Cătălina Neghină

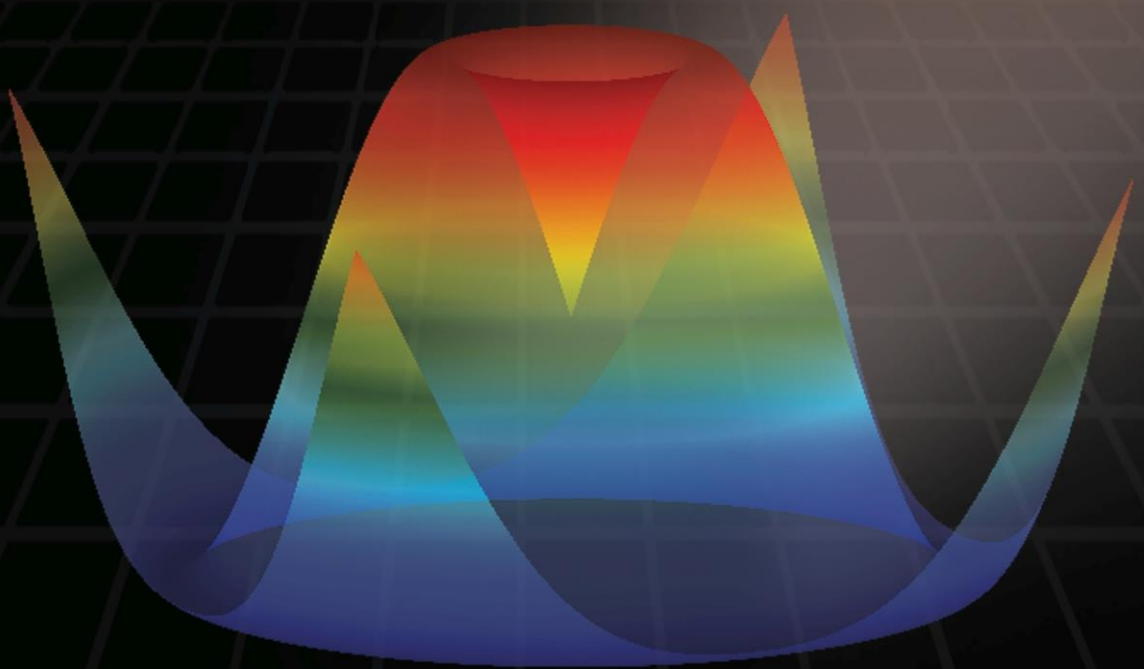
Mihai Neghină

MATLAB

Un limbaj pentru viitorii ingineri

$$f(x,y) = 1 + \sin\left(\sqrt{x^2 + y^2}\right)$$

```
x = -5 : 0.01 : 5;
y = -5 : 0.01 : 5;
[X, Y] = meshgrid(x, y);
Z = 1 + sin(sqrt(X.^2 + Y.^2));
figure(), surf(X,Y,Z,'EdgeColor','none')
```



Cătălina NEGHINĂ, Mihai NEGHINĂ

MATLAB

Un limbaj pentru viitorii ingineri

Editura Matrix Rom

2024

MATRIX ROM

Bucuresti 2024

C.P: 16 - 162
062510 - Bucureşti

tel. 021.4113617
fax. 021.4114280

e-mail: office@matrixrom.ro
www.matrixrom.ro

ISBN: 978-606-25-0883-8

Autori:

Șef lucr. dr. ing. Cătălina NEGHINĂ

Universitatea „Lucian Blaga” din Sibiu

Facultatea de Inginerie, Departamentul de Calculatoare și Inginerie Electrică

Șef lucr. dr. ing. Mihai NEGHINĂ

Universitatea „Lucian Blaga” din Sibiu

Facultatea de Inginerie, Departamentul de Calculatoare și Inginerie Electrică

Referenți științifici:

Conf. univ. dr. ing. Arpad GELLERT

Universitatea „Lucian Blaga” din Sibiu

Facultatea de Inginerie, Departamentul de Calculatoare și Inginerie Electrică

Sef lucr. dr. ing. Alina Cristina VIOREL

Universitatea „Lucian Blaga” din Sibiu

Facultatea de Inginerie, Departamentul de Calculatoare și Inginerie Electrică

Cuvânt înainte

MATLAB este o platformă de programare concepută special pentru ingineri și oameni de știință pentru a analiza, proiecta și modela sisteme adaptate cerințelor tehnologie moderne. Limbajul de programare Matlab este optimizat pentru calcul matriceal, îmbinând claritatea descrierilor matematice oferite de algebra liniară cu flexibilitatea metodelor numerice și cu ușurința generării graficelor. Deoarece evoluția tehnologică este un proces continuu și dinamic care a înregistrat progrese uimitoare în ultimele decenii (mai ales în unele domenii cum ar fi *Inteligență Artificială* și *Machine Learning*, *Medicină* și *biotehnologie*, *Realitate virtuală și augmentată* etc) compania MathWorks scoate pe piață anual câte două versiuni de MATLAB.

Cartea de față se adresează tuturor celor ce își doresc să se familiarizeze cu noțiunile de bază ale limbajului de programare *Matlab 2022b* fără a fi nevoie neapărată de cunoștințe apriori de programare, deși pentru o mai bună înțelegere a informațiilor ar ajuta noțiuni de *Programarea calculatoarelor* și de asemenea noțiuni fundamentale din domeniul *Algebrei Liniare*. Această lucrare a venit ca o continuare firească a cărții "MATLAB. *Un prim pas spre cercetare*" [1] fiind folosită ca suport de curs și laborator pentru studenții *Facultății de Inginerie* din cadrul Universității *Lucian Blaga* din Sibiu.

Lucrarea este structurată în 10 capitole ce prezintă informații fundamentale legate de mediul de lucru Matlab, lucrul cu vectori, matrice și șiruri de caractere, reprezentări grafice 2D și 3D, calcul simbolic, lucru cu diverse tipuri de fișiere (imagini, semnale audio, fișiere Excel și text) precum și o multitudine de funcții deja implementate în Matlab care vor ajuta utilizatorul să-și pună mai rapid ideile în practică. O atenție deosebită s-a acordat reprezentării grafice 2D, insistând pe semnalele sinusoidale, deoarece acestea au o importanță aparte în domeniul electronicii.

Majoritatea funcțiilor din Matlab sunt extrem de versatile, din acest motiv cartea prezintă doar modul în care acestea sunt cel mai des utilizate, adresând însă cititorului rugămintea să consulte documentația tuturor funcțiilor prezentate pentru a avea o vedere cât mai de ansamblu.

Autorii

Cuprins

Capitolul 1
Introducere în Matlab

Mediul de lucru MATLAB este dezvoltat de compania MathWorks, de pe site-ul cărora se și poate descărca softul. Tot acolo sunt puse la dispoziție o serie de tutoriale video de la nivel începător (Matlab Onramp) până la tutoriale axate pe anumite domenii (statistică, rețele neurale, procesare de imagini etc)

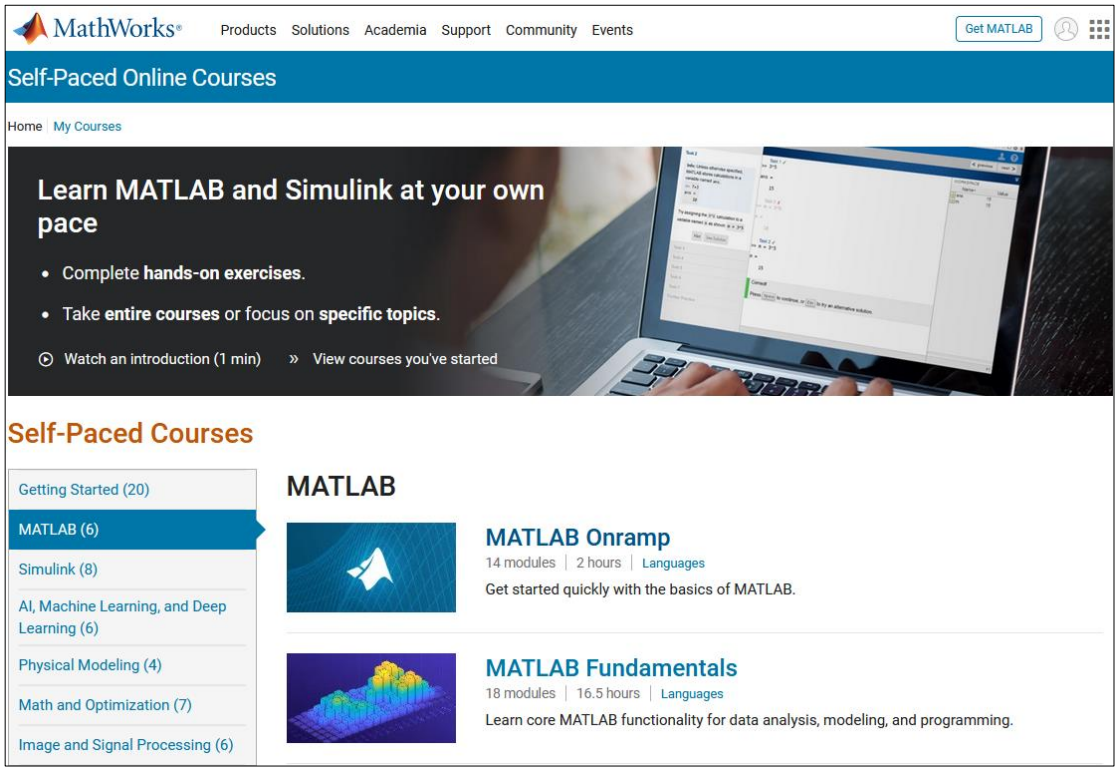


Figura 1.1. Site-ul companiei MathWorks, de unde se poate descărca soft-ul Matlab

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**, iar acest lucru îl sugerează chiar numele limbajului care vine de la **"matrix laboratory"**.

Limbajul MATLAB este optimizat pentru operații matriceale.

MATLAB = **MAT**rix **LAB**oratory

- Se poate lucra cu o varietate de fișiere (fișiere text, fișiere Excel etc) și tipuri de semnale (imagini, semnale audio, semnale video etc).
- Reprezentarea grafică a funcțiilor este extrem de variată: se pot face reprezentări 2D, 3D, histograme, reprezentări procentuale, reprezentări vectoriale etc.
- 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.
- Documentația funcțiilor din Matlab este foarte elaborată, majoritatea funcțiilor având și exemple de utilizare în diverse contexte.
- În Matlab se pot realiza ușor interfețe grafice folosind *App Designer*.
- Matlab-ul dispune de o serie de biblioteci de funcții (*Apps*) 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 R2022b. Majoritatea funcțiilor descrise aici sunt însă disponibile și în versiunile anterioare ala MATLAB-ului.

1.1. Mediul de lucru Matlab

Mediul de lucru Matlab este constituit din mai multe ferestre, cele mai importante fiind:

- **Command Window (Fereastra de comandă):** Este o fereastră de comandă interactivă în care se pot introduce comenzi Matlab și se returnează rezultatele. Este utilă pentru teste rapide, explorarea datelor și pentru a înțelege comportamentul funcțiilor.
- **Workspace (Spațiul de lucru):** Afișează variabilele curente din mediul de lucru, împreună cu valorile acestora.
- **Curent Folder:** Aici sunt afișate toate fișierele și subfolderele din folderul curent

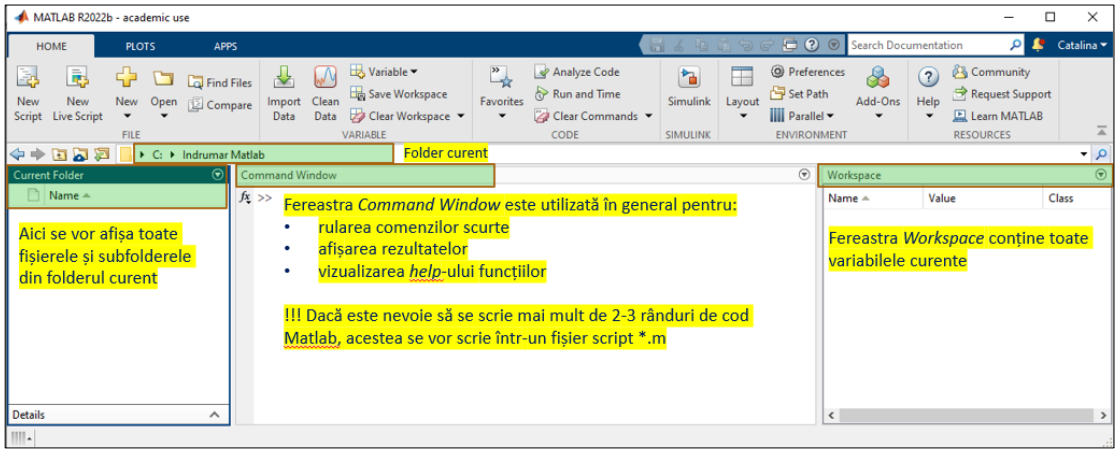


Figura 1.2. Interfața Matlab-ului în modul *Default*

Pentru a afișa interfața Matlab-ului în modul *Default*, se selectează:

Home ➔ Layout ➔ Default

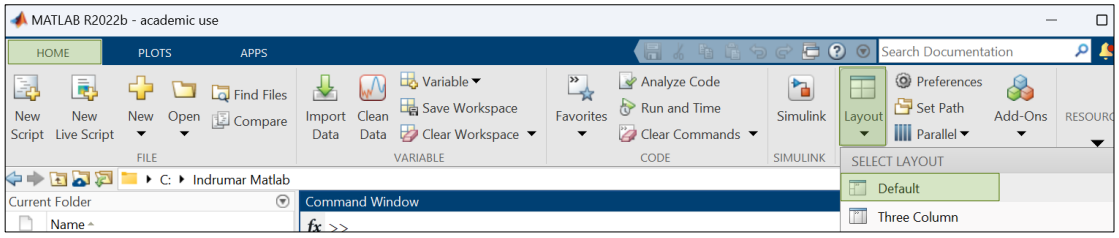


Figura 1.3. Selectarea modului *Default*

1.2. Documentație Matlab

Funcțiile din Matlab au documentație care include de cele mai multe ori, pe lângă descrierea detaliată a parametrilor funcțiilor și exemple de utilizare. Există mai multe moduri de a accesa aceste informații, cele mai uzuale fiind comenzile `help` și `doc`.

- **Comanda `help`**

Sintaxă: `help` *`nume`*

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

În fereastra *Command Window* să se afișeze help-ul funcției `abs`.

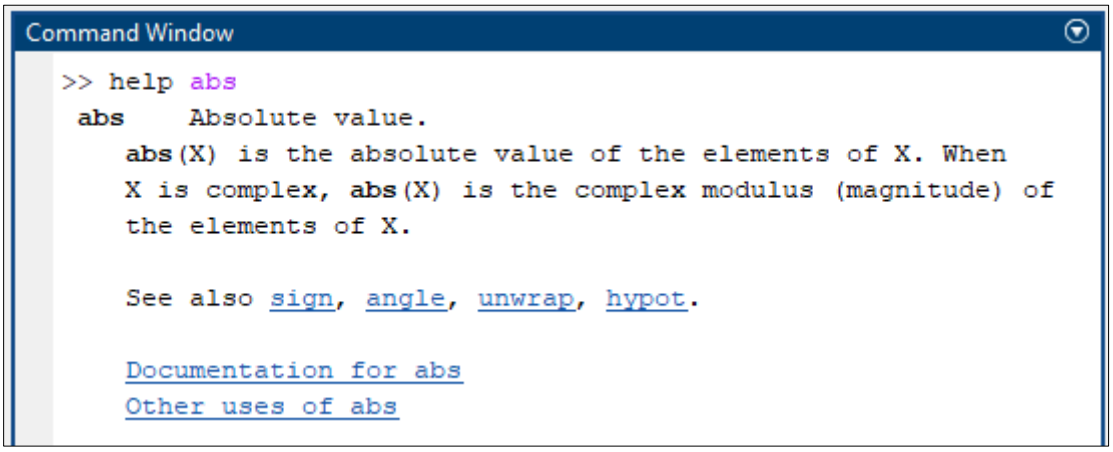


Figura 1.4. Exemplu de afișare a *help*-ului unei funcții

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

- **Comanda `doc`**

Sintaxă: `doc` *`nume`*

Comanda `doc` *`nume`* afișează documentația parametrului `nume` în fereastra *Help*. Spre deosebire de comanda `help`, cu comanda `doc` se afișează mai multe exemple iar textul este formatat, fiind mai ușor de urmărit.

Din fereastra *Command Window* să se afișeze documentația funcției `abs`. În fereastra *Command Window* se va scrie:

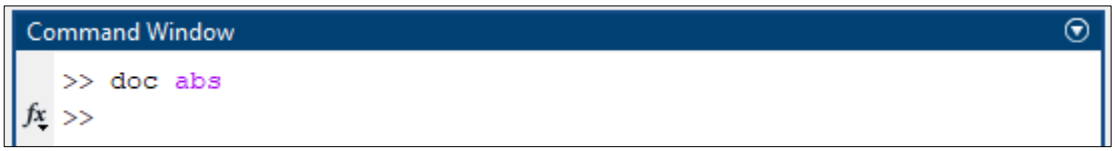


Figura 1.5. Exemplu de apelare a documentației unei funcții

După ce se va apăsa tasta *Enter*, se va deschide fereastra *Help* în care vor fi afișate toate informațiile relevante referitoare la funcția dorită (în cazul de față funcția `abs`).

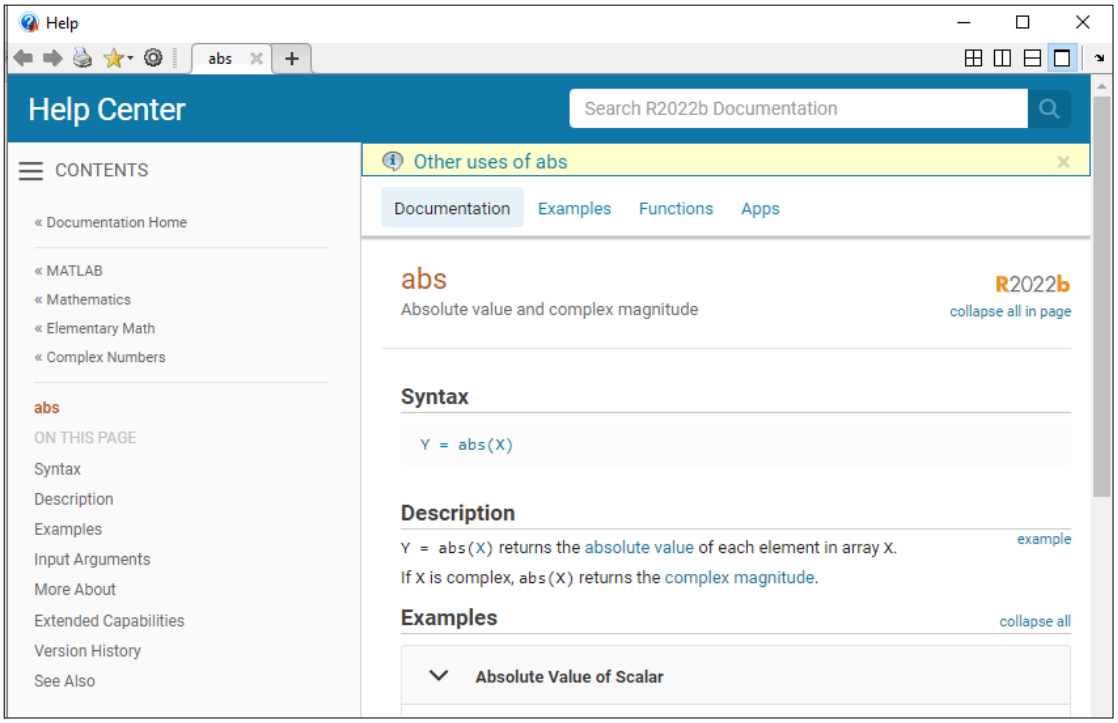


Figura 1.6. Afișarea documentației unei funcții

1.3. Variabile în Matlab

Numele unei variabile trebuie să respecte următoarele reguli:

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

Este indicat a se evita ca numele variabilei să fie același cu cel al unei funcții existente în Matlab (de exemplu `max`, `sum` etc) deoarece numele de variabilă are prioritate față de numele funcției.

MATLAB-ul este *case sensitive*

adică face distincție între litere mici și litere mari.

Inițializarea unei variabile se face astfel:

```
numeVariabila = valoareVariabila
```

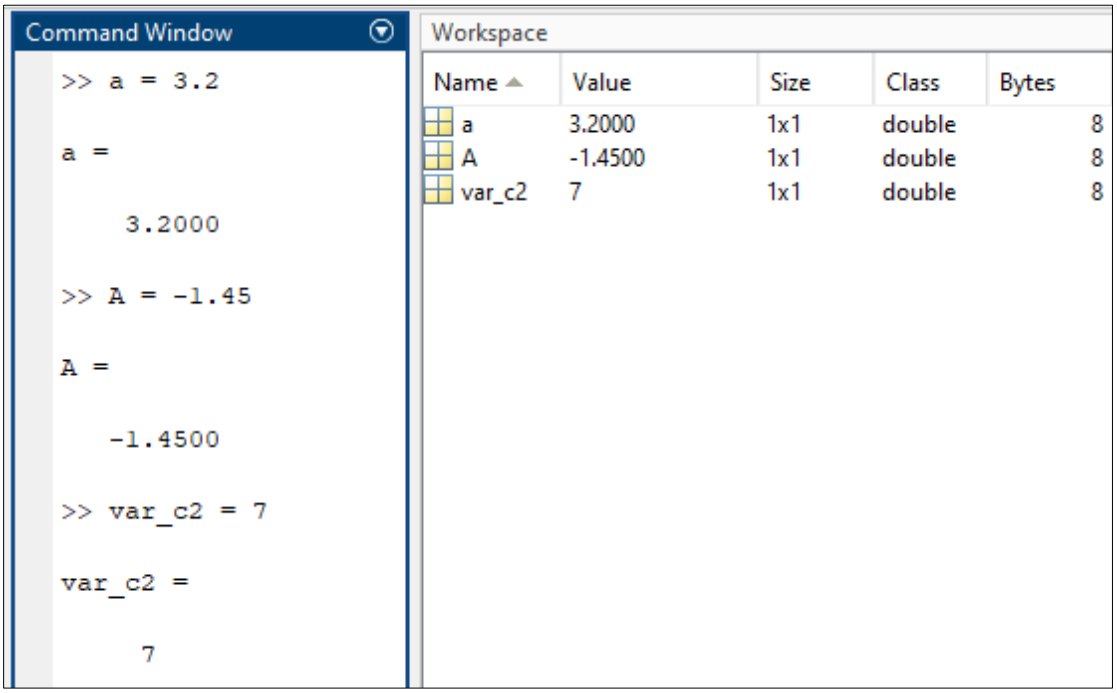


Figura 1.7. Exemple de inițializări corecte de variabile

Așa cum s-a putut observa deja în exemplele anterioare, în Matlab există deja funcția `abs` care calculează modulul unui număr. Dacă definim o variabilă cu numele `abs`, funcția `abs` nu va mai fi recunoscută, așa cum se poate observa în exemplul următor.

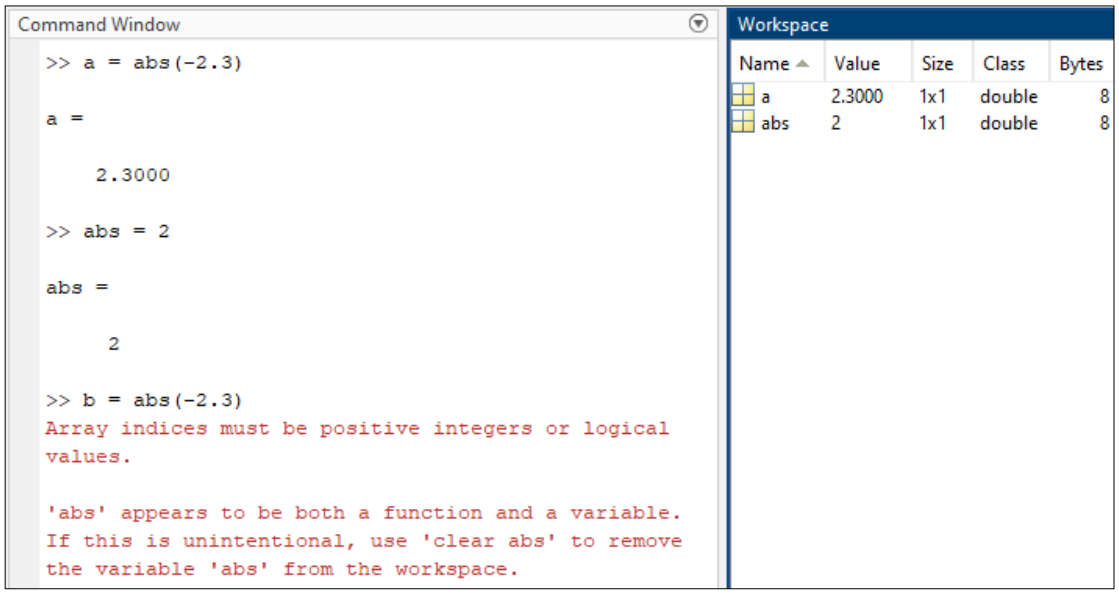


Figura 1.8. Exemplu de inițializare incorectă a unei variabile

În fereastra *Workspace* se poate observa că pentru fiecare variabilă sunt disponibile anumite informații, cele mai importante fiind cele din tabelul următor.

