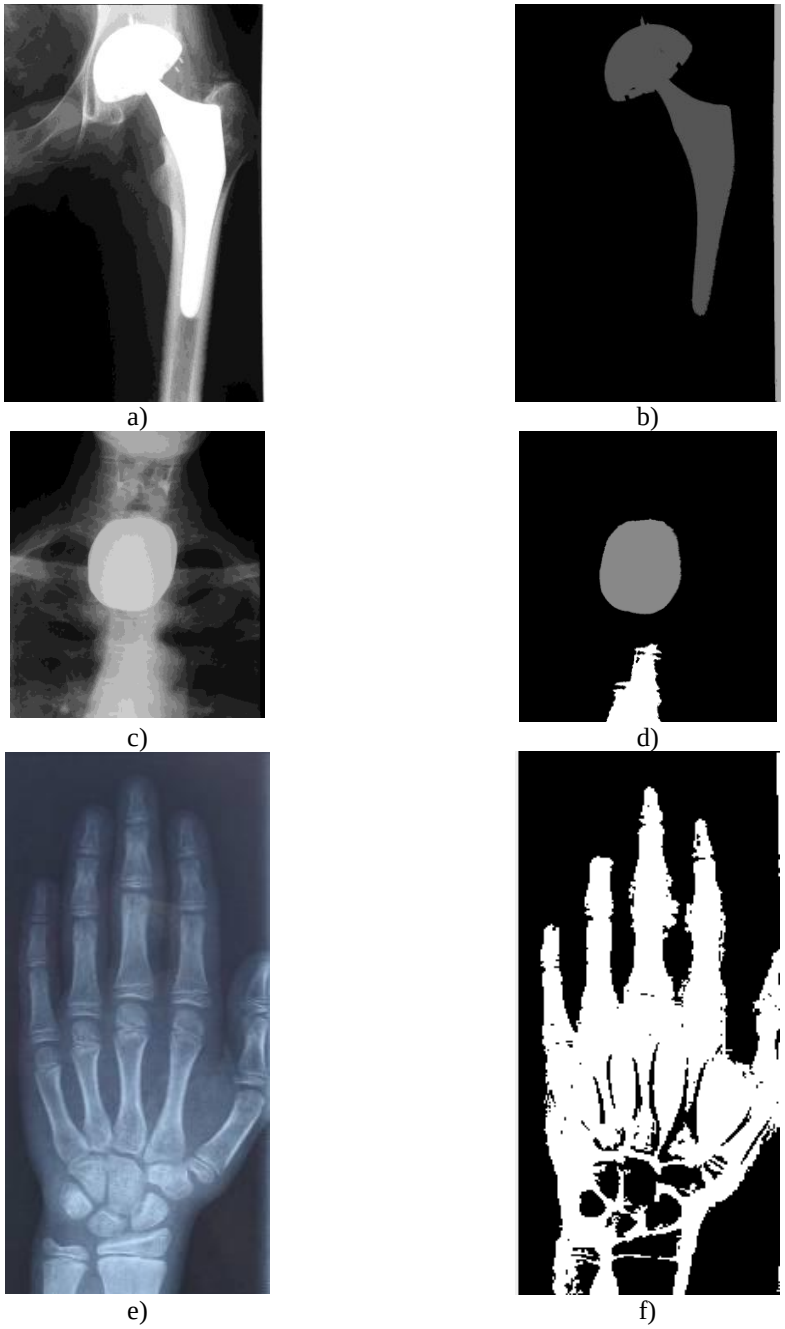


Figura 12.12. Exemple de segmentare cu ajutorul metodei de crestere a regiunilor.
a), c), e) Imaginile inițiale; b), d), f) Rezultatele obtinute în urma procesului de segmentare pentru valorile **S = 250, T = 20** (pentru *a*) , **S = 190, T = 10** (pentru *c*), **S = 120, T = 40** (pentru *e*),

12.3.5 Funcții suport utilizate în metodele de segmentare

Funcția `bwlabel`

Funcția `bwlabel` etichetează componentele conexe ale unei imagini binare.