Tabel 1.1. Parametrii unei variabile vizibili în fereastra *Workspace*

Parametru	Descriere
Name	Numele variabilei
Value	Valoarea variabilei dacă este vorba despre un scalar. Dacă este un vector cu mai mult de 10 elemente se trece dimensiunea (ceea ce este și la *Size*)
Size	Dimensiunea variabilei, astfel: • pentru scalar ➔ 1 x 1 (deoarece scalarul este interpretat ca o matrice cu 1 linie și 1 coloană) • vector linie ➔ 1 x N (unde N reprezintă numărul de elemente) • vector coloană ➔ M x 1 (unde M reprezintă numărul de elemente) • matrice cu un singur strat ➔ M x N (unde M reprezintă numărul de linii și N numărul de coloane) • matrice cu mai multe straturi ➔ M x N x P (unde M reprezintă numărul de linii, N numărul de coloane, P numărul de straturi)
Class	*Class* reprezintă tipul de date. Pentru o valoare de tip numeric, implicit tipul de date va fi *double*.
Bytes	Numărul de octeți pe care este stocată variabila respectivă.

Dacă în fereastra *Workspace* nu apar parametrii din tabelul de mai sus, sau se dorește să se adauge și alți parametri, se va da click dreapta în fereastra *Workspace* pe bara de opțiuni și se bifează/debifează opțiunile dorite.

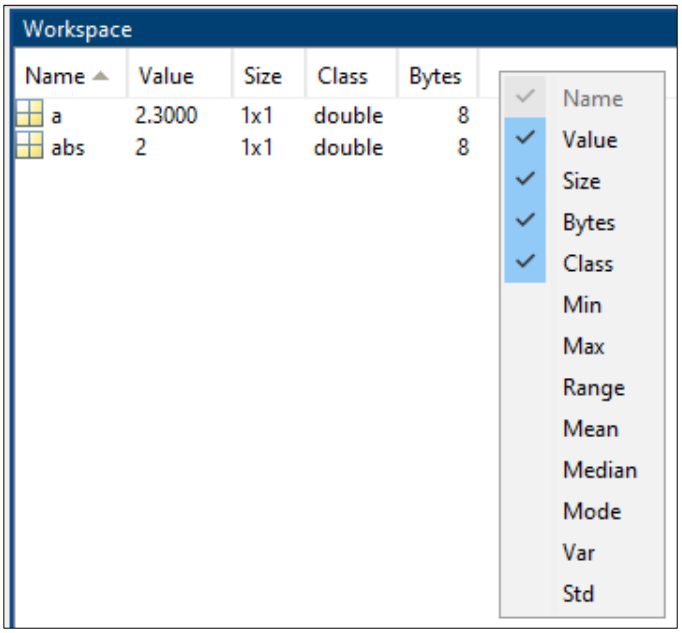


Figura 1.9. Parametrii variabilelor vizibili în fereastra *Workspace*

1.4. Variabile predefinite în Matlab

În Matlab există câteva variabile predefinite și anume:

Variabilă	Valoare

`i`
`j`
	Command Window
	`>> i^2`
	`ans =`
	`-1`
	`>> j^2`
	`ans =`
	`-1`
	`>> sqrt(-1)`
	`ans =`
	`0.0000 + 1.0000i`

`pi`	Command Window
	`>> pi`
	`ans =`
	`3.1416`

`eps`	Command Window
	`>> eps`
	`ans =`
	`2.2204e-16`

`Inf` = *infinity*	Command Window
	`>> 1/0`
	`ans =`
	`Inf`

`NaN` =
Not a Number
	Command Window
	`>> 0/0`
	`ans =`
	`NaN`

1.5. Tipuri de date în Matlab

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`. În diagrama de mai jos sunt principalele tipuri de date utilizate în Matlab.

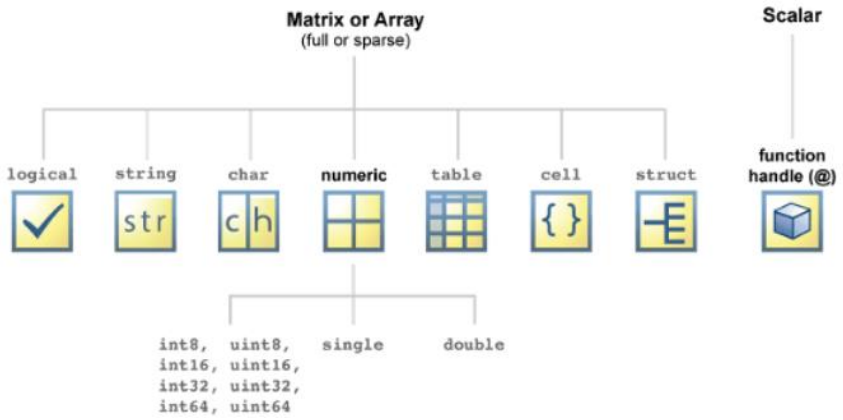


Figura 1.10. Principalele tipuri de date în MATLAB [4]

1.5.1. Tipul de date `logical`

Acest tip de date are doar valorile 0 și 1. Valoarea 0 înseamnă FALS și valoarea 1 înseamnă ADEVĂRAT.

1.5.2. Tipuri de date numerice

Așa cum se poate vedea în *Figura 1.10*, există mai multe tipuri de date pentru a stoca o variabilă numerică, iar acestea sunt descrise în *Tabelul 1.2*.

În Matlab, pentru **valori numerice**, **tipul implicit de date este *double***.

În Matlab, **numerele zecimale se scriu cu punct**.
Exemplu:
```
a = 3.56
```

Tabel 1.2. Tipuri de date numerice în Matlab

Clasa	Domeniul de valori	Observații
int8	$-2^7 \div 2^7-1$	Numere întregi (cu semn) *cifra de la sfârșit reprezintă numărul de biți pe care este stocată variabila*
int16	$-2^{15} \div 2^{15}-1$	
int32	$-2^{31} \div 2^{31}-1$	
int64	$-2^{63} \div 2^{63}-1$	
uint8	$0 \div 2^8-1$	Numere naturale (fără semn) *cifra de la sfârșit reprezintă numărul de biți pe care este stocată variabila*
uint16	$0 \div 2^{16}-1$	
uint32	$0 \div 2^{32}-1$	
uint64	$0 \div 2^{64}-1$	
single	-3.4028e+38 ÷ -1.1755e-38 +1.1755e-38 ÷ +3.4028e+38	Numere reale în precizie simplă
double	-1.79769e+308 ÷ -2.22507e-308 +2.22507e-308 ÷ +1.79769e+308	Numere reale în dublă precizie cu virgulă mobilă

Formatarea valorilor numerice

Implicit, un număr zecimal în Matlab este afișat cu 4 zecimale (indiferent de numărul de zecimale cu care este calculat).

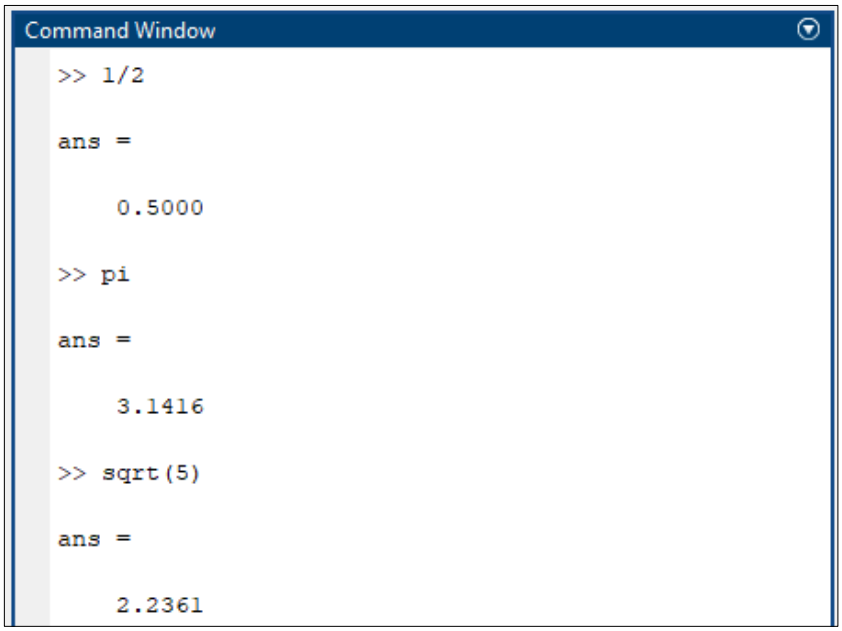


Figura 1.11. Afișarea numerelor zecimale în Matlab.
În modul *default*, afișarea se face cu 4 zecimale

- Dacă se dorește afișarea unei valori cu numărul maxim de zecimale, se folosește comanda `format long`
- Dacă se dorește revenirea la afișarea cu 4 zecimale, se folosește comanda `format short`
- Dacă se dorește afișarea cu 2 zecimale, se folosește comanda `format bank`

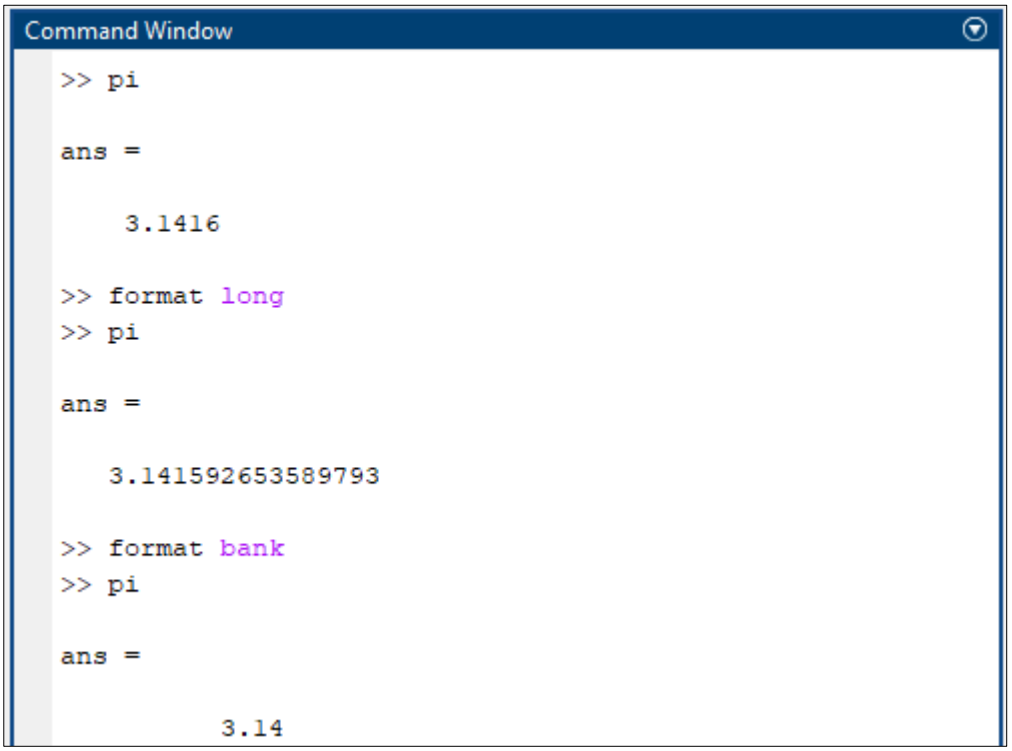


Figura 1.12. Afișarea numerelor în diverse formate

Dacă un număr are mai mult de 9 cifre, atunci el se va afișa în formatul `shortE` (format științific scurt), care se citește astfel: $xe+6 = x \cdot 10^{6}$ și $xe-6 = x \cdot 10^{-6}$.

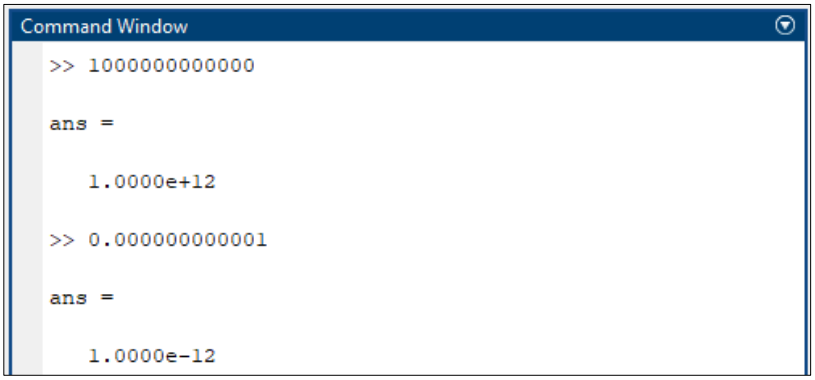


Figura 1.13. Afișarea numerelor în format `shortE`

1.5.3. Tipuri de date pentru caractere

Pentru variabile care să conțină text, există două tipuri de date: `char` și `string`.

- Un text stocat într-o variabilă de tip `char` este un vector de caractere. O variabilă de tip `char` se scrie între **apostroafe (' ')**
- Un text stocat într-o variabilă de tip `string` este un scalar de caractere. O variabilă de tip `string` se scrie între **ghilimele (" ")**

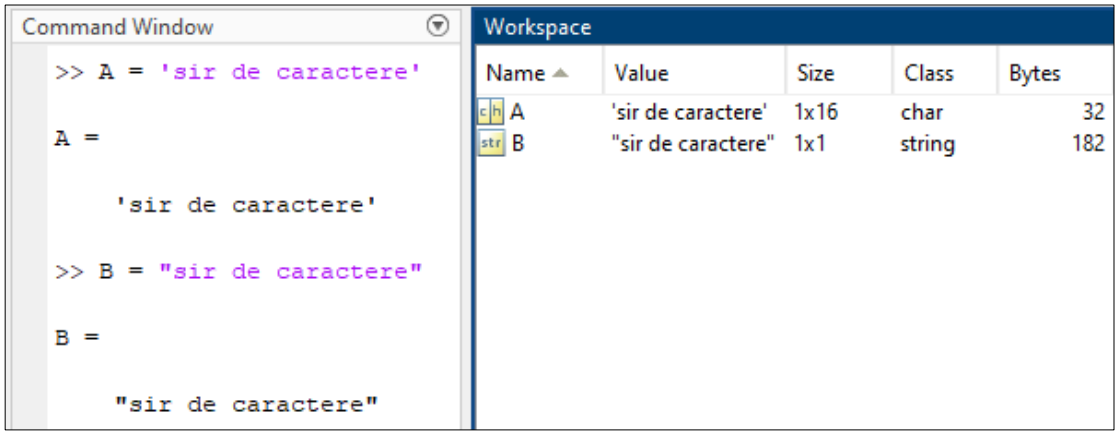


Figura 1.14. Tipurile de date `char` și `string`

Lucrul cu caractere va fi detaliat mai pe larg în *Capitolele 2* și *3*.

1.5.4. Declararea/conversia la tipul de date dorit

Pentru a salva o variabilă cu un anumit tip de date, se folosește sintaxa:

```
numeVar = tipDate(valoare)
```

unde:

- `valoarea` = valoarea ce se stochează în variabilă
- `tipDate` = tipul de date dorit
- `numeVar` = variabila în care se salvează valoarea cu tipul de date dorit

Să se inițializeze variabila `a` cu valoarea 2. Ce tip de date va avea variabila `a` și câți biți va ocupa? Să se salveze în variabila `b`, valoarea 100 având tipul de date `uint8`. Câți biți va ocupa variabila `b`?

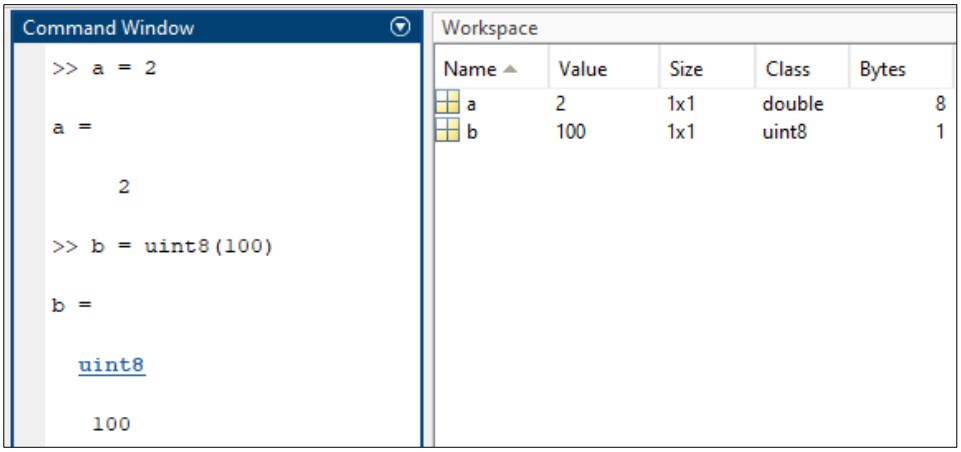


Figura 1.15. Salvarea unei valori cu tip de date impus de utilizator

Pentru a realiza conversia dintr-un tip de date într-un alt tip de date se folosește sintaxa:

```
numeVar2 = tipNouDate(numeVar1)
```

- `numeVar1` = variabila cu tipul de date inițial
- `tipNouDate` = tipul de date în care se realizează conversia
- `numeVar2` = variabila cu noul tip de date

Să se convertească variabila `x` cu valoarea 5 având tipul de date `int8`, în variabila `y` de tip de date `double`.

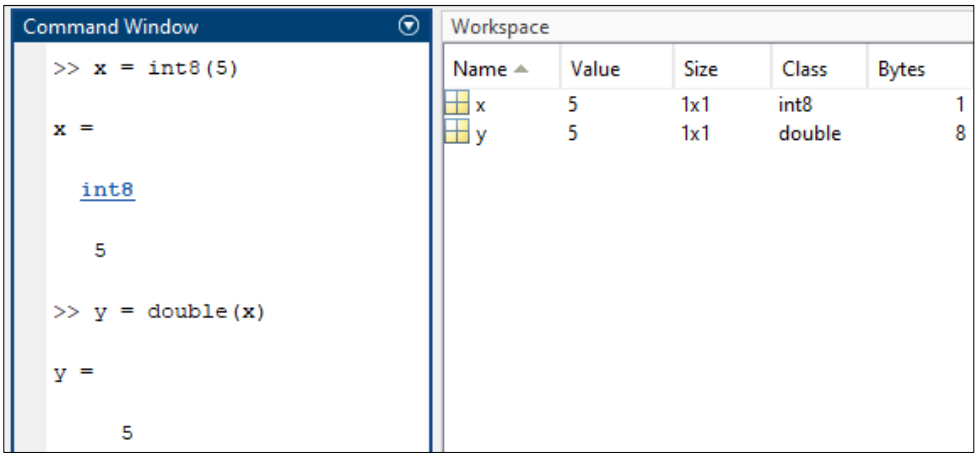


Figura 1.16. Conversia la un alt tip de date

1.6. Caracterul special *punct și virgulă* în Matlab

Caracterul *punct și virgulă* poate fi folosit în diverse contexte printre care:

- **Suprimarea afișării rezultatelor**: dacă se dorește realizarea unei operații sau atribuirea valorii unei variabile fără a afișa rezultatul în consolă, se poate folosi *punct și virgulă* la sfârșitul comenzii.

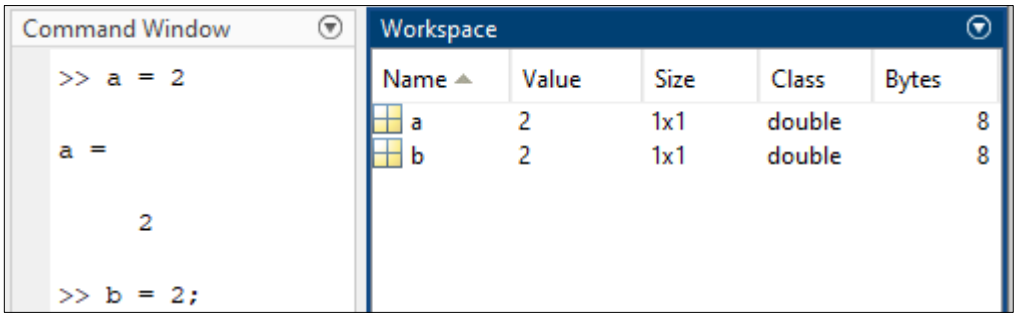


Figura 1.17. Exemplu de utilizare a operatorului *punct și virgulă* pentru a suprima afișarea rezultatului unei variabile (valabil pentru variabila `b`)

În exemplul de mai sus 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*.

Observație: deși în Matlab (spre deosebire de alte limbaje de programare) nu este obligatoriu ca o instrucțiune să se termine cu *punct și virgulă,* este adesea preferată această scriere pentru a evita afișarea excesivă a rezultatelor și pentru a menține consola curată (fereastra *Command Window*).

- **Scrierea mai multor comenzi pe aceeași linie**: *punctul și virgula* poate fi folosit pentru a scrie mai multe comenzi pe aceeași line.

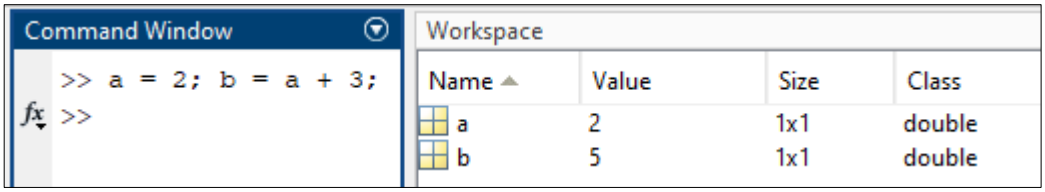


Figura 1.18. Exemplu de utilizare a operatorului *punct și virgulă* pentru a separa instrucțiuni scrise pe aceeași linie

Observație: caracterul *punct și virgulă* mai poate fi folosit și în alte contexte, de exemplu pentru a separa liniile unei matrice, așa cum se va vedea în *Capitolul 2*.

1.7. Operații matematice de bază în Matlab

Simbol	Semnificație	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)

1.8. Operatori relaționali în Matlab

Simbol	Semnificație	Exemplu
=	atribuire (ia valoarea)	`a = 2`
==	verifică egalitatea	`a == 2`
~=	diferit	`a ~= 2`
<	strict mai mic	`a < 2`
<=	mai mic sau egal	`a <= 2`
>	strict mai mare	`a > 2`
>=	mai mare sau egal	`a >= 2`

1.9. Operatori logici în Matlab

Simbol	Semnificație	Exemplu

&& (and)	ȘI logic	`1&&0 = 0`
		`1&&1 = 1`
		`0&&0 = 0`

|| (or)	SAU logic	`1||0 = 1`
		`1||1 = 1`
		`0||0 = 0`

~ (not)	NU logic	`~1 = 0` `~0 = 1`

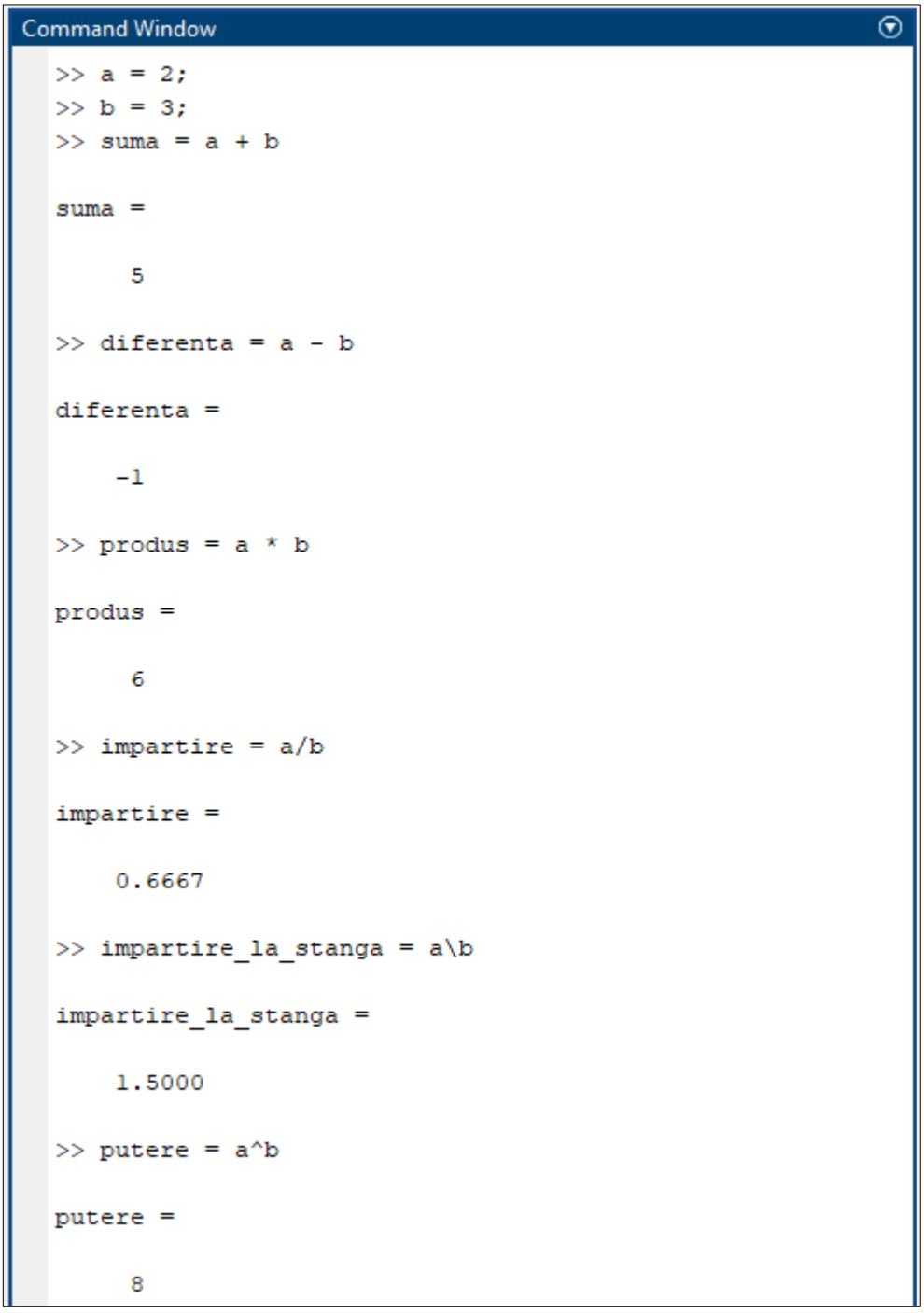


Figura 1.19. Exemple de utilizare a operațiilor matematice de bază

1.10. Funcții matematice uzuale în Matlab

Sintaxă MATLAB	Expresia matematică echivalentă
`sin(x), cos(x)` `tan(x), cot(x)`	sin(x), cos(x), tg(x), ctg(x) cu x exprimat în radiani
`asin(x), acos(x)` `atan(x), acot(x)`	Funcții trigonometrice inverse
`deg2rad(x)`	Transformă x grade în radiani
`sqrt(x)`	Rădăcina pătrată a lui x ➔ $\sqrt{x}$
`nthroot(x,n)`	Radical de ordin n din x ➔ $\sqrt[n]{x}$
`exp(x)`	Funcția exponențială ➔ e^x
`abs(x)`	Funcția modul ➔ $\|x\|$
`mod(x,y)`	Restul împărțirii lui x la y
`log(x)`	Logaritmul natural➔ $\ln(x)$
`log2(x)`	Logaritmul în baza 2 ➔ $\log_2(x)$
`log10(x)`	Logaritmul în baza 10➔ $\log_{10}(x)$
`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

```
Command Window
>> sin(pi/2) % sinus de 90 de grade

ans =

     1

>> deg2rad(90) % transforma 90 de grade in radiani

ans =

    1.5708

>> sqrt(9) % radical din 9

ans =

     3

>> nthroot(27,3) % radical de ordin 3 din 27

ans =

     3

>> exp(2) % constanta lui Euler la puterea a doua

ans =

    7.3891

>> abs(-3.4) % calcul modul

ans =

    3.4000

>> rem(9,4) % restul impartirii lui 9 la 4

ans =

     1
```