Sintaxă: `[out, num] = bwlabel(I_binar, n)`

Funcția returnează matricea rezultat `out`, de aceeași dimensiune cu matricea `I_binar`. Matricea `out` conține etichetele pentru componentele conexe din imaginea binară `I_binar`.

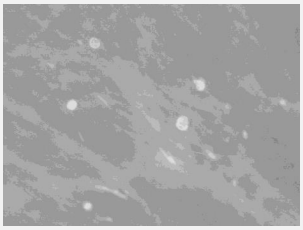
Variabila `n` poate avea valorile 4 sau 8, în funcție de relația de conectivitate a obiectelor conexe date de vecinătatea *V4*, respectiv *V8*.

Matricea rezultat `out` conține valori pozitive. Pixelii etichetați cu valoarea **0** sunt pixeli de fundal; cei cu **1** corespund unui obiect conex, pixelii etichetați cu cifra 2 sunt asociați unui al doilea obiect, și așa mai departe.

Variabila `num` reprezintă numărul de obiecte conexe, distincte din imagine.

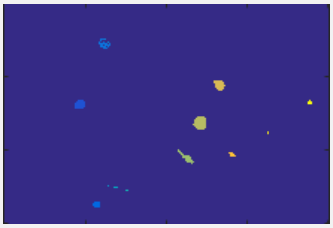
- Să se eticheteze regiunile dintr-o imagine binară folosind `bwlabel`

```
a = imread(cropa.jpg');
% se transforma imaginea grayscale intr-o imagine binara
binar = im2bw(a, 0.75);
% se eticheteaza imaginea binara
out = bwlabel(binar, 4);
figure(1), imshow(binar)
figure(2), imagesc(out)
```



Imagine originala

Imagine binară



Imagine etichetată

O variantă a funcției `bwlabel` este `bwboundaries`, a cărei sintaxă este:

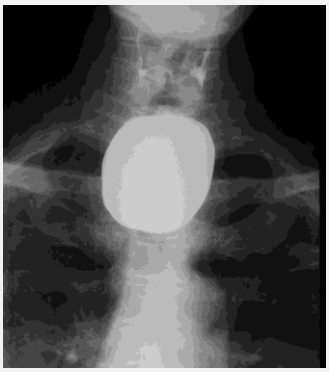
Sintaxă: `[B, L] = bwboundaries(I_binar, n)`

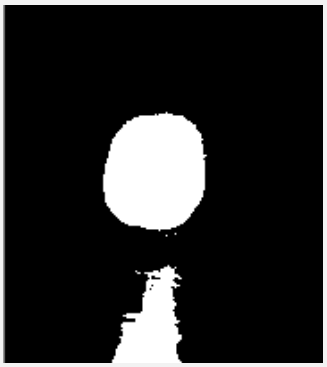
`B` este masca marginilor obiectelor conexe determinate din imaginea de intrare `I_binar`. În rest, parametrii de intrare și cei de ieșire sunt aceeași, ca și în cazul funcției `bwlabel`.

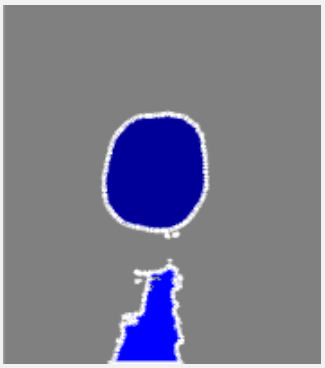
Observație: funcția `bwboundaries` poate avea opțional și parametrul de intrare `'noholes'`, care caută doar marginile exterioare ale obiectelor.

- Să se eticheteze regiunile dintr-o imagine binară folosind `bwboundaries`

```
a = imread('original.tiff');
% se transforma imaginea grayscale intr-o imagine binara
binar = im2bw(a, 0.7);
[B, L] = bwboundaries(binar,'noholes');
figure(1), imshow(binar)
figure(2)
imshow (label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
     boundary = B{k};
     plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
end
```


Imagine originala


Imagine binară


Imagine etichetată

Funcția `regionprops`

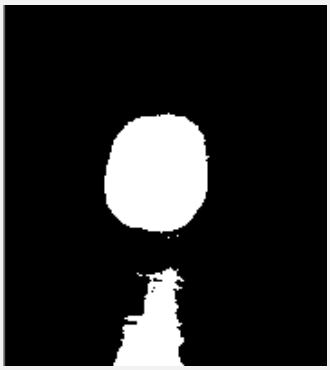
Funcția `regionprops` descrie regiunile (obiectele) dintr-o imagine binară.
Sintaxă: `stats = regionprops(I_binar, proprietate)`
Variabila `stats` returnată este un vector structură de aceeași dimensiune cu numărul obiectelor determinate din imaginea binară de intrare `I_binar`. Câmpurile vectorului structură `stats` desemnează diferite proprietăți ale fiecărei regiuni, specificate prin parametrul `proprietate`.

Parametrul `proprietate` indică printr-o variabilă de tip *string* asociată, proprietatea (sau șirul de proprietăți) care se dorește a fi returnată. Aceste proprietăți pot fi parametri de formă ('Area', 'EulerNumber', 'Orientation', 'BoundingBox' 'Extent', 'Perimeter', 'Centroid', 'Extrema', 'PixelIdxList', 'ConvexArea', 'FilledArea', 'PixelList', 'ConvexHull', 'FilledImage', 'Solidity', 'ConvexImage', 'Image', 'SubarrayIdx', 'Eccentricity', 'MajorAxisLength', 'EquivDiameter', 'MinorAxisLength') sau parametri de valoare ai pixelilor ('MaxIntensity', 'MinIntensity', 'WeightedCentroid', 'MeanIntensity', 'PixelValues')

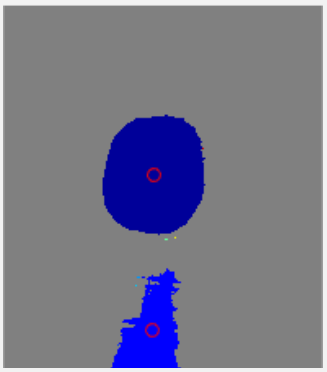
- Să se marcheze într-o imagine binară centrele regiunilor care au o suprafață mai mare de `N` pixeli (`N` este parametru de intrare).

```
a = imread('original.tiff');
% se transforma imaginea grayscale intr-o imagine binara
binar = im2bw(a, 0.7);
% se eticheteaza imaginea binara
[B, L] = bwboundaries(binar,'noholes');
figure(1),imshow(label2rgb(L, @jet, [.5 .5 .5]))
% pentru toate regiunile se extrag parametrii: Area si Centroid
stats = regionprops(L,'Area','Centroid');
N = 100;
for k = 1:length(B)
      area = stats(k).Area;
      centroid = stats(k).Centroid;
      if(area > N)
          hold on
          plot(centroid(1),centroid(2),'ro');
      end
 end
```

Imaginea original

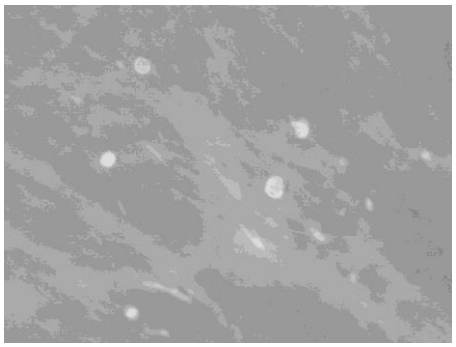


Imaginea binară

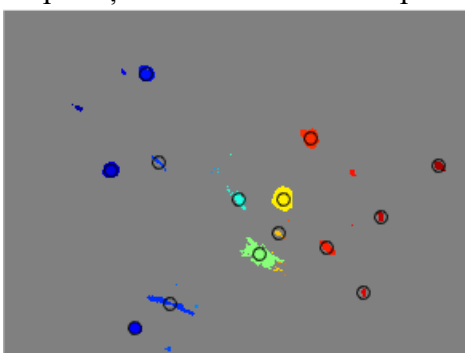


S-a marcat cu cerc roșu centrul regiunilor cu suprafața mai mare de 100 pixeli

În continuare sunt rezultatele obținute pentru o altă imagine medicală. În acest caz s-au marcat cu negru centrele regiunilor care au suprafața mai mare de 100 de pixeli.