Figura 1.20. Exemple de utilizare a funcțiilor matematice de bază

1.11. Operații cu numere complexe în Matlab

Fie numărul complex cu forma algebrică:

$$z = a + b \cdot i$$

Sintaxă Matlab	Expresia matematică echivalentă
`real(z)`	partea reală a unui număr complex z *matematic* ➔ $Re(z) = a$
`imag(z)`	partea imaginară a unui număr complex z *matematic* ➔ $Im(z) = b$
`conj(z)`	conjugatul unui număr complex z *matematic* ➔ $\bar{z} = a - b \cdot i$
`abs(z)`	modulului unui număr complex z *matematic* ➔ $\|z\| = \sqrt{a^2 + b^2}$

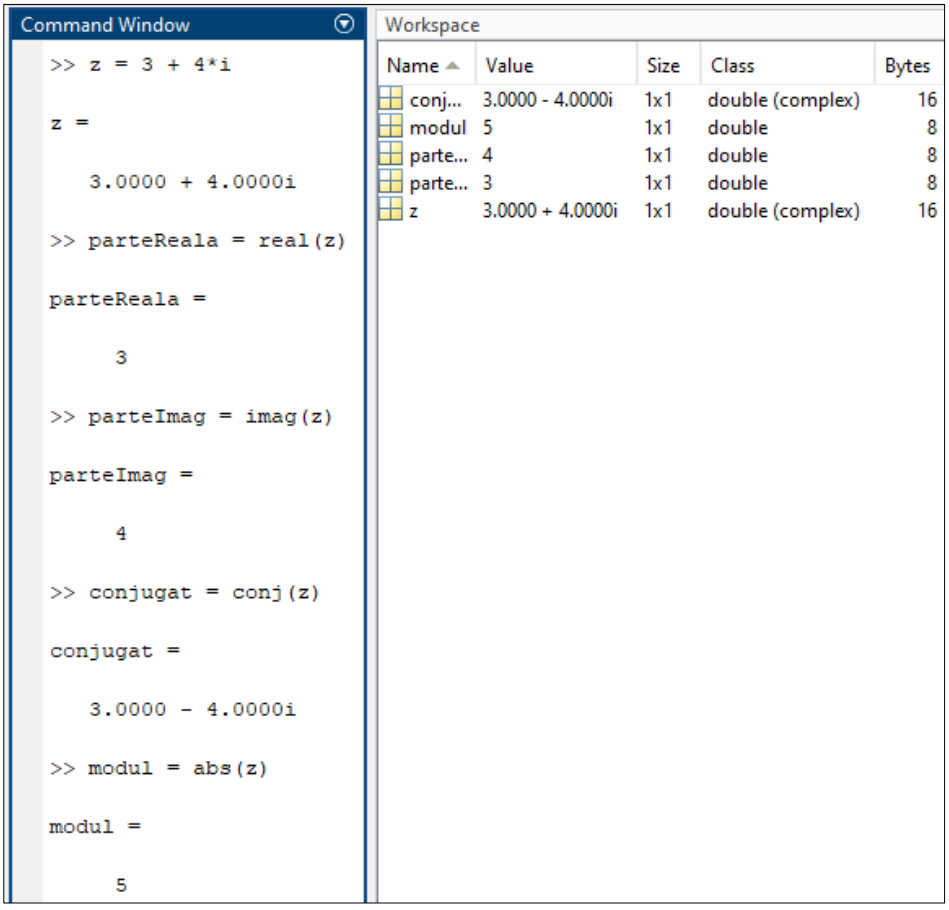


Figura 1.21. Exemplu de operații cu numere complexe

1.12. Editorul MATLAB

Există două tipuri de fișiere în care se poate scrie codul Matlab:

- fișiere **.m* care se deschid cu *New Script*
- fișiere **.mlx* care se deschid cu *New Live Script*.

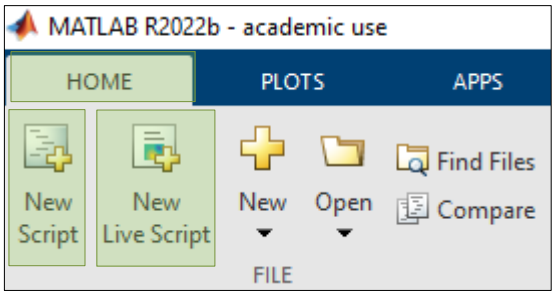


Figura 1.22. Cele două tipuri de fișiere script în Matlab: *New Script* și *New Live Script*

Numele sub care se salvează un fișier trebuie să respecte următoarea regulă: să înceapă cu literă și să conțină numai litere, cifre sau *underline* (_).

Atenție: numele unui fișier nu poate conține spațiu!

1.12.1. Fișierele *New Script*

Un fișier script deschis cu *New Script*, este un fișier *m*-file (se salvează cu extensia **.m*) și poate conține doar cod Matlab și comentarii.

Observație: dacă numele fișierului nu respectă regulile de mai sus (să înceapă cu literă și să conțină numai litere, cifre sau *underline*) atunci Matlabul va semnaliza acest lucru, ca în exemplul de mai jos.

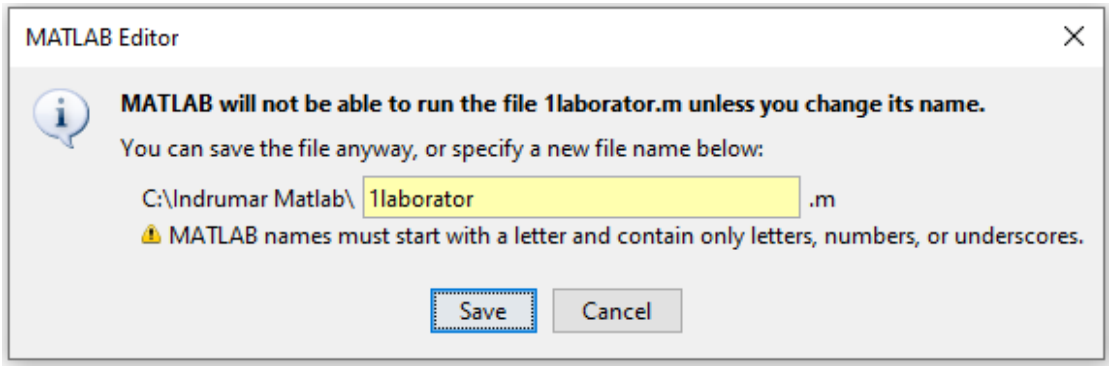


Figura 1.23. Exemplu de denumire **incorectă** a unui fișier Matlab

Pentru a rula un program, se poate folosi tasta **F5** sau se poate alege opțiunea *Editor* și apoi *Run* (săgeata verde) așa cum se arată în exemplul următor.

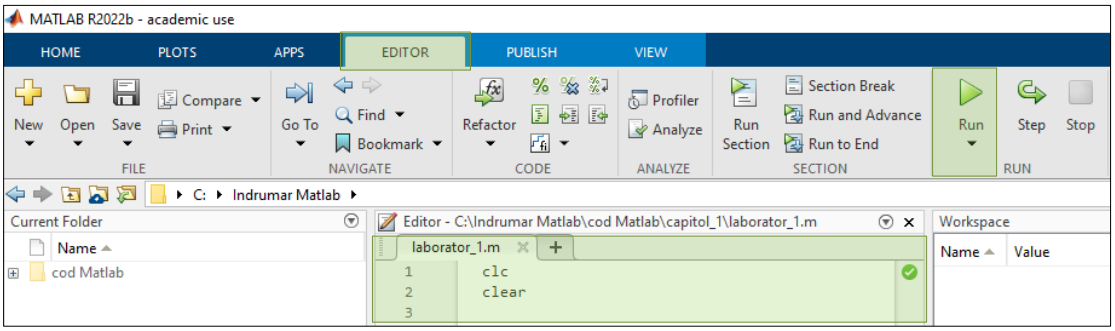


Figura 1.24. Rulare **c**od Matlab *m*-file folosind *Run* sau F5

Este bine ca primele linii ale programului să fie `clc` și `clear`, 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` șterge toate variabilele din *Workspace*.

Observație: așa cum se poate observa în *Figura 1.24* de mai sus:

- directorul curent este: *C:\Indrumar Matlab*
- fișierul Matlab este salvat în directorul:
  C:\Indrumar Matlab\cod Matlab\capitol_1

Dacă directorul curent diferă de directorul în care este salvat fișierul Matlab, la rularea programului (F5), va apărea următorul mesaj:

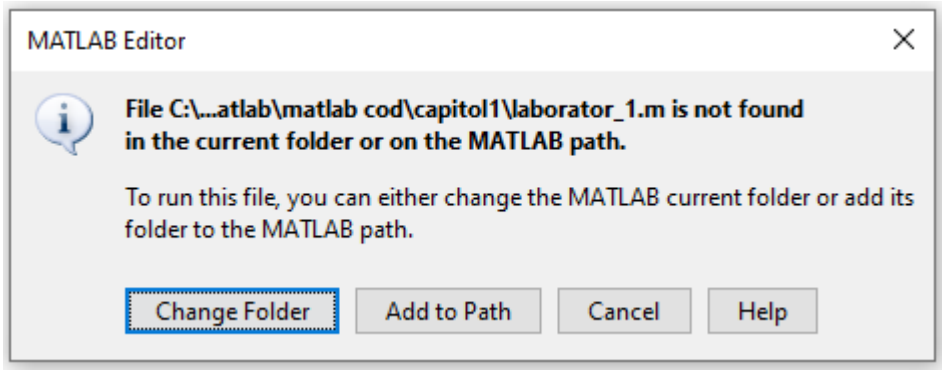


Figura 1.25. Mesaj care apare la rularea programului atunci când directorul curent diferă de directorul în care este salvat fișierul Matlab

La apariția mesajului de mai sus se va alege opțiunea *Change Folder*, pentru a schimba calea directorului curent să fie identică cu calea în care se află fișierului Matlab.

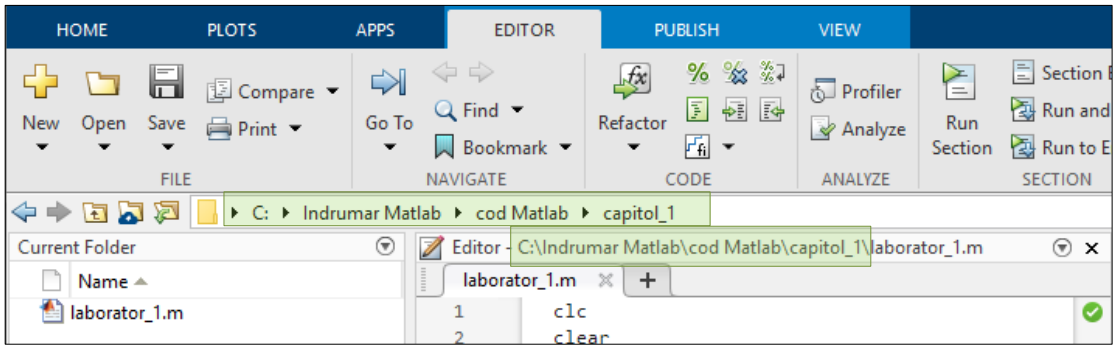


Figura 1.26. Directorul curent și directorul în care este salvat fișierul Matlab după ce s-a rulat programul și s-a ales opțiunea *Change Folder*

Comentarii și secțiuni în fișierele *New Script*

- Pentru a comenta o linie de cod se folosește semnul procent (`%`). Linia comentată va fi scrisă cu verde.
- Pentru a separa fișierul script în secțiuni se folosește de 2 ori semnul procent urmat de spațiu (`%% `). **Atenție!** După `%%` se lasă obligatoriu spațiu dacă se dorește secțiune. Fără spațiu este interpretat ca un comentariu normal.

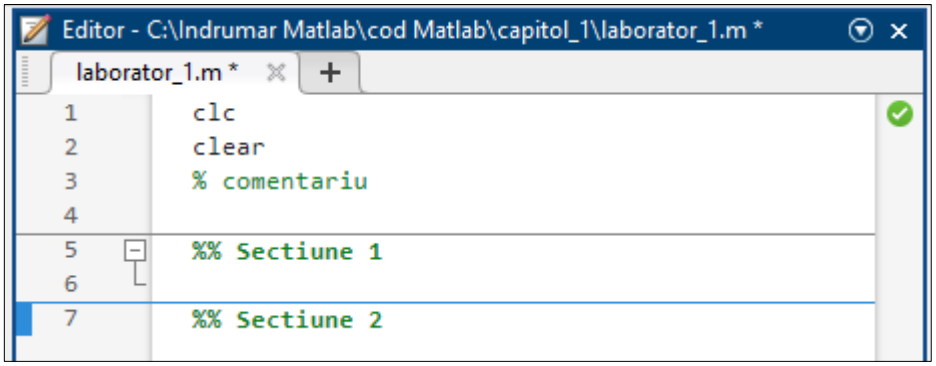


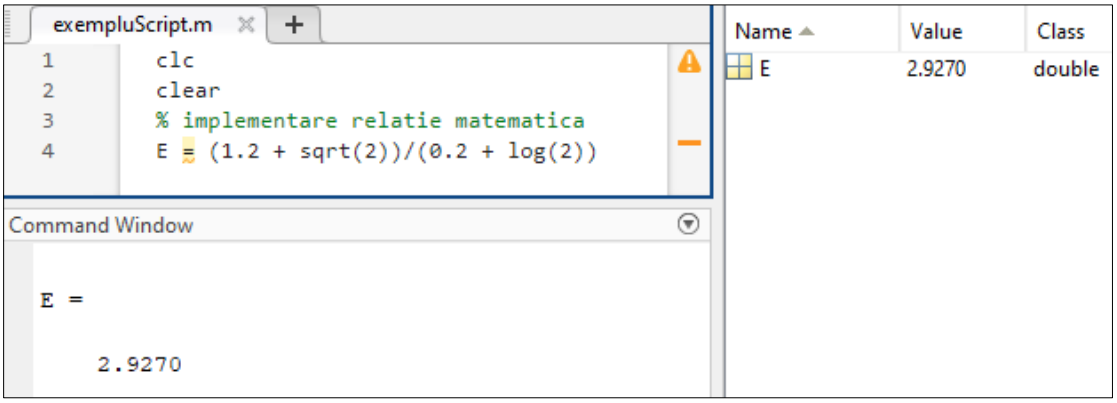
Figura 1.27. Împărțirea codului pe secțiuni

Observații:

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

Să se scrie într-un fișier *New Script* (fișier *m*-file) codul Matlab pentru implementarea relației matematice:

$$E = \frac{1.2 + \sqrt{2}}{0.2 + \ln(2)}$$



1.12.2. Fișierele *New Live Script*

Un fișier script deschis cu *New Live Script*, este un fișier *mlx*-file (se salvează cu extensia **.mlx*). Numele unui fișier **.mlx* trebuie să respecte aceleași reguli ca și fișierele **.m* descrise anterior.

Un astfel de fișier conține codul Matlab și tot aici se pot vizualiza rezultatele obținute precum și graficele generate. În plus, acest tip de fișier mai poate conține text care poate fi formatat precum și imagini și ecuații.

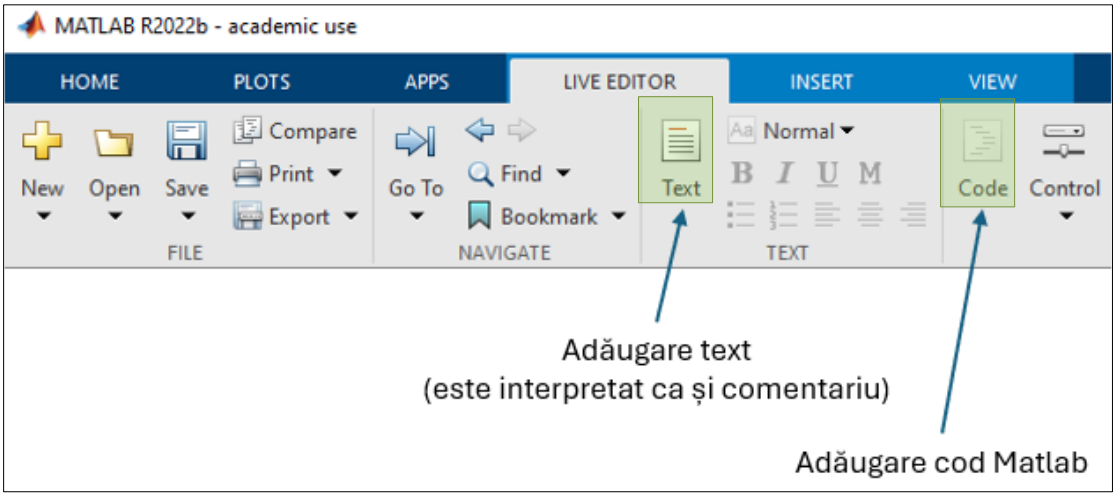
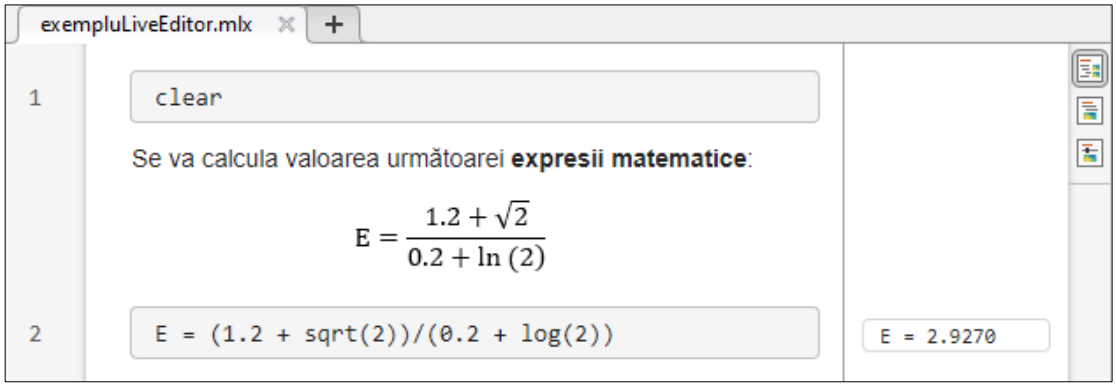


Figura 1.28. Fișierul *New Live Script*

Să se scrie într-un fișier *New Live Script* (fișier *mlx*-file) codul Matlab pentru implementarea următoarei relații matematice:

$$E = \frac{1.2 + \sqrt{2}}{0.2 + \ln(2)}$$



1.13. Comenzi utile în Matlab

Comandă/instrucțiune	Efectul comenzii/instrucțiunii
`clc`	Șterge tot ceea ce s-a afișat în fereastra *Command Window*
`clear`	Șterge toate variabilele din fereastra *Workspace*
`clear numeVar`	Șterge doar variabila cu numele `numeVar` din fereastra *Workspace*
Combinația de taste CTRL + C	Oprește rularea unei aplicații în Matlab, dacă este activă fereastra *Command Window*
CTRL + R	Pentru a comenta mai multe linii de cod în fișierul script
CTRL + T	Pentru a decomenta mai multe linii de cod în fișierul script
F5	Rularea programului scris în *Editor*
Săgeată în sus în *Command Window*	Afișarea comenzilor din *Command Window* în ordine cronologică inversă

1.14. Aplicații

Aplicația 1. De rezolvat în *Command Window*

a) Să se vizualizeze *help*-ul funcției `mod`

b) Să se vizualizeze documentația funcției `exp`

c) Să se afișeze valoarea lui π cu numărul maxim de zecimale posibile în Matlab

d) Să se afișeze valoarea lui π cu 4 zecimale

e) Să se afișeze valoarea lui *e* (constanta lui Euler) cu 2 zecimale

f) Să se calculeze în variabila `rest`, restul împărțirii lui 324 la 13

g) Să se calculeze $\sin(60^o)$

h) Să se rotunjească la cel mai apropiat întreg valoarea lui $\sqrt{13}$

Aplicația 2. Pentru a ocupa cât mai puțin spațiu de memorie, ce tip de date ar trebui folosit pentru a salva următoarele numere?

a) 5 b)120 c) -230 d) 21032 e) valoarea unui pixel dintr-o imagine grayscale

---

Aplicația 3. Într-un fișier *m*-file să se scrie codul Matlab care să calculeze următoarele expresii matematice.

$$E1 = \frac{3}{4\pi}$$

$$E2 = \frac{\sqrt{5} - 1}{1 + log_2 3 + e^2}$$

$$\mathrm{E3} = \frac{\sqrt[3]{2} - 3.2}{0.2 + \ln(3)}$$

$$E4 = \sin^2(x) + cos^2(x) \text{ pentru } x = \frac{\pi}{6}$$

$$E5 = e^{i\pi}$$

Aplicația 4. Într-un fișier *mlx*-file să se scrie codul Matlab care să 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 $\sigma^2 = 0.5$.

Aplicația 5. Fie un număr $a \in \mathbb{N}$, despre care se știe că are 5 cifre. Să se determine automat în Matlab cifra sutelor lui a și să se salveze în variabila `b`.

Aplicația 6. Fie `x` un număr zecimal pozitiv. Să se determine automat în Matlab și să se salveze în `y` prima cifră de după virgulă din `x`.

Aplicația 7. Să se calculeze modulul numărului complex:

$$z = \frac{8+i}{7-4i}$$

Aplicația 8. Să se calculeze conjugatul numărului complex:

$$z = 1 + i + i^2 + i^3 + i^4 + i^5 + i^6$$

Capitolul 2
Vectori, matrice și șiruri de caractere

Elementul de bază cu care lucrează Matlab-ul este **matricea**, acest lucru sugerându-l chiar numele 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.
Observație: Forma de plural a cuvântului **matrice** este tot **matrice**.

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

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & a_{m,3} & \dots & a_{m,n} \end{bmatrix}$$

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`)

$$X = [x_1, x_2, x_3, \dots x_n]$$

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

$$X = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \dots \\ x_{m,1} \end{bmatrix}$$

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

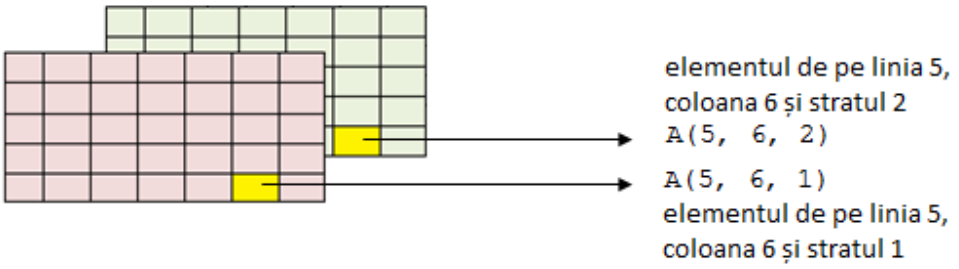


Figura 2.1. Exemplu de matrice cu 5 linii, 7 coloane și 2 straturi

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, $m > 1$, $n > 1$, $p > 1$). Când se va lucra cu o matrice cu o singură linie sau o singură coloană, se va specifica că este vorba despre un vector linie respectiv vector coloană.

!!! Indexarea în Matlab pornește de la **1** !!!

2.1. Vectori în Matlab

- se recomandă ca **definirea** unui vector în Matlab să fie făcută utilizând **paranteze pătrate**

Sintaxă: `numeVectorLinie = [val_1, val_2, … , val_N]`

Observație: deși elementele unui vector pot fi separate și doar prin spațiu, pentru o mai bună vizualizare se recomandă să se folosească virgula, ca în sintaxa de mai sus.