a) Imagine originală



b) Imaginea finală

Figura 12.13. Identificarea regiunilor cu suprafața mai mare de 100 pixeli
Marcarea centrelor acestor regiuni

12.4 Operații morfologice

Cuvântul ***morfologie*** provine din latină unde ***morphos*** înseamnă formă, iar ***logos*** știință. Așadar, morfologia este o știință a formelor, adică o abordare a prelucrării imaginilor bazată pe noțiunea de formă. Din punct de vedere matematic, domeniul morfologiei presupune operarea cu mulțimi de puncte cu rolul de a caracteriza forma obiectelor de interes. În general, operațiile morfologice au doi factori: primul corespunde mulțimii de transformat, iar al doilea factor corespunde mulțimii cu ajutorul și în funcție de care va fi transformată prima mulțime.

În cazul abordării bazate pe morfologie matematică, ideea de bază este în a considera imaginea ca un ansamblu. Asupra acestui ansamblu se aplică transformări prin comparația cu mulțimi mai simple, numite elemente structurante. Scopul acestor transformări este de a obține forme mai simple rezultate din formele inițiale ale imaginii.

Elementul structurant constă într-o mulțime geometrică, arbitrară, cunoscută și impusă. Forma și dimensiunea elementului structurant determină proprietățile testate asupra formei obiectului pe care dorim să îl modificăm.
Elementul structurant este definit de trei elemente caracteristice:

- dimensiunea matricei: care specifică dimensiunea elementului structurant;
- șablonul format din elementele 1 sau 0: care redă forma elementului structurant;
- originea: care poate fi în interiorul elementului structurant sau în afara acestuia

Sintaxa elementului structurant este: `es = strel(shape, parameters)`
Parametrul de intrare `shape` este o variabilă de tip *string* care poate să fie predefinită de o formă geometrică ('pair', 'diamond', 'periodicline', 'disk', 'rectangle', 'line', 'square', 'octagon') a cărei dimensiune este dată de variabila `parameters` sau poate fi arbitrară și atunci `parameters` este redat printr-o vecinătate aleasă.

12.4.1 Operația de erodare

Erodarea morfologică a mulțimii **A** prin elementul structurant **B** este definită ca mulțimea punctelor în care se poate translata elementul structurant **B** considerat, astfel încât acesta să fie inclus în întregime în mulțimea punctelor elementului **A**.
Intuitiv, operația de erodare, după cum îi spune și numele, va avea ca efect micșorarea obiectului **A**. Această micșorare va fi în concordanță cu forma și dimensiunea elementului structurant **B** aplicat.

Sintaxă: `out = imerode(in, es)`

- Imaginea `in` este o imagine binară sau cu niveluri de gri;
- `es` este elementul structurant aplicat mulțimii `in`.

12.4.2 Operația de dilatare

Dilatarea morfologică a mulțimii **A** prin elementul structurant **B** se definește ca mulțimea punctelor în care se poate translata elementul structurant astfel încât acesta să aibă puncte comune cu mulțimea de prelucrat **A**.
Efectul operației de dilatare este opus efectului erodării și anume mărirea mulțimii obiectului de prelucrat **A** în concordanță cu forma și dimensiunea elementului structurant **B** aplicat. Intuitiv, opus erodării, operația de dilatare presupune bordarea mulțimii **A**, latura bordării fiind dată de elementul structurant ales.
Sintaxă: `out = imdilate (in, es)`

- Imaginea `in` este o imagine binară sau cu niveluri de gri;
- `es` este elementul structurant aplicat mulțimii `in`.

12.4.3 Operația de deschidere

Deschiderea morfologică a mulțimii **A** prin elementul structurant **B** se definește ca erodarea mulțimii cu elementul structurant respectiv, urmată de dilatarea cu elementul structurant simetrizat. Prin aplicarea unei operații de deschidere cu un element structurant *disc* centrat în origine, componentele conexe ale mulțimii **A** mai mici decât elementul structurant sunt înlăturate din imagine; convexitățile foarte accentuate ale contururilor sunt diminuate și punțile subțiri de legătură sunt îndepărtate .
Sintaxă: `out = imopen (in, es)`

- Imaginea `in` este o imagine binară sau cu niveluri de gri;
- `es` este elementul structurant aplicat mulțimii `in`

12.4.4 Operația de închidere

Închiderea morfologică a mulțimii **A** prin elementul structurant **B** se definește ca dilatarea mulțimii cu elementul structurant respectiv, urmată de erodarea cu elementul structurant simetrizat. Prin închiderea cu un element structurant *disc* centrat în origine, golurile incluse în obiecte, mai mici decât elementul structurant folosit sunt umplute, se umplu concavitățile puternice ale contururilor și obiectele foarte apropiate sunt fuzionate.
Sintaxă: `out = imclose (in, es)`

- Imaginea `in` este o imagine binară sau cu niveluri de gri;
- `es` este elementul structurant aplicat mulțimii `in`

- Exemplificarea funcțiilor morfologice menționate mai sus.

```
a = imread('dermatoscopie.jpg');
% conversia imaginii color în imagine cu niveluri de gri
gri = rgb2gray(a);
% calcul prag de binarizare cu metoda Otsu
prag = graythresh(gri)*255;
bw = gri < prag; % conversia imaginii grayscale în imagine alb-negru
s = strel('disk', 7); % definirea elementului structurant s
erodare = imerode(bw, s); % apelarea operatiei de erodare
figure(1), imshow(erodare), title('erodare')
dilatare = imdilate(bw, s); % apelarea operatiei de dilatare
figure(2), imshow(dilatare), title('dilatare')
deschidere = imopen(bw, s); % apelarea operatiei de deschidere
figure(3), imshow(deschidere), title('deschidere')
inchidere = imclose(bw, s); % apelarea operatiei de închidere
figure(4), imshow(inchidere), title('inchidere')
```

În *Figura 12.14* sunt prezentate rezultatele obținute cu ajutorul funcțiilor morfologice.