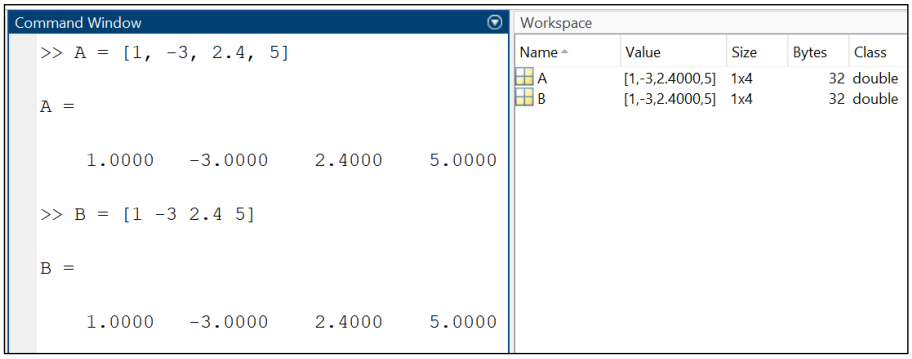


Figura 2.2. Definirea unui vector separând elementele prin virgulă (vectorul A) sau prin spațiu (vectorul B)

- **accesarea** unui element dintr-un vector se face utilizând **paranteze rotunde**

Sintaxă: `valoare = numeVector(indice)`

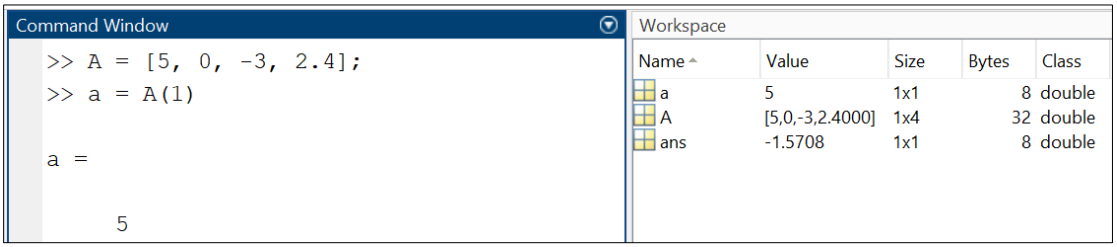


Figura 2.3. Accesarea unui element dintr-un vector

2.2. Matrice în Matlab

- **definirea** unei matrice în Matlab se face utilizând **paranteze pătrate**

Sintaxă: `numeMatrice = [val_1_1, val_1_2, … , val_1_N;`

`val_2_1, val_2_2, … , val_2_N;`

`val_M_1, val_M_2, … , val_M_N]`

Observație: la definirea matricei, pentru a trece pe linia următoare se folosește caracterul special punct și virgulă (;), ca în sintaxa de mai sus.

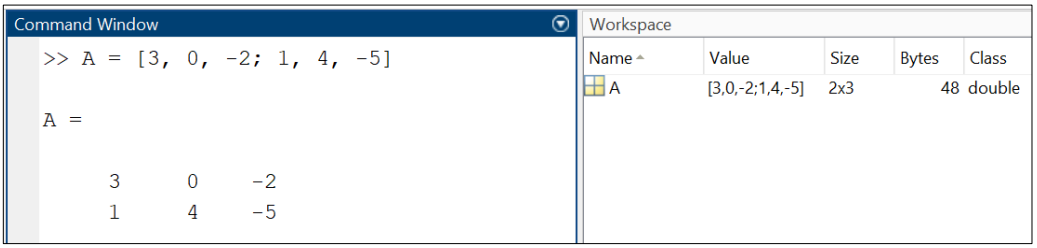


Figura 2.4. Definirea unui matrice

- **accesarea** unui element dintr-o matrice se face utilizând **paranteze rotunde**

Sintaxă: `valoare = numeMatrice(indice_linie, indice_coloana)`

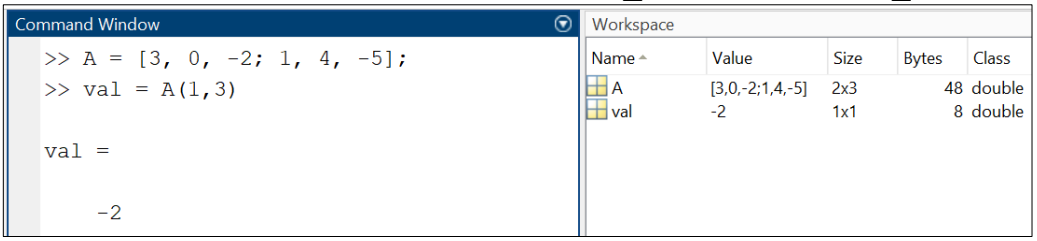


Figura 2.5. Accesarea unui element dintr-o matrice

- definirea un vector nul sau a unei matrice nule

Sintaxă: `numeVar = []`

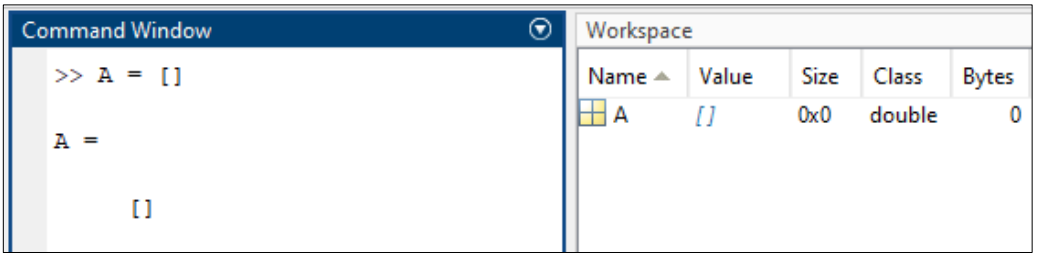


Figura 2.6. Declararea unei variabile nule

2.3. Caracterul special *două puncte* (:)

Caz 1. `X = val_start : pas : val_stop`

Vectorul `X` va conține șirul de numere începând cu valoarea `val_start` și terminând cu valoarea `val_stop`, diferența dintre două valori consecutive fiind dată de parametrul `pas`.

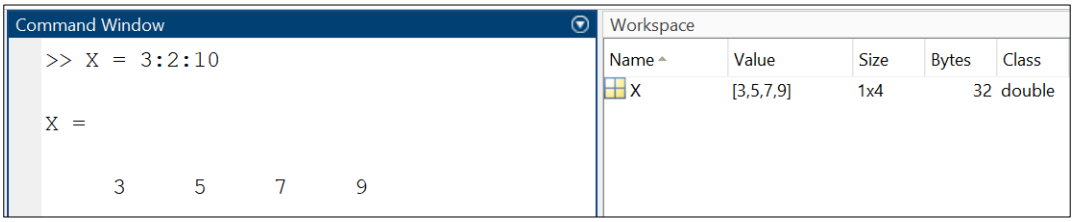


Figura 2.7. Exemplu de utilizare a operatorului două puncte

Observație: dacă nu se specifică parametrul `pas`, acesta va avea implicit valoarea 1.

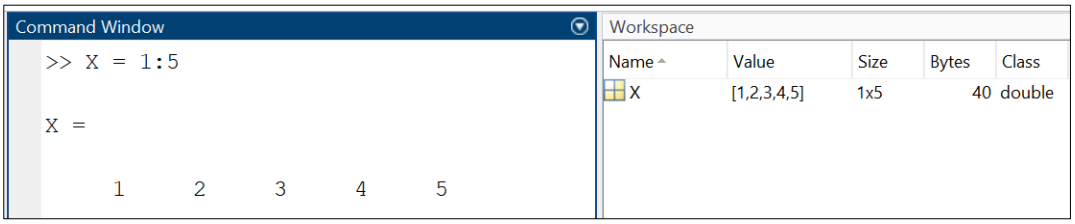


Figura 2.8. Exemplu de utilizare a operatorului două puncte cu pas implicit egal cu 1

Observație: dacă se dorește să se genereze numere consecutive în ordine descrescătoare, trebuie specificat parametrul `pas` ca fiind egal cu -1, altfel se generează un vector nul.

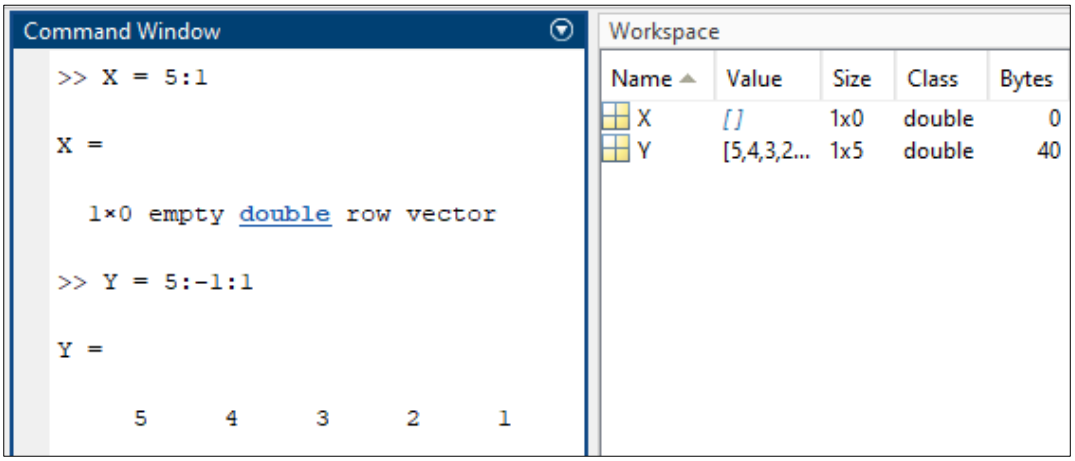


Figura 2.9. Generare incorectă (vectorul `X`) și corectă (vectorul `Y`) a valorilor unui vector în ordine descrescătoare

Caz 2. `B = A(x : y , m : n)`

Matricea `B` va conține elementele din matricea `A`, de la linia `x` la linia `y` și de la coloana `m` la coloana `n`.

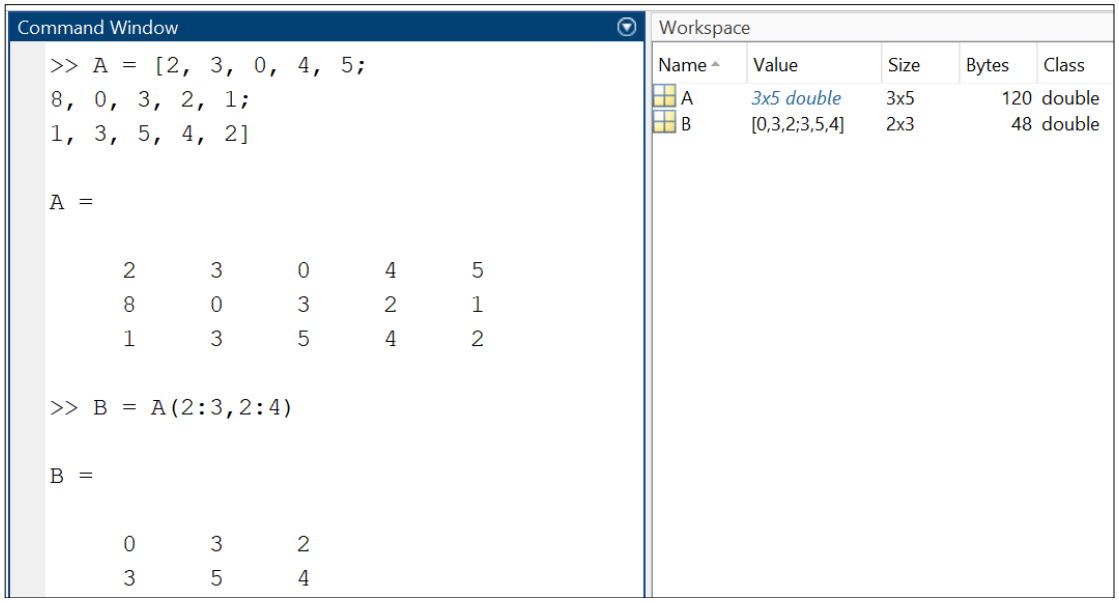


Figura 2.10. Exemplu de utilizare a caracterului special *două puncte* pentru selectarea elementelor dintr-o matrice

Cazuri speciale:

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

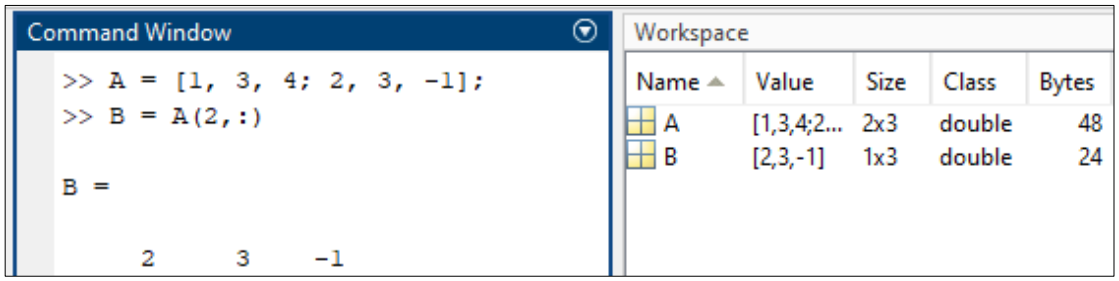


Figura 2.11. Exemplu pentru selectarea unei linii dorite dintr-o matrice

Caz 3. `B = A(:)`

Depinzând de variabila `A`, sintaxa de la cazul 3 poate însemna:

- dacă `A` este **vector coloană**, atunci `B` va fi identic cu `A;`
- dacă `A` este **vector linie**, atunci `B` va conține elementele din A aranjate într-un **vector coloană**;
- dacă `A` este **matrice**, atunci `B` va fi **vector coloană** și va conține elementele din `A` aranjate pe coloane.

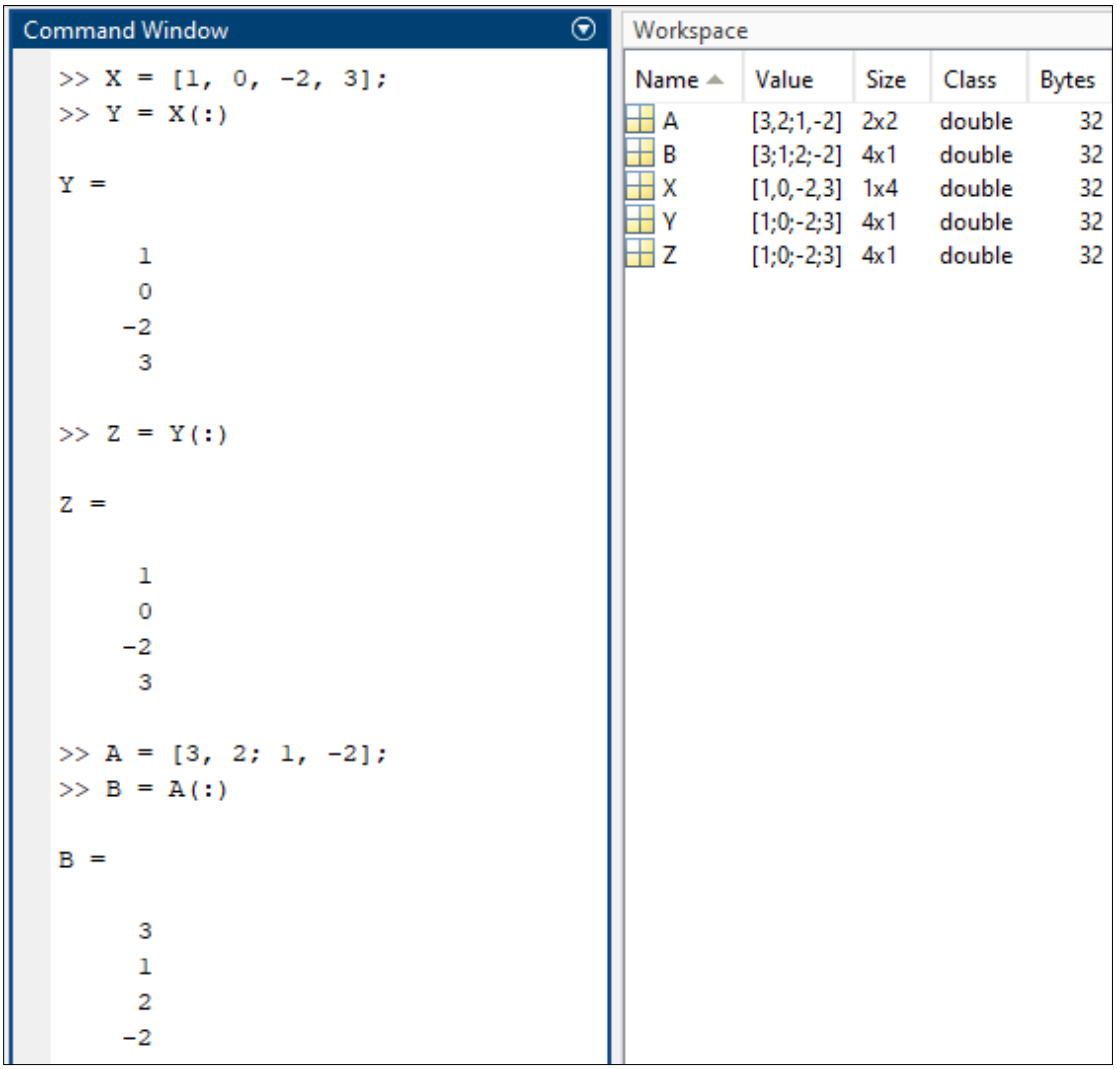


Figura 2.12. Exemplu de utilizare a caracterului special *două puncte* pentru transformarea vectorului linie (`X`) în vector coloană (`Y`) și pentru transformarea unei matrice (`A`) în vector coloană (`B`). Se poate observa că aplicarea caracterului special *două puncte* unui vector coloană (`Y`) nu are niciun efect (vectorul `Z` este identic cu `Y`)

2.4. Operații între scalari și vectori/matrice

Fie `A` un vector sau o matrice. Între `A` și un scalar `n` se pot defini următoarele operații:

Simbol	Rol	Sintaxă	Observații

+	Adunare	`C = n + A`
		sau
		`C = A + n`
			Se adună valoarea `n` la fiecare element din `A`

–	Scădere	`C = n - A`
		sau
		`C = A - n`
			Diferența dintre `n` și fiecare element din `A`
			
			Diferența dintre fiecare element din `A` și `n`

*	Înmulțire	`C = n * A`
		sau
		`C = A * n`
			Se înmulțește `n` cu fiecare element din `A`

/	Împărțire	`C = A/n`
		sau
		`C = n./A`
			Se împarte fiecare element din `A` la `n`
			
			Se împarte `n` la fiecare element din `A`

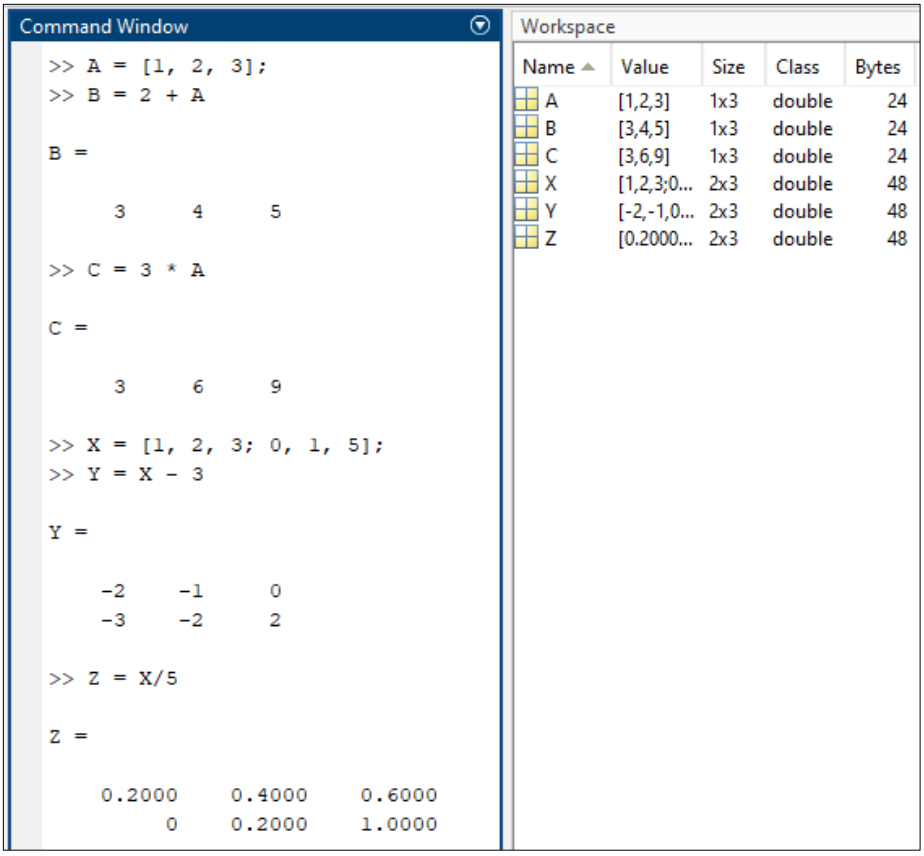


Figura 2.13. Exemplu de operații între scalari și vectori/matrice

2.5. Operații cu vectori

Operațiile cu vectori din Matlab respectă regulile algebrei liniare.

Fie vectorii `A` și `B`. Între `A` și `B` pot fi definite următoarele operații:

Simbol	Rol	Sintaxă	Observații
+	Adunare	`C = A + B`	`A` și `B` să aibă aceleași dimensiuni (ambii vectori linie sau vectori coloană cu `n` elemente). `C` va fi un vector cu `n` elemente.
–	Scădere	`C = A - B`	`A` și `B` să aibă aceleași dimensiuni (ambii vectori linie sau vectori coloană cu `n` elemente). `C` va fi un vector cu `n` elemente.
*	Înmulțire	`C = A * B`	• `A` este vector linie cu `n` elemente și `B` este vector coloană cu `n` elemente. `C` va fi un scalar. • `A` este vector coloană cu `n` elemente și `B` este vector linie cu `n` elemente. `C` va fi o matrice pătratică de ordin `n`.
.'	Transpusa	`C = A.'`	Transformă vector linie în vector coloană și invers.
'	Transpus conjugata	`C = A'`	Transformă vector linie în vector coloană și invers. Dacă vectorul `A` conține numere complexe, atunci se realizează și conjugatul numerelor.

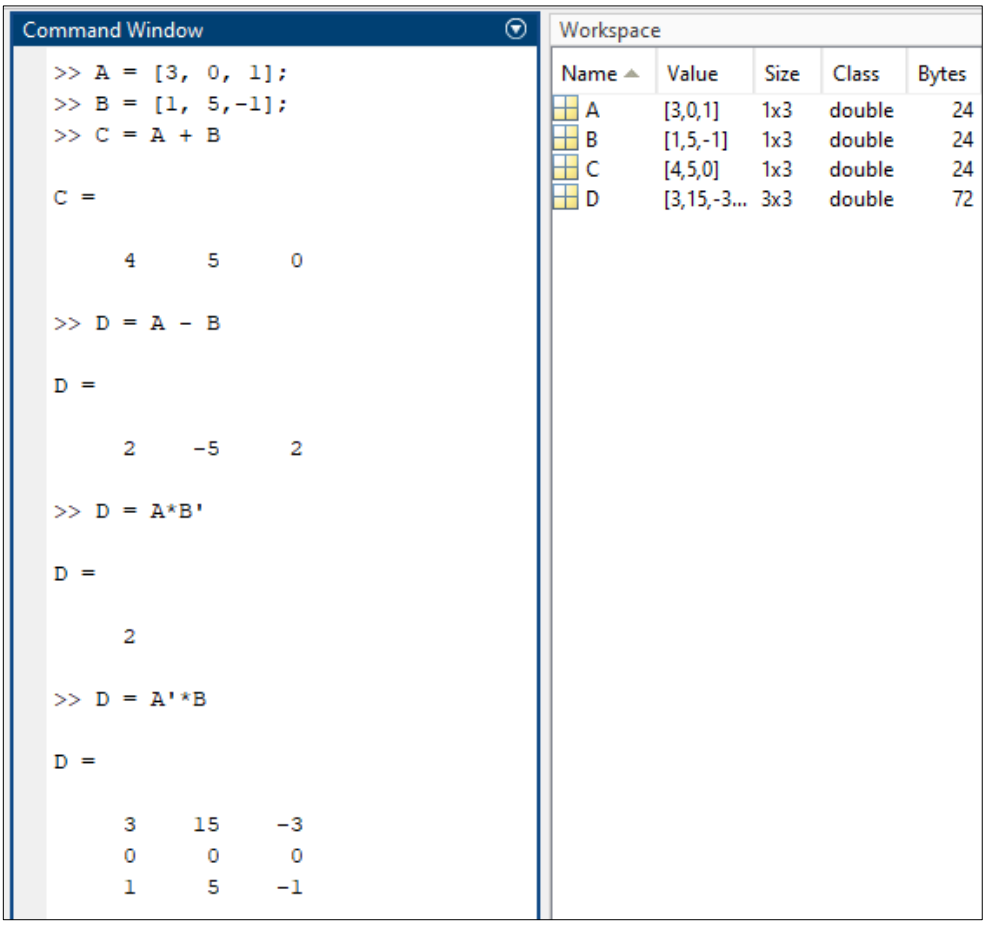


Figura 2.14. Operații între doi vectori

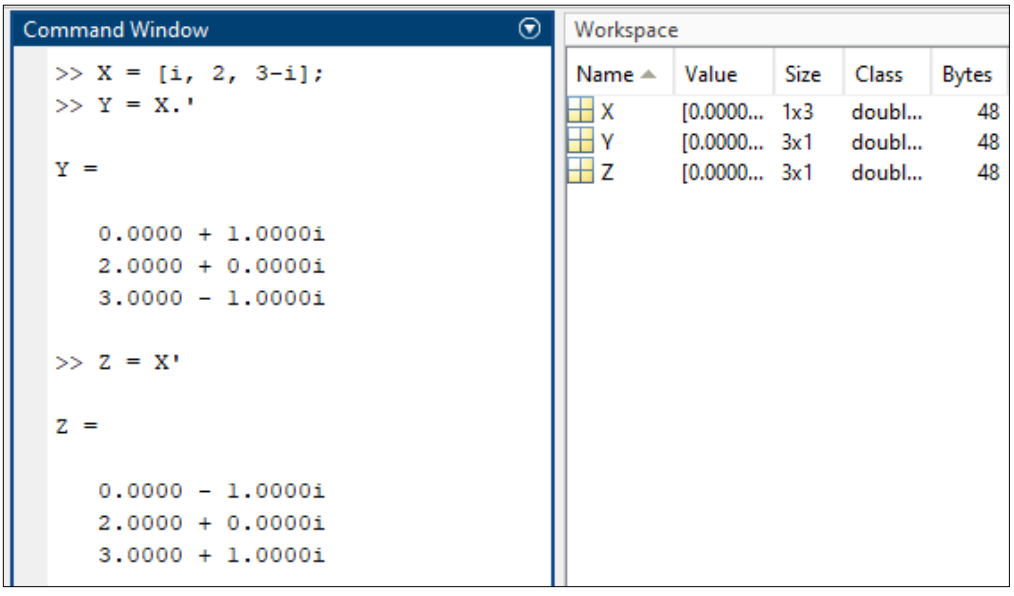


Figura 2.15. Exemplu pentru transpusa și transpus conjugata

2.6. Operații cu matrice

Operațiile cu matrice din Matlab respectă regulile algebrei liniare.