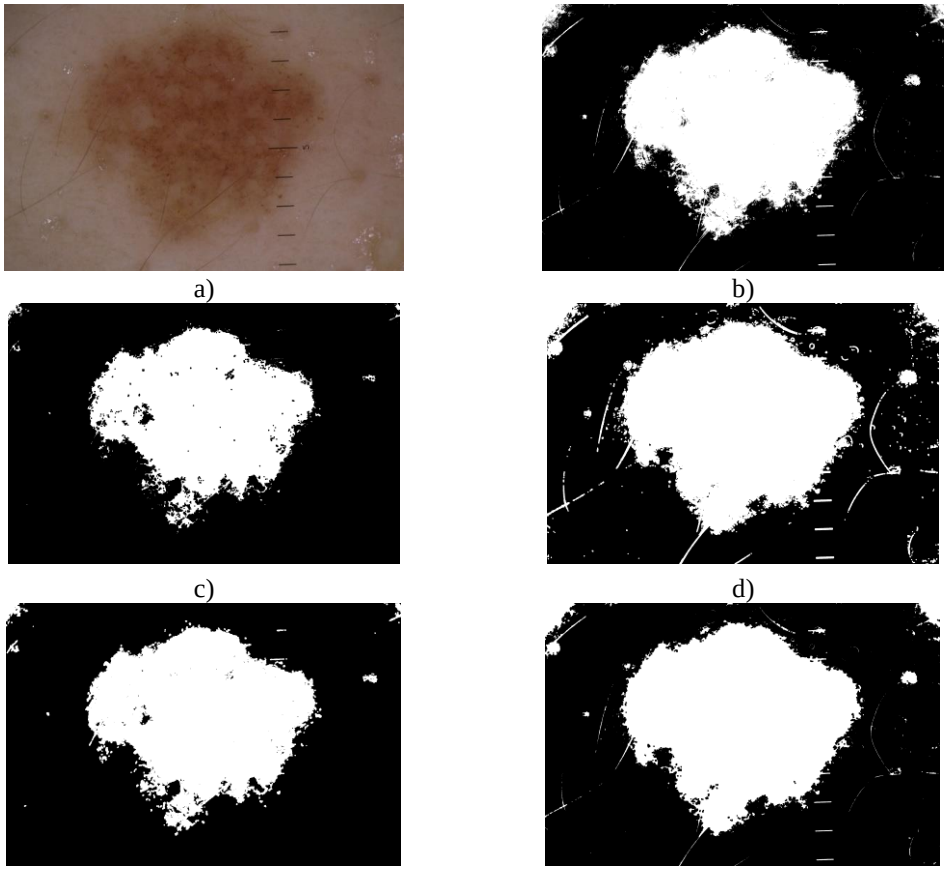


Figura 12.14. a) Imaginea inițială; Rezultatul obținut în urma operației de binarizare cu metoda Otsu; c) Rezultatul aplicării operației de erodare; d) Rezultatul aplicării operației de dilatare; e) Rezultatul aplicării operației de deschidere; f) Rezultatul aplicării operației de închidere

13. Publicarea codului MATLAB

Opțiunea *Publishing* creează un document ce include codul MATLAB, comentariile și rezultatele obținute (inclusiv figurile afișate). Pentru a împărți codul în secțiuni se inserează simbolurilor **%%**. Pentru a publica codul MATLAB se selectează opțiunea *Publish* din *Toolstrip* și apoi se apasă butonul pentru publicare  .

În mod implicit, formatul în care se publică codul este formatul *html*. Însă se poate opta și pentru alte formate și anume: *doc, pdf, ppt, LateX, xml.* Pentru a selecta formatul în care doriți să publicați, selectați *Publish* din *Toolstrip*, apoi *Publish/Edit Publishing Options/Output Settings* și selectați *Output File Format.*

Fie un obiect surprins la două momente de timp în două poziții diferite. Să se calculeze distanța de deplasare în pixeli între cele două poziții ale obiectului și să se marcheze centrul obictului la cele două momente de timp.
Implementarea în MATLAB este următoarea:

```
clc
clear all
close all

%% citire imagini la cele doua momente de timp
imag1 = imread('D:\carte Matlab\poze\imag1.jpg');
imag2 = imread('D:\carte Matlab\poze\imag2.jpg');

figure(1)
subplot(2,1,1), imshow(imag1), title('imagine la t1')
subplot(2,1,2), imshow(imag2), title('imagine la t2')

%% imagine grayscale
imag1_gray = double(rgb2gray(imag1));
imag2_gray = double(rgb2gray(imag2));

%% diferenta imaginilor in modul
imag_dif = abs(imag1_gray(:,:,1) - imag2_gray(:,:,1));
figure(2)
image(imag_dif), colormap(gray(256)),title('diferenta imaginilor in
modul')

%% histograma diferenta imagini
figure(3)
imhist(uint8(imag_dif)), title('histograma diferenta imagini')
%determinare prag de segmentare cu metoda Otsu
prag = graythresh(imag_dif)*255;
disp(['prag de segmentare = ', num2str(prag)])
```

```
%% binarizare imagine_diferenta
bw = imag_dif > prag;
figure(4)
imshow(bw), title('imaginea binara')
% inchidere imagine
se = strel('disk',10);
bw = imclose(bw, se);
figure(5)
imshow(bw), title('imaginea binara in urma inchiderii')
%% determinare centre obiecte
stats = regionprops(bw, 'MajorAxisLength','Centroid');
% determinarea celor mai mari 2 BoundingBox
[val,poz] = sort([stats.MajorAxisLength],'descend');
centru1 = stats(poz(1)).Centroid;
centru2 = stats(poz(2)).Centroid;

%% marcarea centrelor
figure(6)
subplot(2,1,1), imshow(imag1_gray/255)
hold on
plot(centru2(1), centru2(2),'*r')
title('marcarea centrului pentru imag1')
hold off
subplot(2,1,2), imshow(imag2_gray/255)
hold on
plot(centru1(1), centru1(2),'*r')
title('marcarea centrului pentru imag2')
hold off

%% calcul distanta in pixeli
distanta_pixeli = sqrt((centru1(1)-centru2(1))^2 +(centru1(2)-
centru2(2))^2)
```

În urma implementării codului și a folosirii opțiunii *Publish*, se obține un document cu următorul conținut:

```
clc
clear all
close all
```

citire imagini la cele doua momente de timp

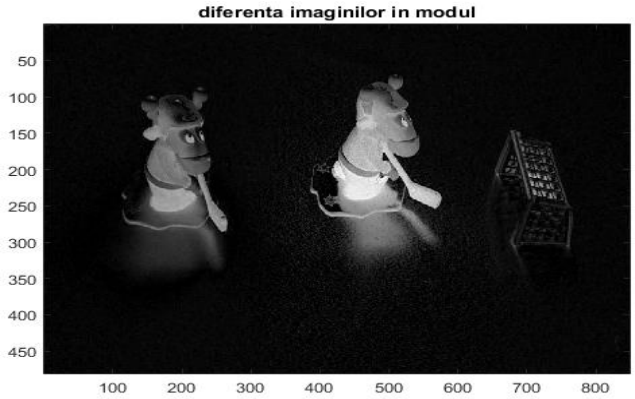
```
imag1 = imread('D:\carte Matlab\poze\imag1.jpg');
imag2 = imread('D:\carte Matlab\poze\imag2.jpg');

figure(1)
subplot(2,1,1), imshow(imag1), title('imagine la t1')
subplot(2,1,2), imshow(imag2), title('imagine la t2')
```

imagine grayscale

```
imag1_gray = double(rgb2gray(imag1));
imag2_gray = double(rgb2gray(imag2));
```

diferenta imaginilor in modul

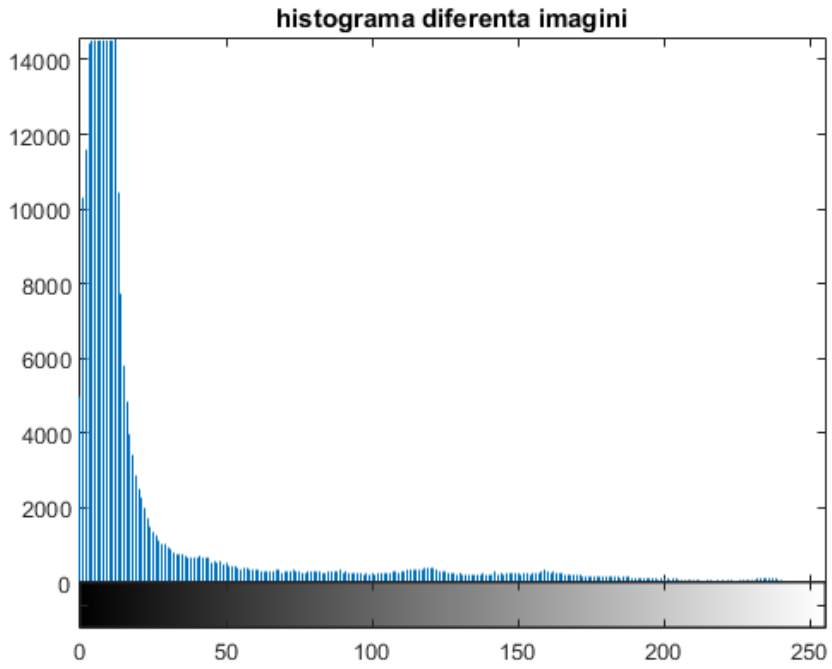
```
imag_dif = abs(imag1_gray(:,:,1) - imag2_gray(:,:,1));
figure(2)
image(imag_dif), colormap(gray(256)),title('diferenta imaginilor in modul')
```