Fie matricele `A` și `B`. Între `A` și `B` pot fi definite următoarele operații:

Simbol	Rol	Sintaxă	Observații
+	Adunare	`C = A + B`	`A` și `B` să aibă aceleași dimensiuni (același număr de linii și de coloane).
–	Scădere	`C = A - B`	`A` și `B` să aibă aceleași dimensiuni (același număr de linii și de coloane).
*	Înmulțire	`C = A * B`	Numărul de coloane din `A` să fie identic cu numărul de linii din `B`. `C` va avea același număr de linii ca `A` și același număr de coloane ca `B`.
^	Ridicare la putere	`C = A^n`	`A` trebuie să fie matrice pătratică. `n` este un scalar.
.'	Transpusa	`C = A.'`	Transpusa lui `A`.
'	Transpus conjugata	`C = A'`	Transpus conjugata lui `A`.

Observație: operațiile dintre vectori și matrice se rezumă la a considera că vectorii sunt cazuri particulare de matrice.

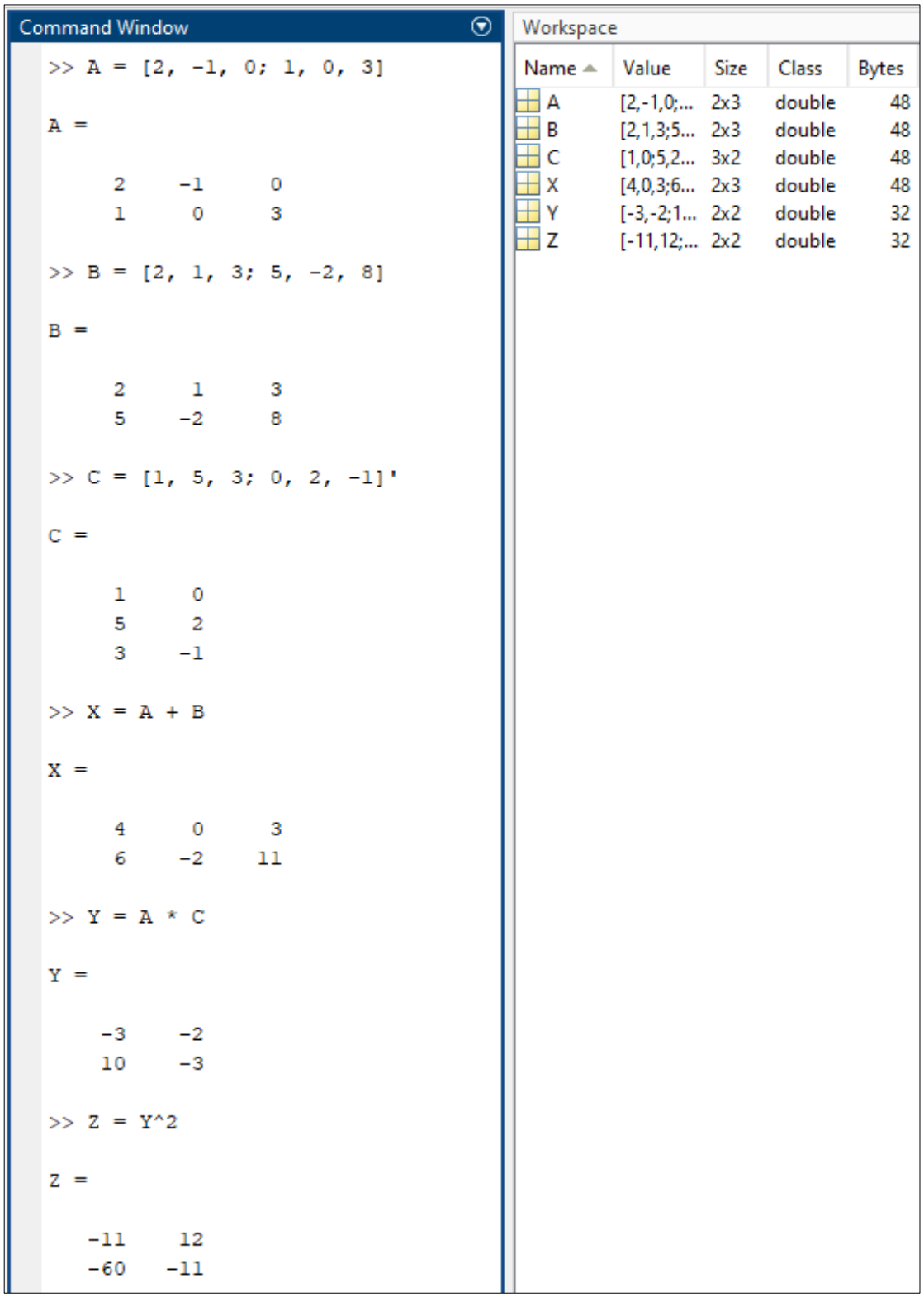


Figura 2.16. Exemple de operații cu matrice

2.7. Caracterul special *punct* (.)

Caracterul special *punct* se poate folosi în Matlab în 4 contexte complet diferite:

a) Pentru a scrie numere zecimale (s-a discutat în *Capitolul 1*)

b) Pentru a realiza transpusa unei matrice (s-a discutat la *Capitol 2.5*)

c) Pentru a realiza operații matriceale element cu element

d) Pentru tipul de date `struct` (a se vedea *Capitolul 7*)

În acest subcapitol se va prezenta efectul folosirii caracterului special *punct* atunci când se lucrează cu vectori și matrice.

Fie `A` și `B` doi vectori (sau matrice). Folosind caracterul special *punct* se pot realiza următoarele operații:

Simbol	Rol	Sintaxă	Observații
`.*`	Înmulțire element cu element	`C = A .* B`	`A` și `B` să aibă aceleași dimensiuni (același număr de linii și de coloane) `A(i,j)` se înmulțește cu `B(i,j)`.
`./`	Împărțire element cu element	`C = A ./ B`	`A` și `B` să aibă aceleași dimensiuni (același număr de linii și de coloane) `A(i,j)` se împarte la `B(i,j)`.
`.^`	Ridicare la putere element cu element	`C = A.^n`	`A` trebuie să fie matrice pătratică. Fiecare element din `A` se ridică la puterea `n`. `n` este un scalar.

Dacă operațiile de înmulțire (*), împărțire (/), ridicare la putere (^), dintre două matrice, sunt precedate de caracterul special ***punct***, atunci acele operații se realizează element cu element, fără a mai respecta regulile algebrei liniare.

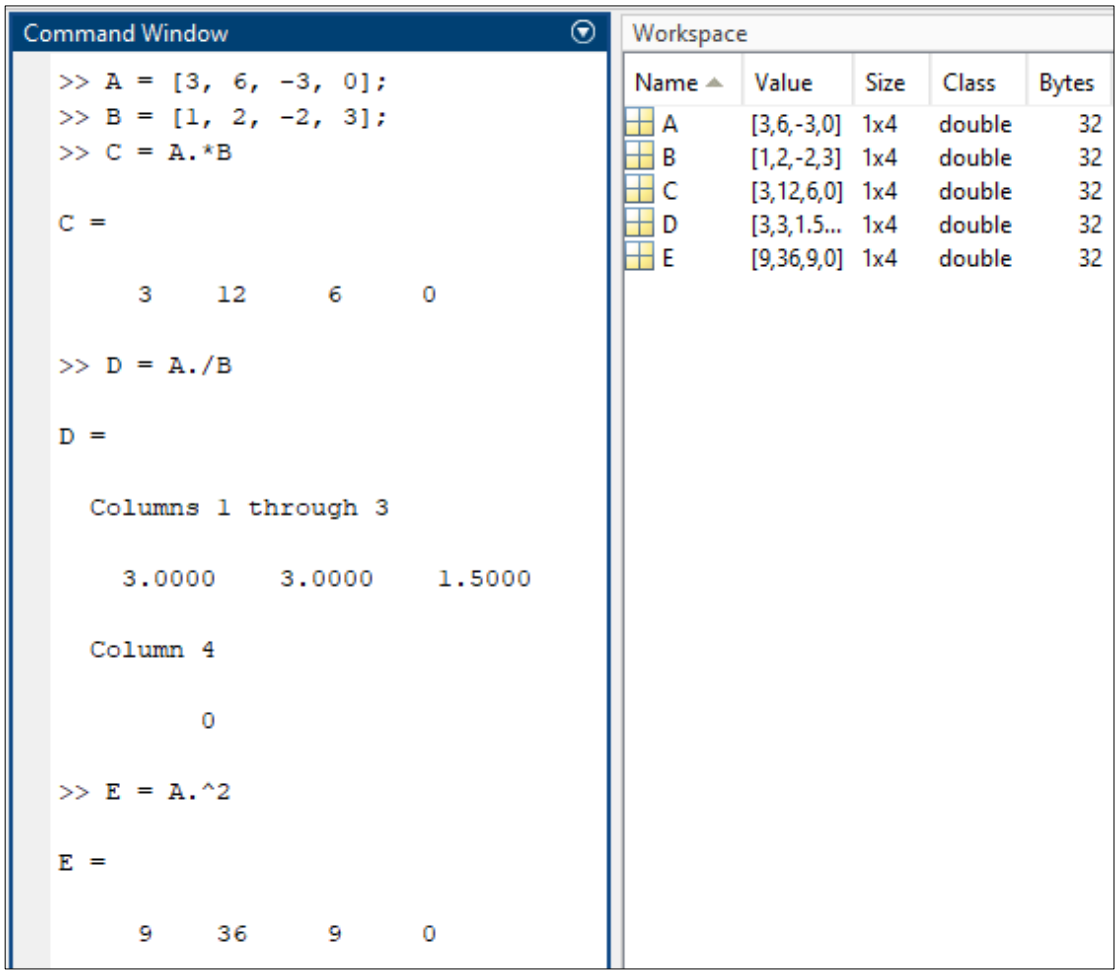


Figura 2.17. Exemple de operații matematice precedate de caracterul special *punct*

2.8. 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**

Sintaxă: `C = [A, B]`

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

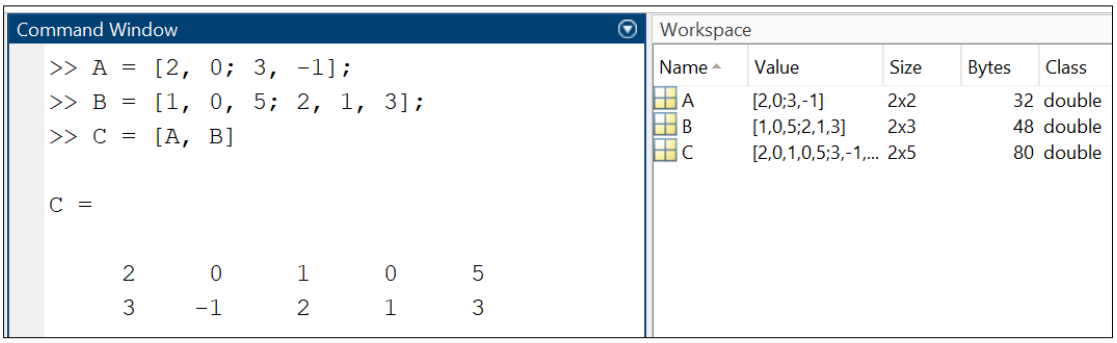


Figura 2.18. Exemplu de concatenare pe orizontală

- **Concatenarea pe verticală**

Sintaxă: `C = [A; B]`

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

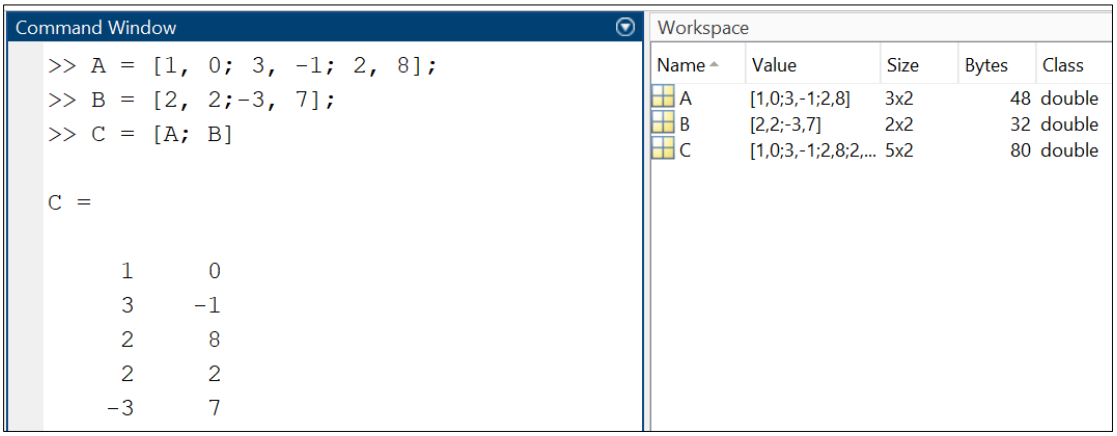


Figura 2.19. Exemplu de concatenare pe verticală

2.9. Șiruri de caractere în Matlab

Pentru variabile care să conțină text, există două tipuri de date: `char` și `string`.

- Un text stocat într-o variabilă de tip `char` este un vector de caractere. O variabilă de tip `char` se scrie între **apostroafe (' ')**.
- Un text stocat într-o variabilă de tip `string` este un scalar de caractere. O variabilă de tip `string` se scrie între **ghilimele (" ")**.

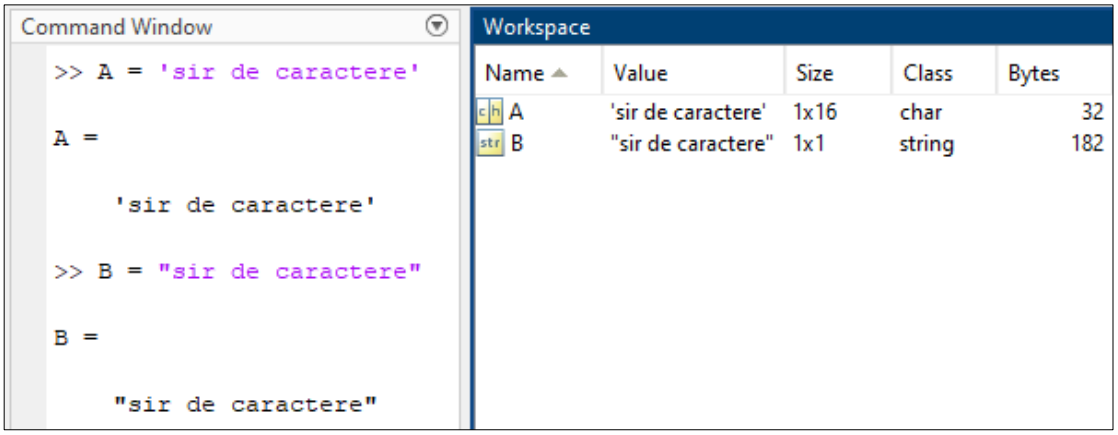


Figura 2.20. Tipurile de date `char` și `string`

Observații:

- Dacă șirul de caractere nu trebuie ulterior procesat, atunci nu contează dacă îl scriem între apostroafe (de tip `char`) sau între ghilimele (de tip `string`).
- Dacă vrem însă să lucrăm cu acele șiruri de caractere, trebuie să avem însă grijă la tipul de date (`char` sau `string`) deoarece de cele mai multe ori sunt diferențe.

Concatenarea șirurilor de caractere

- Dacă se dorește concatenarea a 2 sau mai multe șiruri de caractere stocate în variabile de tip `char`, atunci se poate folosi sintaxa de concatenare de la vectori: `C = [A, B]`
- Dacă se dorește concatenarea a 2 sau mai multe șiruri de caractere stocate în variabile de tip `string`, atunci se folosește sintaxa: `Z = A + B`

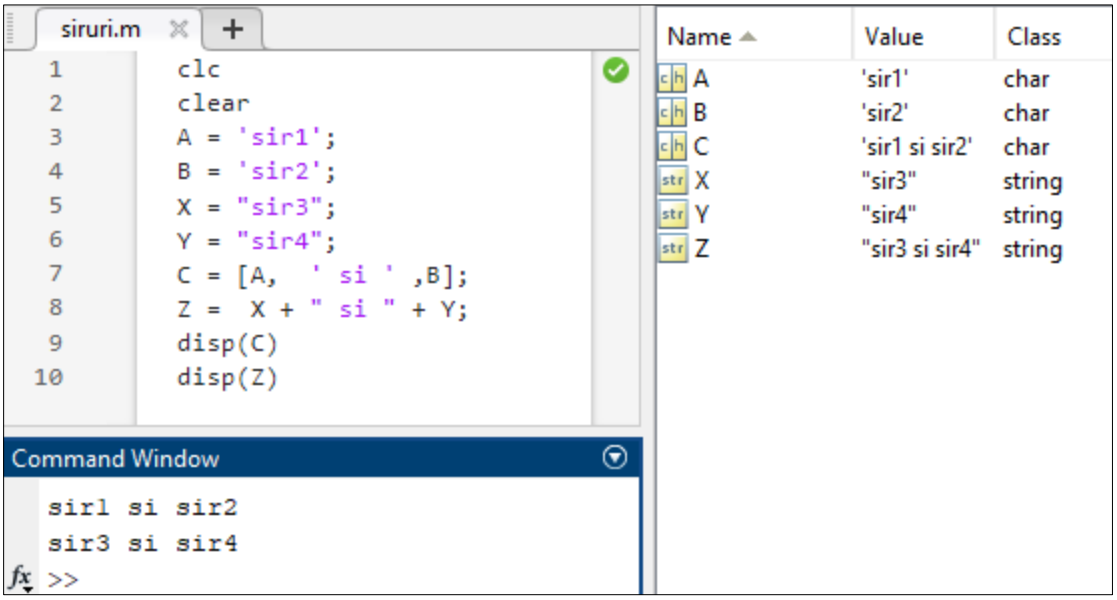


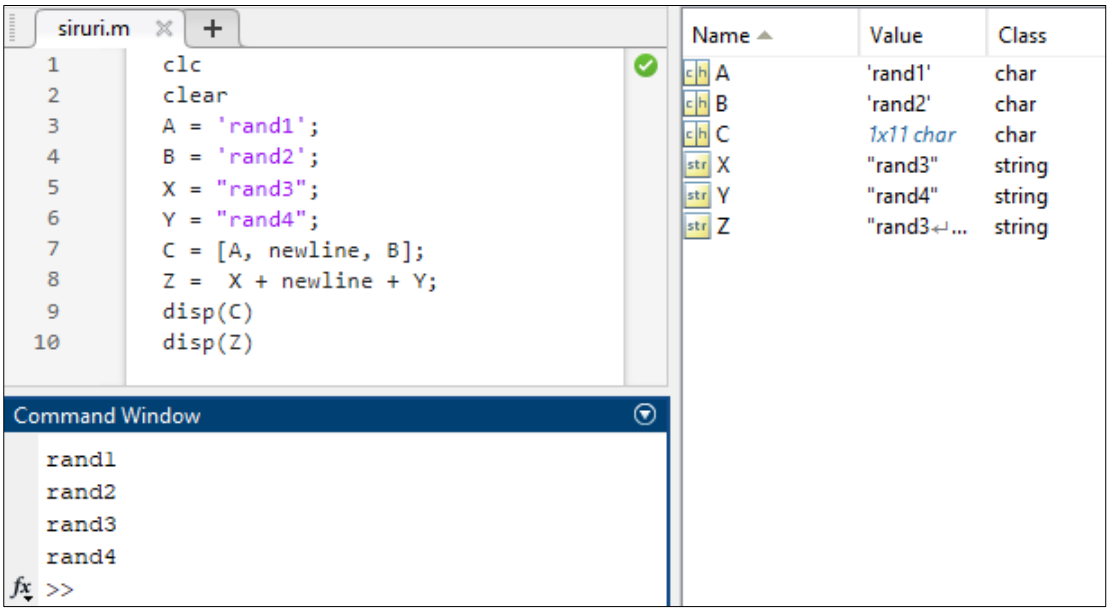
Figura 2.21. Exemple de concatenare pentru `char` și `string`

Observație: concatenarea șirurilor de caractere cu operatorul "+" este valabilă doar pentru tipul de date `string`.

Adăugarea unei noi linii de text

Pentru a scrie un text pe mai multe linii se poate folosi comanda `newline`.

Aceasta funcționează atât pentru date de tip `char` cât și `string`.



Fgura 2.22. Adăugarea unei noi linii de text

2.10. Aplicații

Aplicația 1. Să se scrie într-un fișier script Matlab vectorul:

$$X = [1, 0, 10, -5, 2]$$

a) Să se salveze în variabila `y` al 3-lea element din `X`.

b) Toate elementele vectorului `X` să fie adunate cu valoarea 5 iar rezultatul să fie salvat în variabila `Y`.

c) Să se ridice la pătrat fiecare element din `X` iar rezultatul să se salveze în variabila `Z`.

d) Prin două metode distincte să se transforme vectorul linie `X` în vectorul coloană `W`.

e) Să se salveze în variabila `p`, produsul dintre `X` și X^T

f) Să se salveze în variabila `Q`, produsul dintre X^T și `X`.

g) Să se înlocuiască al 3-lea element din `X` cu valoarea lui la puterea a 2-a.

h) Să se elimine elementul de pe a 3-a poziție din `X`.

Aplicația 2. Să se scrie într-un fișier script Matlab matricea:

$$A = \begin{bmatrix} 1 & 0 & -5 \\ 2 & 8 & 3 \end{bmatrix}$$

a) Să se salveze în variabila `a` elementul din matricea `A` de pe linia 2 și coloana 3.

b) Toate elementele din matricea `A` să se înmulțească cu valoarea 2 și rezultatul să se salveze în variabila `B`.

c) Să se ridice la pătrat fiecare element din matricea `A` iar rezultatul să se salveze în variabila `C`.

d) Să se salveze în vectorul `D` toate elementele din matricea `A`, citite pe coloane.

e) Să se înlocuiască elementul din matricea `A` de pe linia 1 și coloana 3 cu radicalul valorii existente pe poziția respectivă.

Aplicația 3. Să se scrie într-un fișier script Matlab următoarea matrice:

$$A = \begin{bmatrix} 1 & 0 & 5 & 2 \\ -1 & 3 & 4 & 0 \\ 2 & 3 & 1 & -3 \end{bmatrix}$$

a) să se salveze în vectorul `B` linia 1 din matricea `A`.

b) să se salveze în vectorul `C` coloana 3 din matricea `A`.

c) să se salveze în matricea `D` elementele din `A` de pe liniile 1, 2 și coloanele 2, 3, 4.

d) să se înmulțească matricea `A` cu transpusa ei și să se salveze rezultatul în variabila `E`.

e) să se concateneze pe verticală matricea `A` cu vectorul `B`.

f) să se concateneze pe orizontală matricea `A` cu vectorul `C`.

g) să se elimine linia 2 din matricea `A`.

Aplicația 4. Să se genereze:

a) un vector `X1` conținând elementele: -10, -9, -8, ..., -1, 0, 1, ..., 8, 9, 10

b) un vector `X2` conținând elementele: 3, 6, 9, 12, ... , 30

c) un vector `X3` conținând elementele: 10, 9, 8, ... , 1, 0

d) un vector `X4` conținând elementele: 1, 4, 9, 16, 25, 36, ... ,100

e) un vector `X5` conținând elementele:

$$\frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \dots, \frac{10}{2}$$

f) un vector `X6` conținând elementele:

$$\frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \dots, \frac{2}{10}$$

g) un vector `X6` conținând elementele:

$$\frac{4}{1}, \frac{4}{5}, \frac{4}{9}, \dots, \frac{4}{77}$$

Capitolul 3
Funcții uzuale pentru lucrul cu vectori, matrice și șiruri de caractere

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: 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, aflarea dimensiunii unei matrice etc.