histograma diferenta imagini

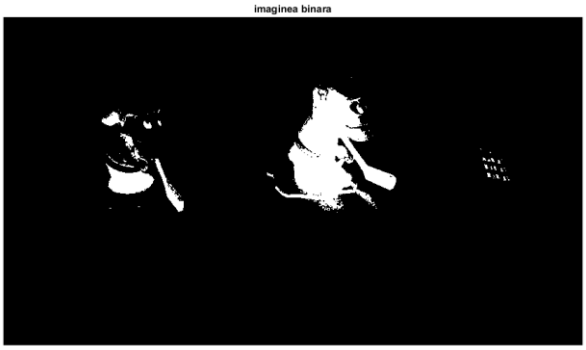
```
figure(3)
imhist(uint8(imag_dif)), title('histograma diferenta imagini')
%determinare prag de segmentare cu metoda Otsu
prag = graythresh(imag_dif)*255;
disp(['prag de segmentare = ', num2str(prag)])
```

```
prag de segmentare = 127
```



binarizare imagine_diferenta

```
bw = imag_dif > prag;
figure(4)
imshow(bw), title('imaginea binara')
% inchidere imagine
se = strel('disk',10);
bw = imclose(bw, se);
figure(5)
imshow(bw), title('imaginea binara in urma inchiderii')
```

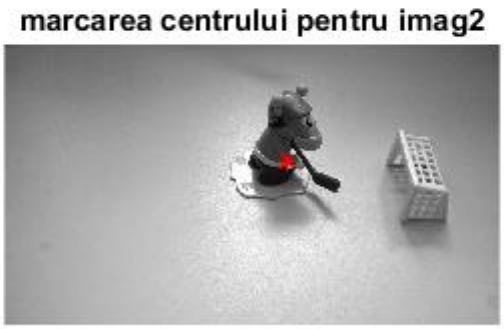


determinare centre obiecte

```
stats = regionprops(bw, 'MajorAxisLength','Centroid');

% determinarea celor mai mari 2 BoundingBox
[val,poz] = sort([stats.MajorAxisLength],'descend');
centru1 = stats(poz(1)).Centroid;
centru2 = stats(poz(2)).Centroid;
```

marcarea centrelor

```
figure(6)
subplot(2,1,1), imshow(imag1_gray/255)
hold on
plot(centru2(1), centru2(2),'*r')
title('marcarea centrului pentru imag1')
hold off
subplot(2,1,2), imshow(imag2_gray/255)
hold on
plot(centru1(1), centru1(2),'*r')
title('marcarea centrului pentru imag2')
hold off
```



calcul distanta in pixeli

```
distanta_pixeli = sqrt((centru1(1)-centru2(1))^2 +(centru1(2)-centru2(2))^2)
```

```
distanta_pixeli =

  288.6627
```

Published with MATLAB® R2015a

Bibliografie

[1] *MATLAB, Programming Fundamentals, R2015b,* The MathWorks, Inc

[2] *MATLAB, Creating Graphical User Interfaces, R2015b,* The MathWorks, Inc

[3] Rafael C. Gonzalez, Richard E. Woods, "*Digital Image Processing*", 2002 by Prentice-Hall, ISBN 0-201-18075-8

[4] Ioan P. Mihu, Cătălina Neghină, "*Prelucrarea Digitală a Semnalelor. Aplicații didactice în Matlab*", ISBN 978-606-12-0796-1

[5] Alina Elena Sultana, Sever Pașca, Șerban Oprișescu, "*Imagistică Medicală – îndrumar de laborator pentru uzul studentilor"*, ISBN 978-973-755-726-1

[6] Ioan P. Mihu, "*ANSI-C pentru microcontrollere*", 2012, ISBN 978-973-0-12166-7

Sursele imaginilor medicale care nu aparțin autorilor:

[7] http://peipa.essex.ac.uk/info/mias.html

[8] http://www.healthcare.siemens.com/computed-tomography/clinical-imaging-solutions/ct-neurology-engine

[9] http://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri

[10] http://www.wvuhradtech.com/nuclear%20medicine.htm

[11] http://www.babble.com/pregnancy/follow-up-ultrasound-today/

[12] http://radiopaedia.org/images/24562

Notă: Toate imaginile în afara celor medicale menționate în bibliografie, aparțin autorilor și nu pot fi folosite fără acordul acestora.