3.1. Generarea diverselor matrice particulare

Funcție Matlab	Sintaxă Matlab	Explicații
zeros	`Z = zeros(m, n)`	Generează o matrice `Z` de dimensiune `m x n` conținând doar valoarea 0.
ones	`O = ones(m, n)`	Generează o matrice `O` de dimensiune `m x n` conținând doar valoarea 1.
eye	`I = eye(m)`	Generează matricea unitate `I` de dimensiune `m x m`.
rand	`R = rand(m, n)`	Generează o matrice `R` de dimensiune `m x n` cu **valori reale** pseudorandom, uniform distribuite în intervalul (0, 1).
randi	`R = randi([val_min,val_max],m,n)`	Generează o matrice `R` de dimensiuni `m x n` cu **valori întregi** pseudorandom, distribuite uniform într-un interval dat: `[val_min,val_max]`.

Observație: pentru funcțiile `zeros`, `ones`, `rand`, `randi`, dacă se specifică un singur parametru `n` la dimensiune, atunci se va genera o matrice pătratică de ordin `n`.

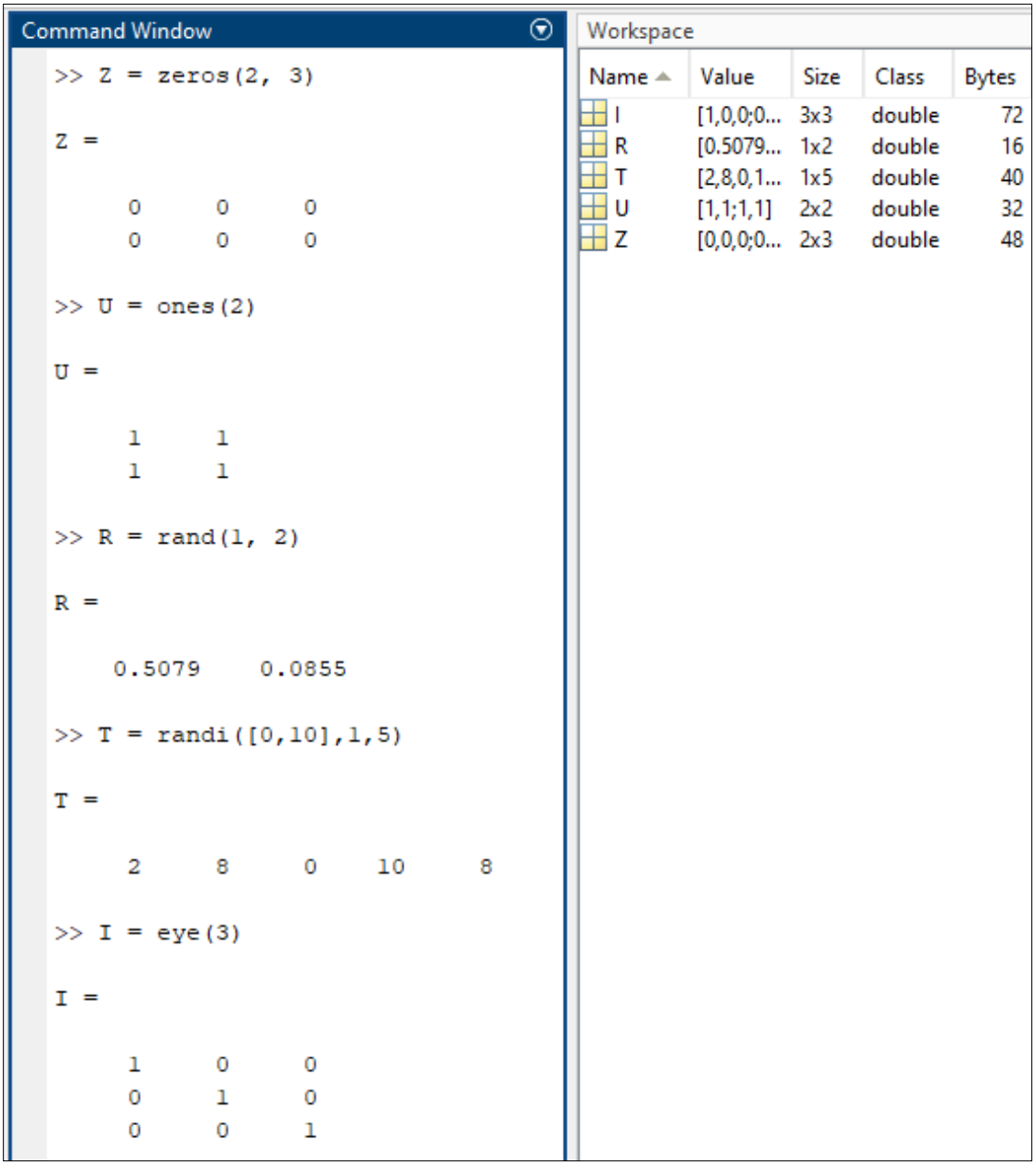


Figura 3.1. Generarea diverselor matrice particulare

3.2. Determinarea dimensiunii unui vector/matrice

Prin dimensiune se înțelege numărul de linii, numărul de coloane (eventual numărul de straturi). Pentru a determina dimensiunea unui vector sau a unei matrice `X` se poate folosi funcția `size`.

Sintaxă: `[M,N] = size(X)`

- în `M` se va salva numărul de linii din `X`
- în `N` se va salva numărul de coloane din `X`

Observație 1: Sintaxa de mai sus a funcției `size` funcționează pentru vectori și matrice. Dacă `X` este o matrice multistrat (de exemplu o imagine color), atunci sintaxa trebuie să fie:

```
[M,N,P] = size(X)
```

- în `M` se va salva numărul de linii din `X`
- în `N` se va salva numărul de coloane din `X`
- în `P` se va salva numărul de straturi

altfel, dacă se întorc doar parametrii `M` și `N`, `N` va conține produsul dintre *numărul de coloane* si *numărul de straturi*

Observație 2: Funcția `size` mai poate fi folosită cu un parametru suplimentar care să specifice dimensiunea pe care să se determine numărul de elemente.

Sintaxă: `val = size(X, DIM)`

- dacă `DIM` = 1, în `val` se va salva numărul de linii din `X`
- dacă `DIM` = 2, în `val` se va salva numărul de coloane din `X`
- dacă `DIM` = 3, în `val` se va salva numărul de straturi din `X`

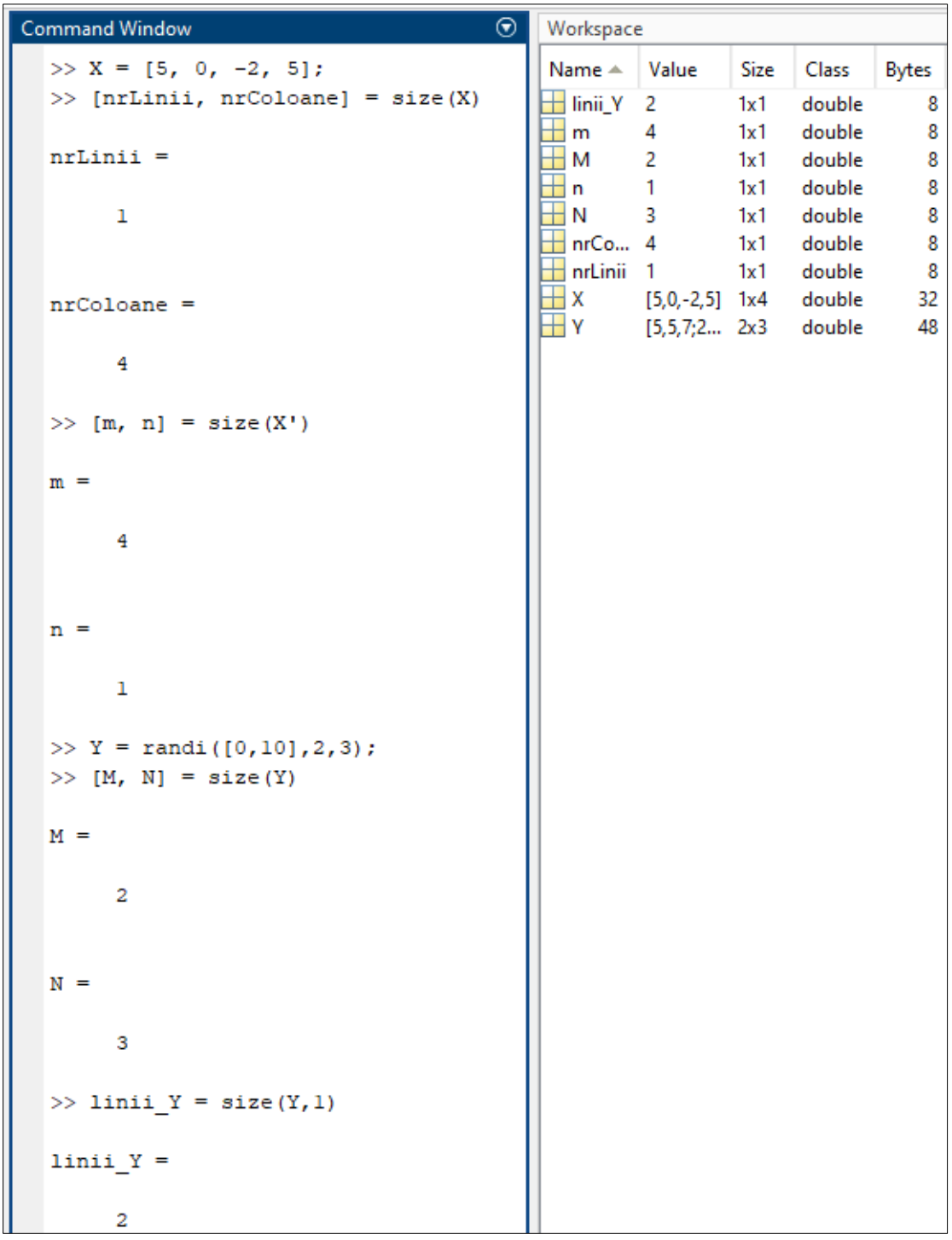


Figura 3.2. Exemple de folosire pentru funcția `size`

3.3. Determinarea numărului de elemente dintr-un vector/matrice

- **Pentru un vector `X`** se pot folosi funcțiile `length` sau `numel` pentru a determina numărul de elemente.

Observație: rezultatul este echivalent cu `size(X,1)` (dacă `X` este vector coloană) sau `size(X,2)` (dacă `X` este vector linie).

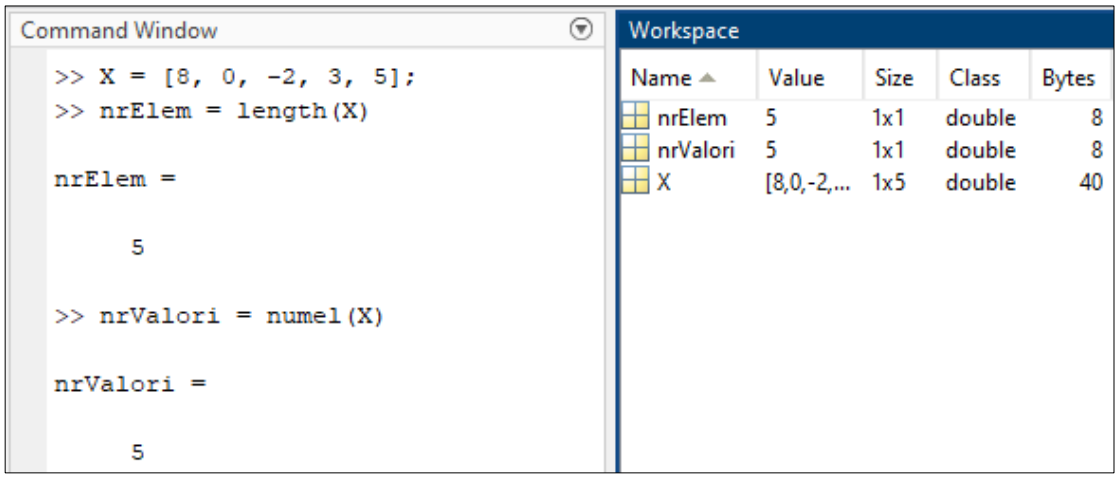


Figura 3.3. Determinarea numărului de elemente dintr-un vector

- **Pentru o matrice `A`** se poate folosi funcția `numel` pentru a determina numărul de elemente.

Observație: rezultatul este echivalent cu `size(A,1) x size(A,2)`

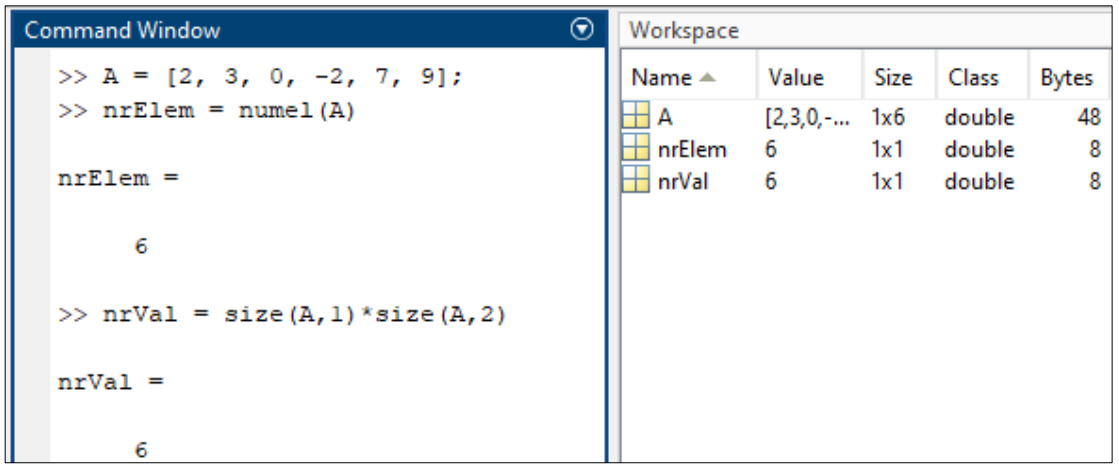


Figura 3.4. Determinarea numărului de elemente dintr-o matrice

3.4. Funcții Matlab care realizează operații matematice asupra vectorilor

În tabelul de mai jos sunt trecute cele mai uzuale funcții din Matlab care realizează operații matematice asupra **vectorilor**.

Funcție Matlab	Sintaxă Matlab	Descriere
sum	`s = sum(X)`	Se salvează în `s` suma tuturor elementelor din vectorul `X`.
prod	`p = prod(X)`	Se salvează în `p` produsul tuturor elementelor din vectorul `X`.
mean	`m = mean(X)`	Se salvează în `m` media tuturor elementelor din vectorul `X`.
min	`val = min(X)` `[val, poz] = min(X)`	Se salvează în `val` valoarea minimă din `X` Se salvează în: • `val`, valoarea minimă din `X` • `poz`, poziția pe care se găsește valoarea minimă din `X`
max	`val = max(X)` `[val, poz] = max(X)`	Se salvează în `val` valoarea maximă din `X` Se salvează în: • `val`, valoarea maximă din `X` • `poz`, poziția pe care se găsește valoarea maximă din `X`
find	`poz = find(condiție)`	Se salvează în `poz` indicii elementelor care respectă condiția dintre paranteze

Observație: dacă un vector conține valoarea minimă sau maximă de mai multe ori, atunci când se folosesc funcțiile `min` și `max` care să returneze și valorile pozițiilor elementelor de minim respectiv maxim, se va returna doar poziția primului element găsit. Dacă se dorește găsirea tuturor indicilor elementelor se folosește funcția `find`.

Fie un vector linie `X` cu 4 elemente. Să se determine suma, produsul, valoarea minimă, valoarea maximă și media elementelor din `X`.

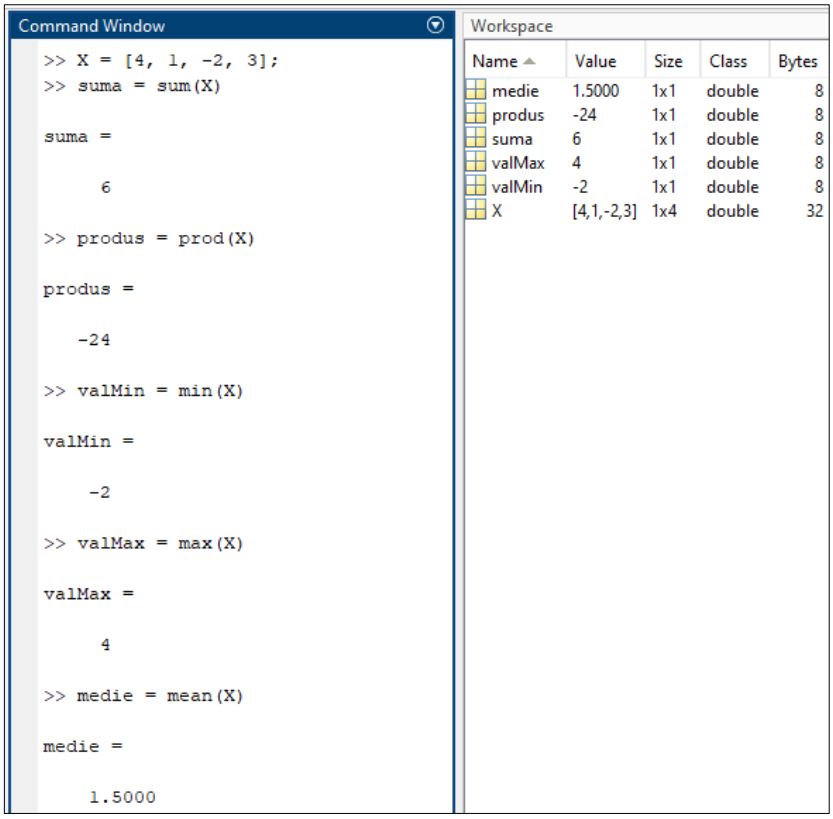


Figura 3.5. Exemple de funcții Matlab aplicate vectorilor

Fie un vector `X` cu 5 valori reale în intervalul (0, 1). Să se găsească valoarea minimă din `X` și indexul acesteia.

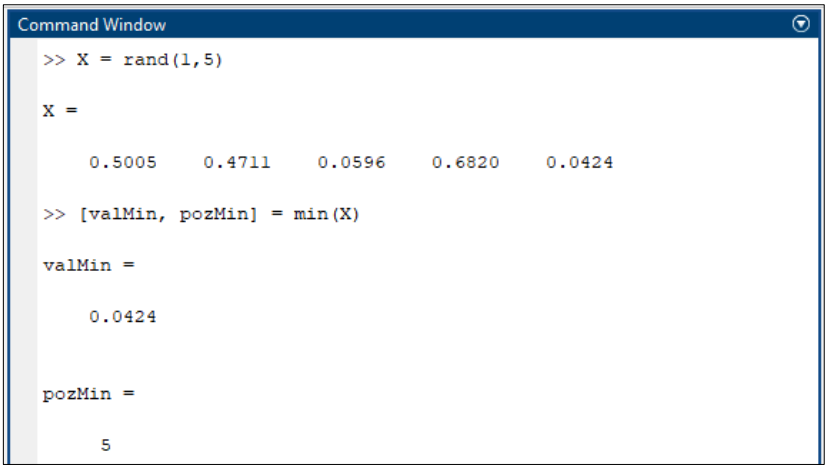


Figura 3.6. Exemplu de folosire al funcției `min`

Fie un vector `X` cu 10 valori întregi generate pseudorandom în intervalul [0, 5]. Să se găsească indicii tuturor elementelor mai mari decât 2.

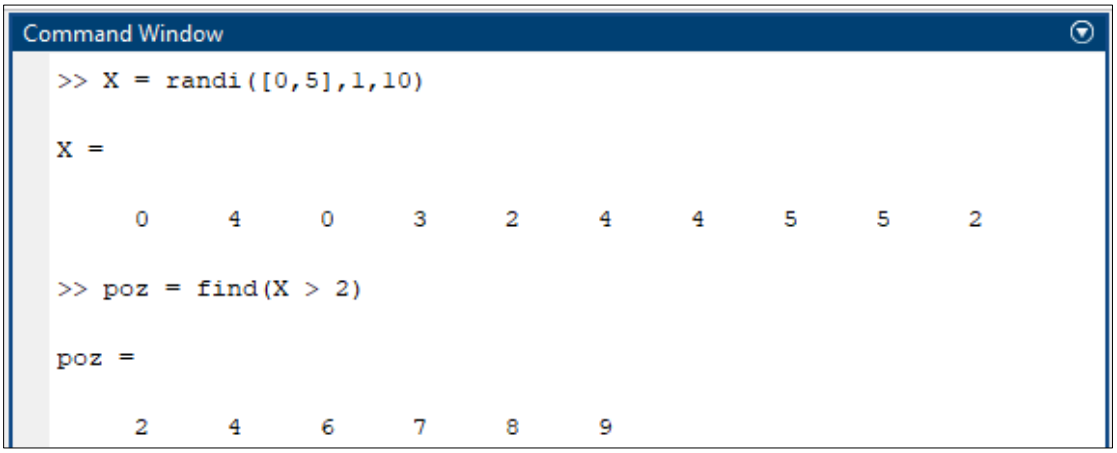


Figura 3.7. Exemplu de folosire a funcției `find`

Fie un vector `X` cu 10 valori întregi generate pseudorandom în intervalul [0, 5]. Să se găsească indicii tuturor elementelor cu valoarea maximă din `X`.

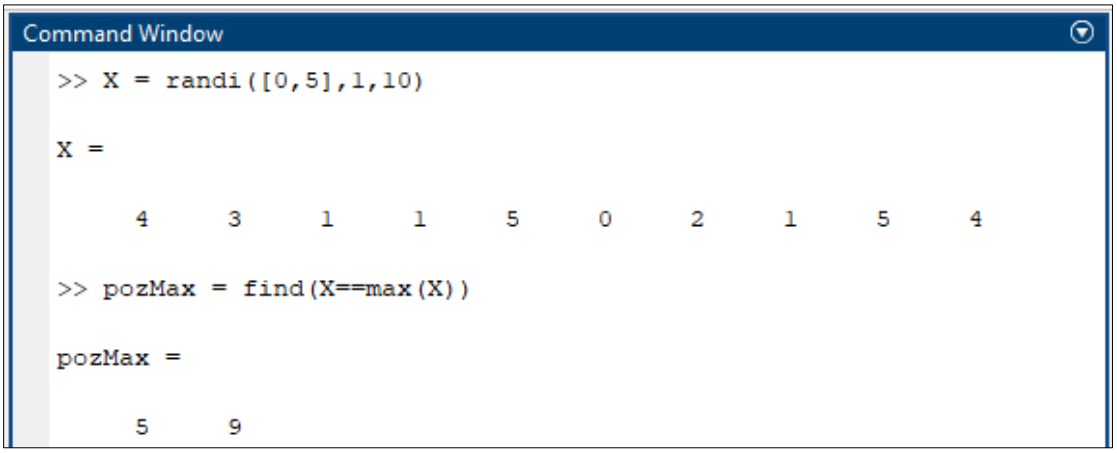


Figura 3.8. Exemplu de folosire a funcției `find`

3.5. Funcții Matlab care realizează operații matematice asupra matricelor

În tabelul de mai jos sunt trecute cele mai uzuale funcții din Matlab care realizează operații matematice asupra **matricelor**.

Funcție Matlab	Sintaxă Matlab	Descriere
sum	s1 = sum(A) s1 = sum(A,1) s2 = sum(A,2) s = sum(A,'all')	suma pe coloane suma pe coloane suma pe linii suma **tuturor** elementelor din A
prod	p1 = prod(A) p1 = prod(A,1) p2 = prod(A,2) p = prod(A,'all')	produsul pe coloane produsul pe coloane produsul pe linii produsul **tuturor** elementelor din A
mean	m1 = mean(A) m1 = mean(A,1) m2 = mean(A,2) m = mean(A, 'all')	media pe coloane media pe coloane media pe linii media **tuturor** elementelor din A
min	valMin1 = min(A, [], 1) valMin2 = min(A, [], 2) valMin = min(A, [], 'all')	minimul pe fiecare coloană din A minimul pe fiecare linie din A minimul **tuturor** elementelor din A
max	valMax1 = max(A, [], 1) valMax2 = max(A, [], 2) valMax = max(A, [], 'all')	maximul pe fiecare coloană din A maximul pe fiecare linie din A maximul **tuturor** elementelor din A
find	[L, C] = find(conditie)	întoarce linia și coloana elementelor care respectă condiția dintre paranteze

Fie o matrice A_{2x3}. Să se determine suma elementelor pe fiecare coloană, suma elementelor pe fiecare linie, suma tuturor elementelor, valoarea minimă și valoarea maximă din A.

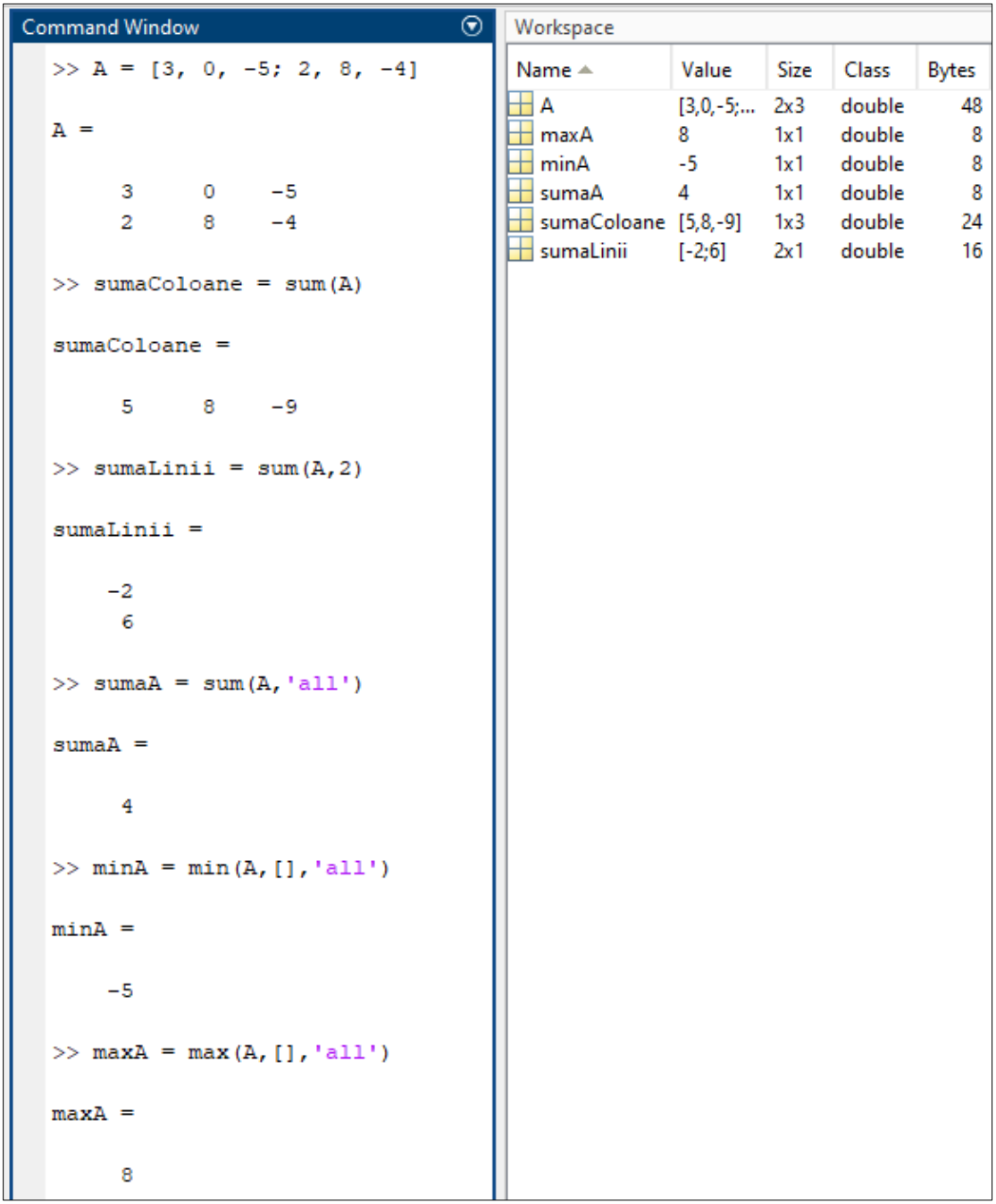


Figura 3.9. Exemple de funcții Matlab aplicate vectorilor

3.6. SINTEZĂ: Funcții Matlab uzuale pentru lucrul cu vectori și matrice

Operație	Vector `x`	Matrice bidimensională `A`
Dimensiune	`[M,N] = size(X)`	`[M,N] = size(A)`
Număr de elemente	`nrElem = length(X)` `nrElem = numel(X)`	`nrElem = size(A,1)*size(A,2)` `nrElem = numel(A)`

Suma tuturor elementelor	`s = sum(X)`	`s = sum(sum(A))`
		`s = sum(A, 'all')`
		`s = sum(A(:))`

Media tuturor elementelor	`media = mean(X)`	`media = mean(mean(A))`
		`media = mean(A,'all')`
		`media = mean(A(:))`

Produsul tuturor elementelor	`produs = prod(X)`	`produs = prod(prod(A))`
		`produs = prod(A,'all')`
		`produs = prod(A(:))`

Valoarea minimă a tuturor elementelor	`valMin = min(X)`	`valMin = min(min(A))`
		`valMin = min(A,[],'all')`
		`valMin = min(A(:))`

Valoarea maximă a tuturor elementelor	`valMax = max(X)`	`valMax = max(max(A))`
		`valMax = max(A,[],'all')`
		`valMax = max(A(:))`

Găsirea indicilor elementelor care respectă o condiție	`poz = find(conditie)`	`[L, C] = find(conditie)`

3.7. Sortarea elementelor dintr-un vector

Sintaxă: `Xsortat = sort(X,tipSortare)`

- dacă `tipSortare = 'ascend'` atunci ordonarea elementelor din vectorul `X` se face crescător. Implicit tipul de sortare este `'ascend'`.
- dacă `tipSortare = 'descend'` atunci ordonarea elementelor din vectorul `X` se face descrescător.

Observație: funcția `sort` mai poate întoarce pe lângă vectorul cu elementele sortate `Xsortat` și un vector al indicilor are specifică modul în care elementele lui `X` au fost rearanjate pentru a obține rezultatul sortat `Xsortat`.

Sintaxă: `[Xsortat, Indici] = sort(X,tipSortare)`

Fie un vector `X` cu 5 valori reale în intervalul (0, 1). Să se salveze în `Y` elementele din `X` în ordine crescătoare. Să se salveze în variabila `pozMax` poziția pe care se află cel mai mare element din `X`. Să se salveze în `valMax` valoarea maximă din `X` (în trei moduri diferite).

```
Command Window
>> X = rand(1,5)

X =

    0.3127    0.1615    0.1788    0.4229    0.0942

>> [Y, pozitii] = sort(X)

Y =

    0.0942    0.1615    0.1788    0.3127    0.4229


pozitii =

     5     2     3     1     4

>> pozMax = pozitii(end)

pozMax =

     4

>> valMax = X(pozMax)

valMax =

    0.4229

>> valMax = Y(end)

valMax =

    0.4229

>> valMax = max(X)

valMax =

    0.4229
```

3.8. Funcții Matlab care realizează operații cu matrice specifice algebrei liniare

Funcție Matlab	Sintaxă Matlab	Descriere
`det`	`d = det(A)`	în `d` se salvează determinantul matricei `A`.
`inv`	`A_inv = inv(A)`	în `A_inv` se salvează inversa matricei `A`.
	`C = A.'`	în `C` se salvează transpusa matricei `A`.
	`C = A'`	în `C` se salvează transpus conjugata matricei `A`.
`diag`	`D = diag(A)`	`D` va fi un vector coloană cu elementele de pe diagonala principală a matricei `A`.

Fie o matrice A_{3x3} cu valori numere întregi în intervalul [-5, 5]. Să se calculeze determinantul, inversa și transpusa. Să se salveze în variabila `rez`, raportul dintre suma elementelor de pe diagonala principală și suma tuturor elementelor din `A`.

```
Command Window
>> A = randi([-5, 5],3)

A =

     0    -4    -4
     2    -4    -5
     2     5     1

>> detA = det(A)

detA =

  -24.0000

>> invA = inv(A)

invA =

   -0.8750    0.6667   -0.1667
    0.5000   -0.3333    0.3333
   -0.7500    0.3333   -0.3333

>> transA = A.'

transA =

     0     2     2
    -4    -4     5
    -4    -5     1

>> rez = sum(diag(A))/sum(A,'all')

rez =

    0.4286
```

3.9. Funcții Matlab care realizează operații cu șiruri de caractere

- **Conversia unei variabile de tip numeric în șir de caractere**

Sintaxă: `num2str(numar)`

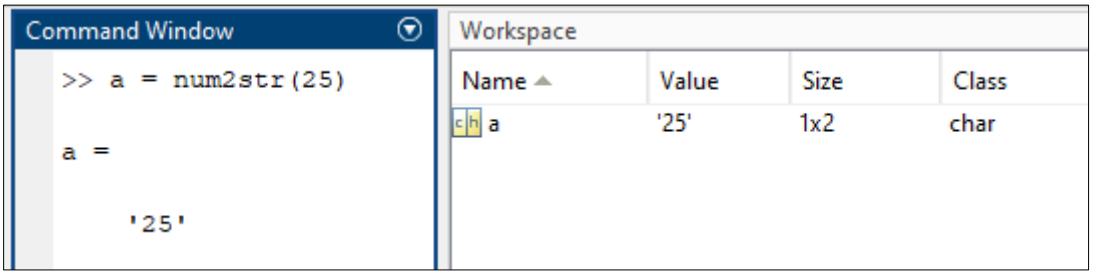


Figura 3.10. Exemplu de conversie a unei valori numerice în șir de caractere

- **Conversia unei variabile de tip șir de caractere în variabilă numerică**

Sintaxă: `str2num('valNumerica')`

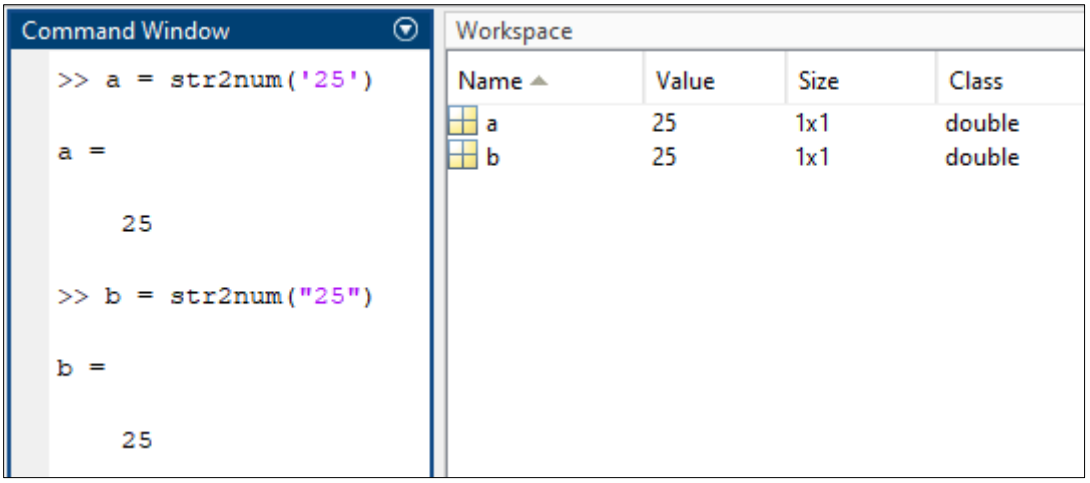


Figura 3.11. Exemplu de conversie a unui șir de caractere în variabilă de tip numeric

- **Concatenare dintre un șir de caractere și o valoare numerică**

În funcție de tipul de date al șirului de caractere, concatenare se va realiza astfel:

- Concatenare între variabilă `A` de tip `char` și variabilă `B` numerică

  Sintaxă: `C = [A , num2str(B)]`

- Concatenare între variabilă `A` de tip `string` și variabilă `B` numerică

  Sintaxă: `C = A + B`

concatSirCaractcuValNum.m

```
1   clc
2   clear
3   close all
4   A = [1, 5, 3, 9];
5   medie = mean(A);
6   % concatenare intre variabila de tip char si variabila numerica
7   disp(['Media elementelor din vectorul A este = ',num2str(medie)])
8   % concatenare intre variabila de tip string si variabila numerica
9   disp("Media elementelor din vectorul A este = " + medie)
```

Command Window

```
Media elementelor din vectorul A este = 4.5
Media elementelor din vectorul A este = 4.5
```

Figura 3.12. Concatenare dintre un șir de caractere și o valoare numerică

- **Conversia din `char` în `string`**

Sintaxă: `convertCharsToStrings('sir de caractere')`

Command Window

```
>> a = 'sir';
>> A = convertCharsToStrings(a);
>>
```

Workspace

Name	Value	Size	Class
a	'sir'	1x3	char
A	"sir"	1x1	string

Figura 3.13. Exemplu de conversie din `char` în `string`

- **Conversia din `string` în `char`**

Sintaxă: `convertStringsToChars('sir de caractere')`

Command Window

```
>> b = "sir de caractere";
>> B = convertStringsToChars(b);
>>
```

Workspace

Name	Value	Size	Class
b	"sir de c...	1x1	string
B	'sir de c...	1x16	char

Figura 3.14. Exemplu de conversie din `string` în `char`

Observație: mai multe funcții pentru lucrul cu șiruri de caractere găsiți aici:

https://www.mathworks.com/help/matlab/characters-and-strings.html

3.10. Aplicații

Aplicația 1. Fie un vector linie **X**, conținând 5 valori reale generate pseudorandom, cu distribuție uniformă în intervalul (0,1).

a) Să se salveze în variabila **rangeX** diferența dintre cea mai mare valoare din X și cea mai mică valoare din X.

b) Să se salveze în vectorul **putereX** toate elementele din X ridicate la puterea a 3-a.

c) Să se salveze în vectorul **sortX** elementele din X ordonate descrescător.

d) Să se salveze în vectorul **primele2valori** cele mai mari 2 elemente din X.

e) Să se salveze în vectorul **valoriMari** elementele din X mai mari decât 0.5.

f) Să se salveze în variabila **nrElem** numărul elementelor din X mai mari decât 0.5.

g) Să se salveze în variabila **medie** media tuturor elementelor din X mai mari de 0.5.

Aplicația 2. Fie un vector coloană **Y** cu N = 11 elemente numere naturale generate pseudorandom în intervalul [0, 10].

a) Să se salveze în variabila **valMedie** media vecinilor elementului de pe poziția centrală din Y.

b) Între elementele de pe pozițiile 2 și 3 din Y să se insereze valoarea 100. Rezultatul să se salveze în vectorul **Yplus1**.

c) Între elementele de pe pozițiile 2 și 3 din Y să se insereze 20 de valori de 100. Rezultatul să se salveze în vectorul **Z**.

Aplicația 3. Generați o matrice A cu 3 linii și 3 coloane cu valori întregi seudorandom în intervalul [-10, 10].

a) Calculați în variabila **sumA** suma tuturor elementelor din A.

b) Calculați în variabila **medieA** media tuturor elementelor din A.

c) Calculați în variabila **sumDiag** suma elementelor de pe diagonala principală din A.

d) Calculați în variabila **minA** valoarea minimă din A.

Aplicația 4. Fie un vector linie `X` ce conține N = 100 de numere reale generate pseudorandom cu valori în intervalul (0, 1). Să se calculeze deviația standard a valorilor din `X` utilizând formula:

$$y = \sqrt{\frac{1}{N-1} \cdot \sum_{i=1}^{N} (X_i - m)^2}$$

unde: m reprezintă media elementelor din X, N reprezintă numărul elementelor din X

- Să se salveze în variabila `mesaj` textul „*Valoarea deviației standard este:* ” urmată de valoarea lui y calculată cu formula de mai sus. *Exemplu*: dacă valoarea lui y este 0.3, atunci mesajul va fi: „*Valoarea deviației standard este:* 0.3”

Aplicația 5. Să se genereze un vector `X` conținând elementele

```
3, 6, 9, 12,..., 195, 198, 201
```

Să se calculeze în variabila `sumaX` suma elementelor din `X`.

Aplicația 6. Să se calculeze valoarea sumei:

$$z = \left(\frac{4}{1}\right) - \left(\frac{4}{3}\right) + \left(\frac{4}{5}\right) - \left(\frac{4}{7}\right) + \left(\frac{4}{9}\right) - \left(\frac{4}{11}\right) + \cdots + \left(\frac{4}{4001}\right) - \left(\frac{4}{4003}\right)$$

Observație: suma de mai sus este o aproximație a lui π.

Aplicația 7. Să se calculeze valoarea sumei:

$$y = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots + \frac{1}{1000^2}$$

Observație: suma de mai sus este o aproximație a lui $\frac{\pi^2}{6}$

Aplicația 8. Să se calculeze valoarea sumei:

$$y = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \frac{1}{8} + \cdots + \frac{1}{999} - \frac{1}{1000}$$

Observație: suma de mai sus este o aproximație a lui $\ln(2)$

Capitolul 4
Reprezentări grafice 2D folosind funcțiile `plot` și `stem`

În acest capitol se va descrie pas cu pasul modul de reprezentare a unui grafic într-o figură, a mai multor grafice în aceeași figură în același sistem de coordonate (`hold on ... hold off`) precum și reprezentarea mai multor grafice în aceeași figură în sisteme de coordonate diferite (`subplot`). Reprezentările grafice se vor realiza în spațiul 2D, folosind funcțiile `plot` și `stem`. Vor fi prezentate de asemenea principalele proprietăți ale acestor funcții.

4.1. Funcția `figure`

Pentru orice reprezentare grafică este nevoie de o fereastră de afișare care se deschide folosind funcția `figure`.

Sintaxă: `figure(nrFigură)`

Să se deschidă o fereastră cu numărul 3.

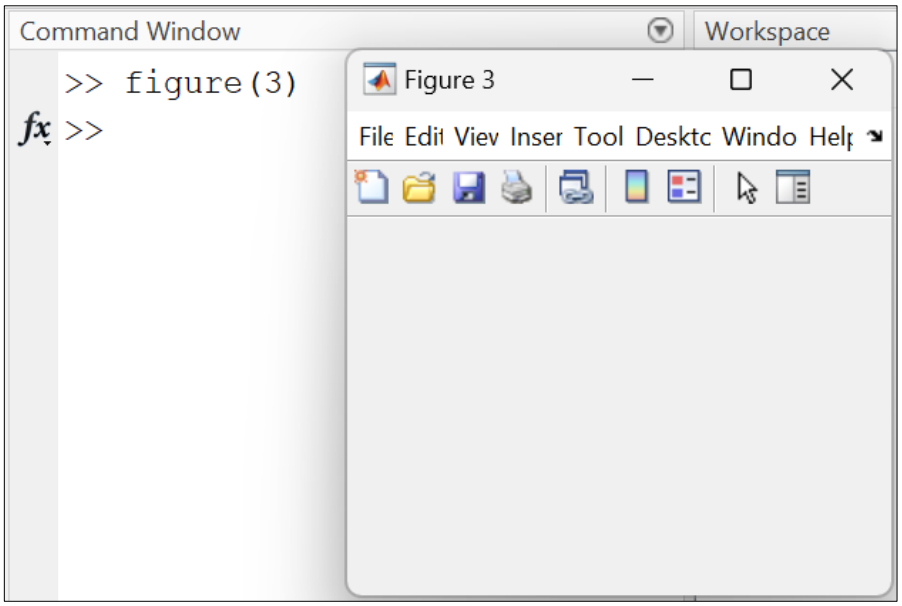


Figura 4.1. Deschiderea unei ferestre pentru reprezentare grafică

Observații:

- Dacă înainte de o reprezentare grafică nu se deschide o fereastră pentru afișare cu funcția `figure`, atunci reprezentarea grafică se va realiza în ultima fereastră deschisă. Dacă nu există nicio fereastră deschisă atunci se va deschide în mod implicit fereastra cu numărul 1;
- Dacă se apelează funcția `figure` fără să se specifice niciun număr între parantezele rotunde, atunci Matlab-ul va incrementa în mod automat numărul figurii. Cu alte cuvinte, sintaxa: `figure(), figure(), figure()` va deschide 3 ferestre cu numerele 1, 2 și 3;
- Dacă se dorește ca o fereastră să aibă pe lângă număr și un nume, atunci se folosește sintaxa:

```
figure('Name','NumeFigura')
```

  unde `NumeFigura` este numele dorit;

- Dacă se dorește ca o fereastră să aibă doar nume, fără număr, atunci se folosește sintaxa:

```
figure('Name','Nume figura','NumberTitle','off')
```

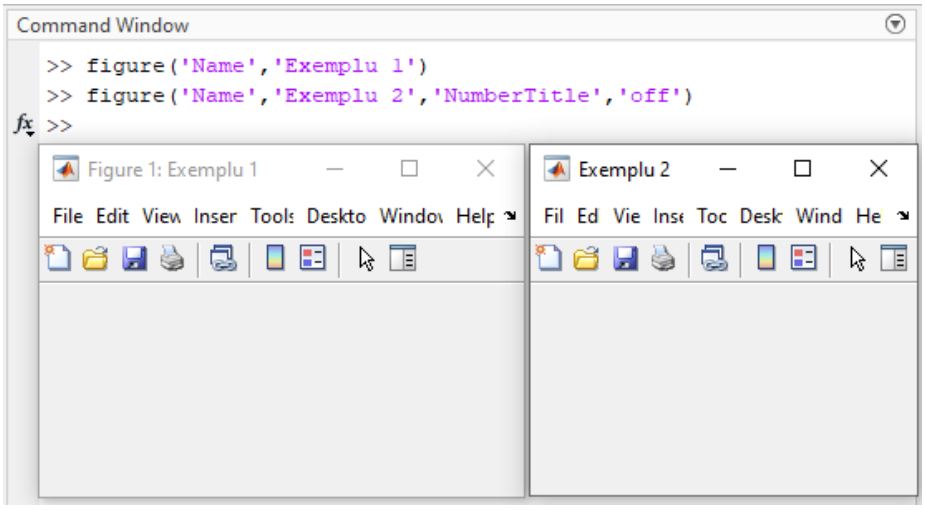


Figura 4.2. Deschiderea unei ferestre pentru reprezentare grafică care să aibă nume

Pentru a închide toate ferestrele deschise în Matlab se folosește comanda `close all`

4.2. 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.
- Vectorii `X` și `Y` trebuie să aibă aceleași dimensiuni.

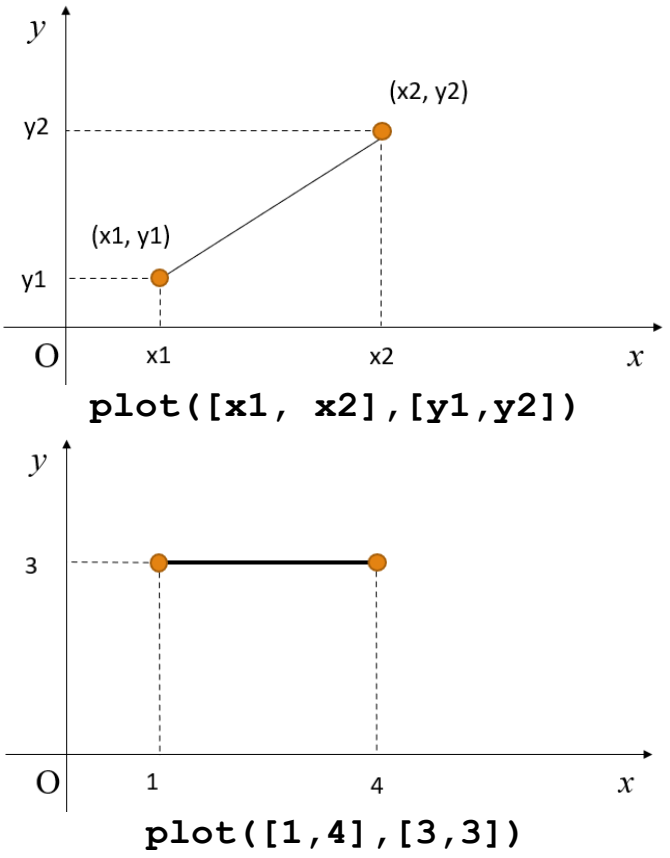


Figura 4.3. Exemplificare grafică a funcției `plot`

Observație: Dacă funcția `plot` se folosește cu un singur parametru, de exemplu `plot(Y)`, în acest caz se consideră că pe axa O*x* sunt indicii eșantioanelor (numerele 1, 2, 3, … , `n`, unde `n` reprezintă numărul de eșantioane din vectorul `Y`).

Fie vectorii `X = [4, 5, 7, 8]` și `Y = [-1, 3, 3, -1]`.

Să se reprezinte cu funcția `plot` graficul determinat de perechile de puncte `{X(i), Y(i)}`.

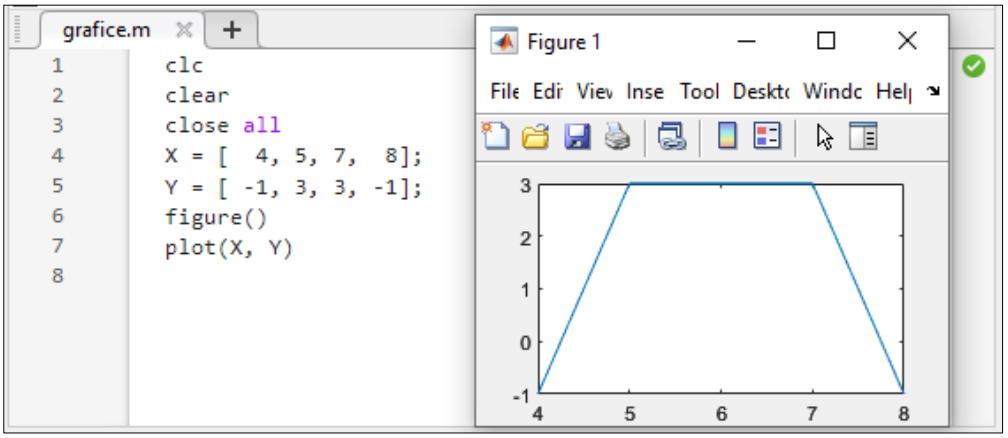


Figura 4.4. Exemplu de reprezentare grafică cu funcția `plot`

Când reprezentăm un grafic, este bine ca acesta să conțină:

- **Titlu** reprezentativ pentru graficul respectiv, funcția `title`
  Sintaxă: `title('titlul figurii')`
- **Semnificația axei O*x*,** funcția `xlabel`
  Sintaxă: `xlabel('semnificația axei Ox')`
  Exemplu: `xlabel('timp[s]')`
- **Semnificația axei O*y*,** funcția `ylabel`
  Sintaxă: `ylabel('semnificația axei Oy')`
  Exemplu: `ylabel('amplitudine[V]')`

Modificarea limitelor axelor Ox și Oy

- modificarea limitelor pentru axele O*x* și O*y*, funcția `axis`
  Sintaxă: `axis([xmin, xmax, ymin, ymax])`
- modificarea limitelor doar pentru axa O*x*, funcția `xlim`
  Sintaxă: `xlim([xmin, xmax])`
- modificarea limitelor doar pentru axa O*y*, funcția `ylim`
  Sintaxă: `ylim([ymin, ymax])`

unde `xmin, xmax, ymin, ymax`, reprezintă limitele minime respectiv maxime pentru axele Ox și Oy.

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 4.1. 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

Observații:

- Nu contează ordinea în care sunt scrise cele trei proprietăți (culori, tipul de linie, și markere). Pentru a trasa un grafic cu linie *punctată*, marker *pătrat* și culoare *magenta* se poate utiliza oricare dintre următoarele instrucțiuni:

```
plot(X,Y,':sm')
plot(X,Y,'s:m')
plot(X,Y,'m:s') etc
```

- Pentru o altă culoare în afara celor menționate în tabelul de 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])
```

- Pentru o mai bună vizualizare a valorilor din grafic se poate folosi funcția `grid`

Alte proprietăți:

- `MarkerEdgeColor` pentru a selecta culoarea de contur a markerului

Sintaxă: `plot(x, y, 'MarkerEdgeColor','simbol culoare')`

unde `'simbol culoare'` poate fi orice simbol din tabelul *Tabel 4.1*

Exemplu: `plot(x, y, 'MarkerEdgeColor','r')` va marca cu roșu conturul markerului

- `MarkerFaceColor` pentru a selecta culoarea de umplere a markerului

Sintaxă: `plot(x, y, 'MarkerFaceColor','simbol culoare')`

unde `'simbol culoare'` poate fi orice simbol din tabelul *Tabel 4.1*

Exemplu: `plot(x, y, 'MarkerFaceColor','r')` va marca cu roșu interiorul markerului

- `MarkerSize` pentru a seta dimesiunea markerului

Sintaxă: `plot(x, y, 'MarkerSize', valNum)`

unde `valNum` este o valoare numerică pentru dimesiunea markerului

Exemplu: `plot(x, y, 'MarkerSize', 5)` va trasa markerul de dimensiune 5

- `LineWidth` pentru a seta grosimea liniei cu care sunt unite punctele

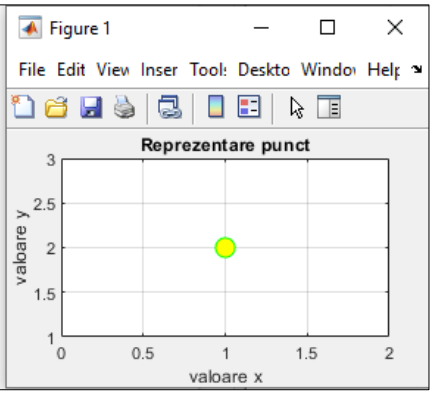
Sintaxă: `plot(x, y, 'LineWidth', valNum)`

unde `valNum` este o valoare numerică pentru grosimea liniei

Exemplu: `plot(x, y,'LineWidth',2)` va trasa graficul cu o linie de grosime 2

Să se reprezinte grafic punctul de coordonate (1, 2). Markerul să fie un cerc cu conturul verde, interiorul galben și de dimensiune 10.

```
clc
clear
close all
x = 1;
y = 2;
plot(1,2,'o','MarkerFaceColor','y',...,
    'MarkerEdgeColor','g',...,
    'MarkerSize',10)
grid
xlabel('valoare x'), ylabel('valoare y')
title('Reprezentare punct')
```



Să se reprezinte graficul din *Figura 4.4*, adăugând titlu și semnificații axelor O*x* și O*y*. Axa O*x* să fie reprezentată în intervalul [0, 12] și axa O*y* în intervalul [-1, 5].

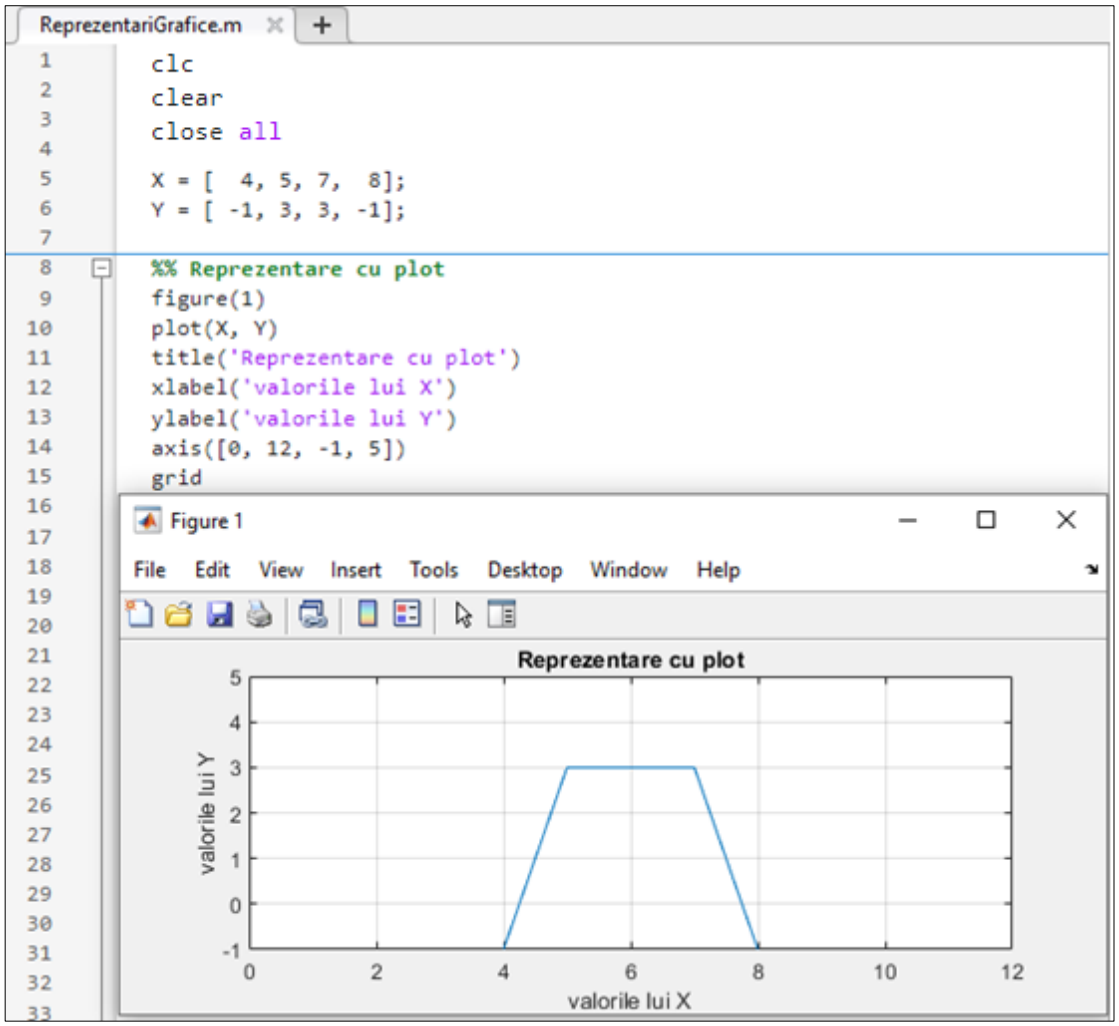


Figura 4.5. Exemplu de reprezentare grafică cu funcția `plot`

Observație: Dacă se specifică tipul marker-ului fără a se specifica și tipul de linie, atunci funcția `plot` va marca doar punctele din care este constituit graficul, fără a le mai interpola.

Pentru graficul din *Figura 4.4* să se reprezinte cu cercuri roșii doar punctele din care este format graficul, fără a trasa și liniile.

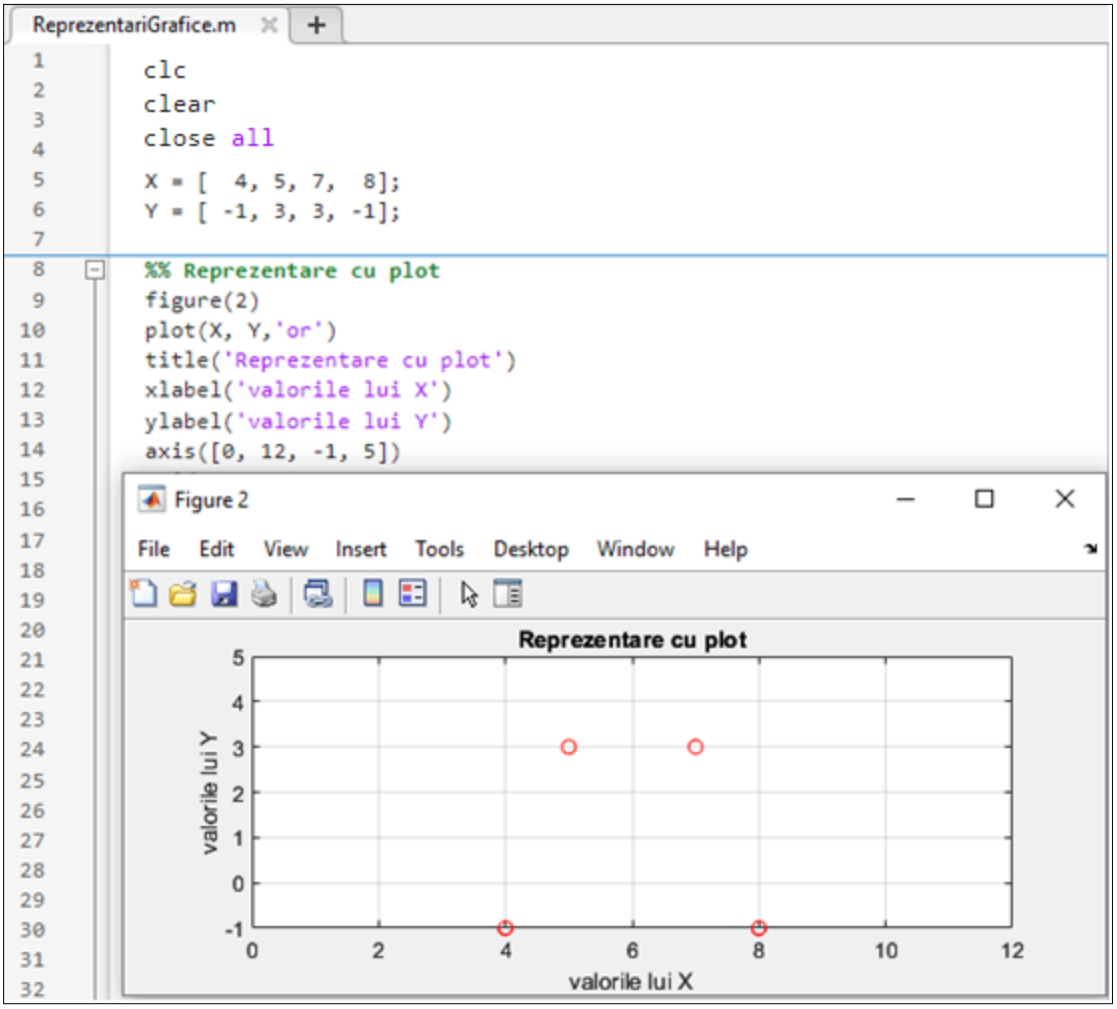


Figura 4.6. Exemplu de reprezentare grafică cu funcția `plot`
S-au reprezentat grafic doar punctele din care este format graficul, fără a le interpola

Să se personalizeze graficul din *Figura 4.4* astfel: linia să fie albastră și punctată, punctele să fie cercuri roșii cu contur verde. Grosimea liniei să fie 2.

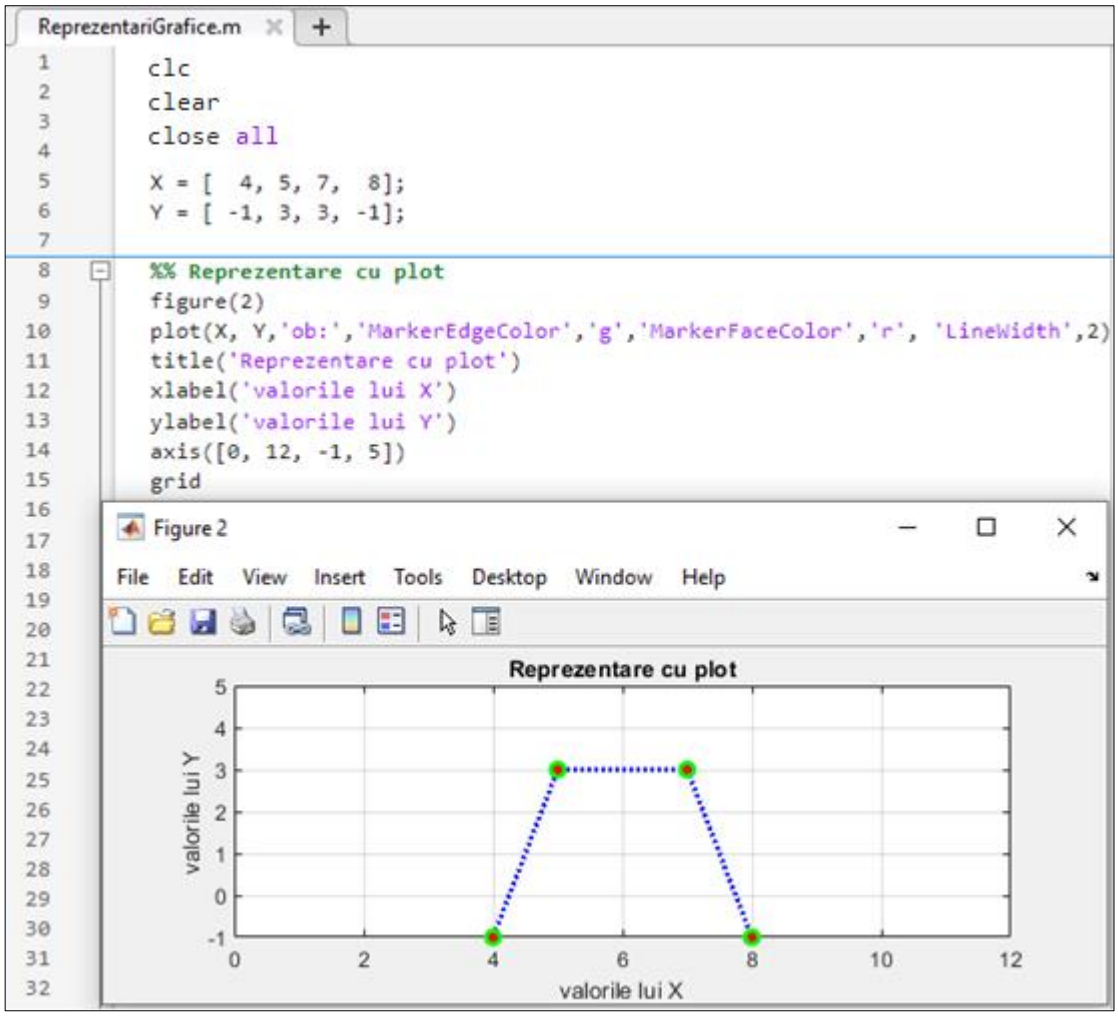


Figura 4.7. Exemplu de reprezentare grafică cu funcția `plot`
S-au folosit diferite culori, markere și tipuri de linie

4.3. Personalizarea axelor Ox, Oy și adăugarea textului pe grafic

Fie punctele A(5, 2) și B(7, 9). Cu noțiunile prezentate până acum, putem reprezenta cele 2 puncte astfel:

```
1   clc
2   clear
3   close all
4
5   x1 = 5;
6   x2 = 7;
7   y1 = 2;
8   y2 = 9;
9   plot([x1,x2],[y1, y2],'ob','MarkerFaceColor','g')
10  axis([4,9,1,10]), grid
11  xlabel('axa Ox'), ylabel('axa Oy')
12  title('Punctele A si B')
```

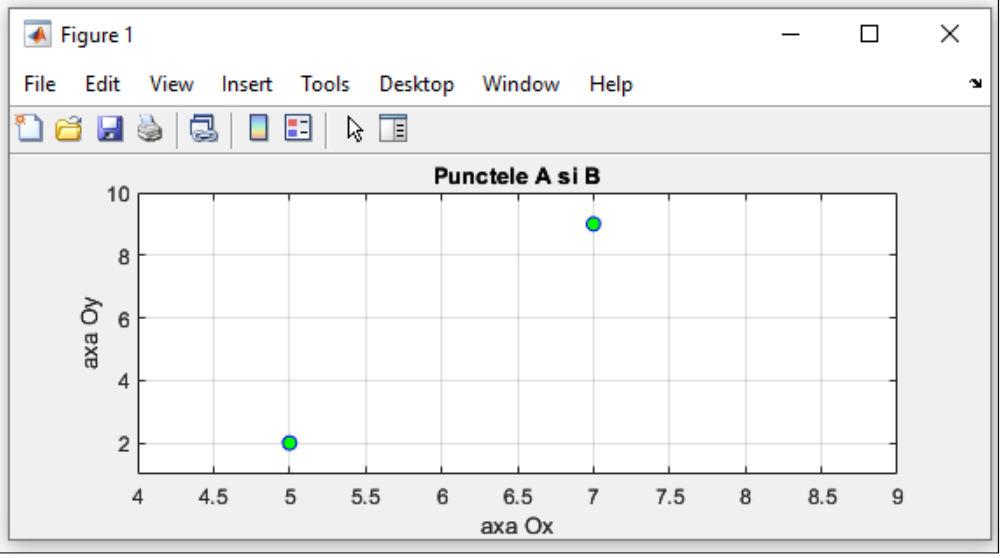
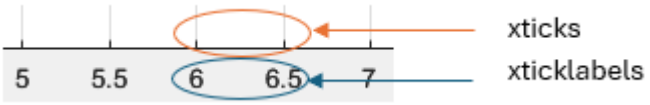


Figura 4.8. Reprezentarea grafică a punctelor A(5, 2) și B(7, 9)

Observații:

- Dacă privim axa O*x*, observăm niște marcaje (*xticks*) iar în dreptul lor niște valori numerice (*xticklabels*). Fix același lucru se întâmplă și pentru axa O*y*.



- Marcajele și valorile lor sunt adăugate automat de Matlab în funcție de valorile punctelor ce sunt reprezentate grafic.

Reprezentarea grafică a axelor poate fi însă personalizată, astfel încât pe axele Ox și Oy să apară ce valori ne dorim. Pentru a realiza acest lucru se folosesc:

- pentru axa O*x*, funcțiile: `xticks` și `xticklabels`
- pentru axa O*y*, funcțiile: `yticks` și `yticklabels`

4.3.1. Funcțiile `xticks` și `xticklabels`

- funcția `xticks` precizează valorile de pe axa O*x* unde urmează să se adauge etichete cu funcția `xticklabels`
- funcția `xticklabels` precizează etichetele care se vor scrie pe axa O*x* în punctele precizate cu funcția `xticks`

Sintaxă: `xticks([marcaje_Ox])`

`marcaje_Ox` = valorile de pe axa O*x* unde vor fi introduse marcajele

Sintaxă: `xticklabels({nume_marcaje_Ox})`

`nume_marcaje_Ox` = denumirile marcajelor ce vor apărea pe axa O*x*

Exemplu:

```
xticks([0.1, 0.3, 0.4, 0.7])
xticklabels({'val1','val2','val3','val4'})
```

Codul de mai sus va insera pe axa O*x* marcaje pentru valorile 0.1, 0.3, 0.4 și 0.7 iar informațiile inserate în dreptul acestor marcaje vor fi:

- pentru 0.1 ➔ `'val1'`
- pentru 0.3 ➔ `'val2'`
- pentru 0.4 ➔ `'val3'`
- pentru 0.7 ➔ `'val4'`

4.3.1. Funcțiile `yticks` și `yticklabels`

- funcția `yticks` precizează valorile de pe axa O*y* unde urmează să se adauge etichete cu funcția `yticklabels`
- funcția `yticklabels` precizează etichetele care se vor scrie pe axa O*y* în punctele precizate cu funcția `yticks`

Dacă dorim de exemplu ca reprezentarea grafică a punctelor `A` și `B` din exemplul anterior să fie la modul generic, fără să apară valori numerice pe grafic, putem folosi funcțiile menționate anterior, astfel:

```
1   clc
2   clear
3   close all
4
5   x1 = 5;
6   x2 = 7;
7   y1 = 2;
8   y2 = 9;
9   plot([x1,x2],[y1, y2],'ob','MarkerFaceColor','g')
10  axis([4,9,1,10]), grid
11  xticks([x1, x2])
12  xticklabels({'x_1', 'x_2'})
13  yticks([y1, y2])
14  yticklabels({'y_1', 'y_2'})
15  title('Reprezentarea punctelor A(x_1, y_1) si B(x_2, y_2)')
```

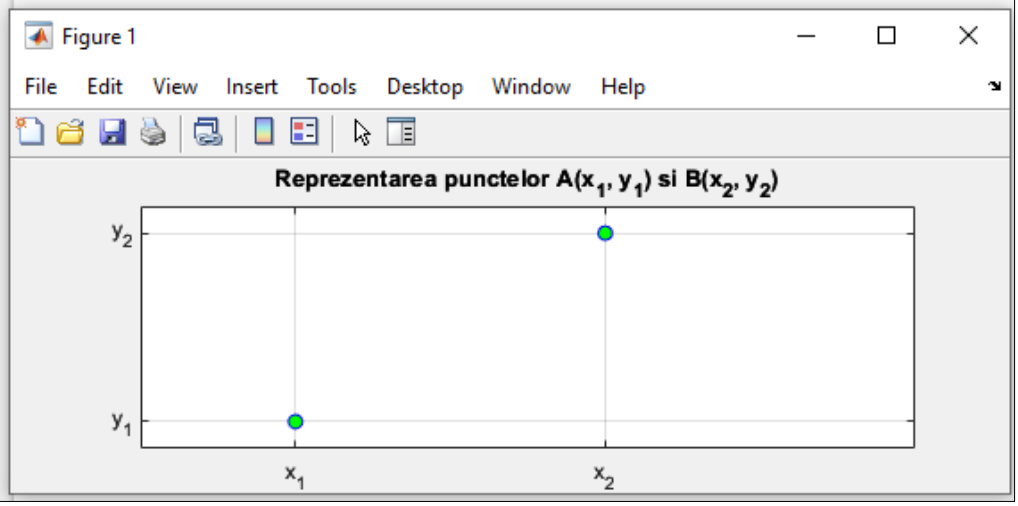


Figura 4.9. Personalizarea axelor O*x* și O*y*

Se poate observa că:

- pe axa O*x*, în locul valorilor 5 și 7 au apărut parametrii x_1 și x_2
- pe axa O*y*, în locul valorilor 2 și 9 au apărut parametrii y_1 și y_2
- în cazul șirurile de caractere care conțin *undersore* (_), are loc transformarea în *Subscript* a ceea ce urmează după *undersore*.

Exemplu: `'x_1'` ➔ x_1

Lista completă a caracterelor speciale care pot apărea într-un titlu o găsiți în *Anexa B.*

4.3.2. Funcția `text`

Funcția `text` permite introducerea de text pentru a descrie punctele unui grafic.

Adăugarea descrierii unui singur punct:

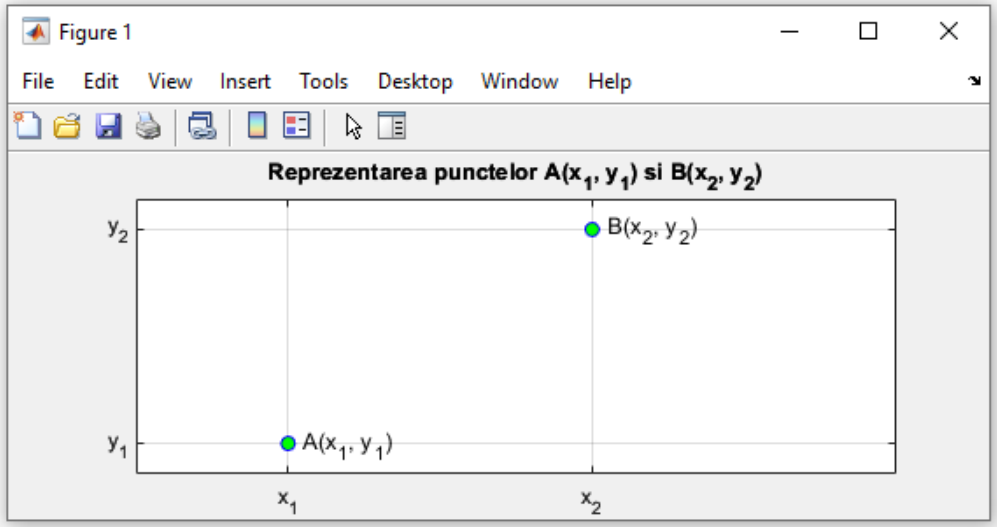
Sintaxă: `text(coordonata Ox, coordonata Oy, 'text')`

Adăugarea descrierii mai multor puncte:

Sintaxă: `text([coordonate Ox, coordonate Oy], {'text1', 'text2',...})`

Fie punctele A(5, 2) și B(7, 9). Să se reprezinte grafic punctele, adăugând pe grafic informații despre puncte (de exemplu coordonatele).

```
1   clc
2   clear
3   close all
4
5   x1 = 5;
6   x2 = 7;
7   y1 = 2;
8   y2 = 9;
9   plot([x1,x2],[y1, y2],'ob','MarkerFaceColor','g')
10  axis([4,9,1,10]), grid
11  xticks([x1, x2])
12  xticklabels({'x_1', 'x_2'})
13  yticks([y1, y2])
14  yticklabels({'y_1', 'y_2'})
15  text(x1+0.1,y1,'A(x_1, y_1)')
16  text(x2+0.1,y2,'B(x_2, y_2)')
17  title('Reprezentarea punctelor A(x_1, y_1) si B(x_2, y_2)')
```



Observație: liniile de cod 15 și 16 de mai sus pot fi scrise comasat astfel:

```
text([x1+0.1,x2+0.1],[y1,y2],{'A(x_1, y_1)', 'B(x_2, y_2)'})
```

4.4. 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`).
- Vectorii `X` și `Y` trebuie să aibă aceleași dimensiuni.
- Proprietățile pentru culoare, tip de linie și marker sunt cele de la funcția `plot`.

Să se reprezinte doar eșantioanele din care este format graficul din *Figura 4.4*.

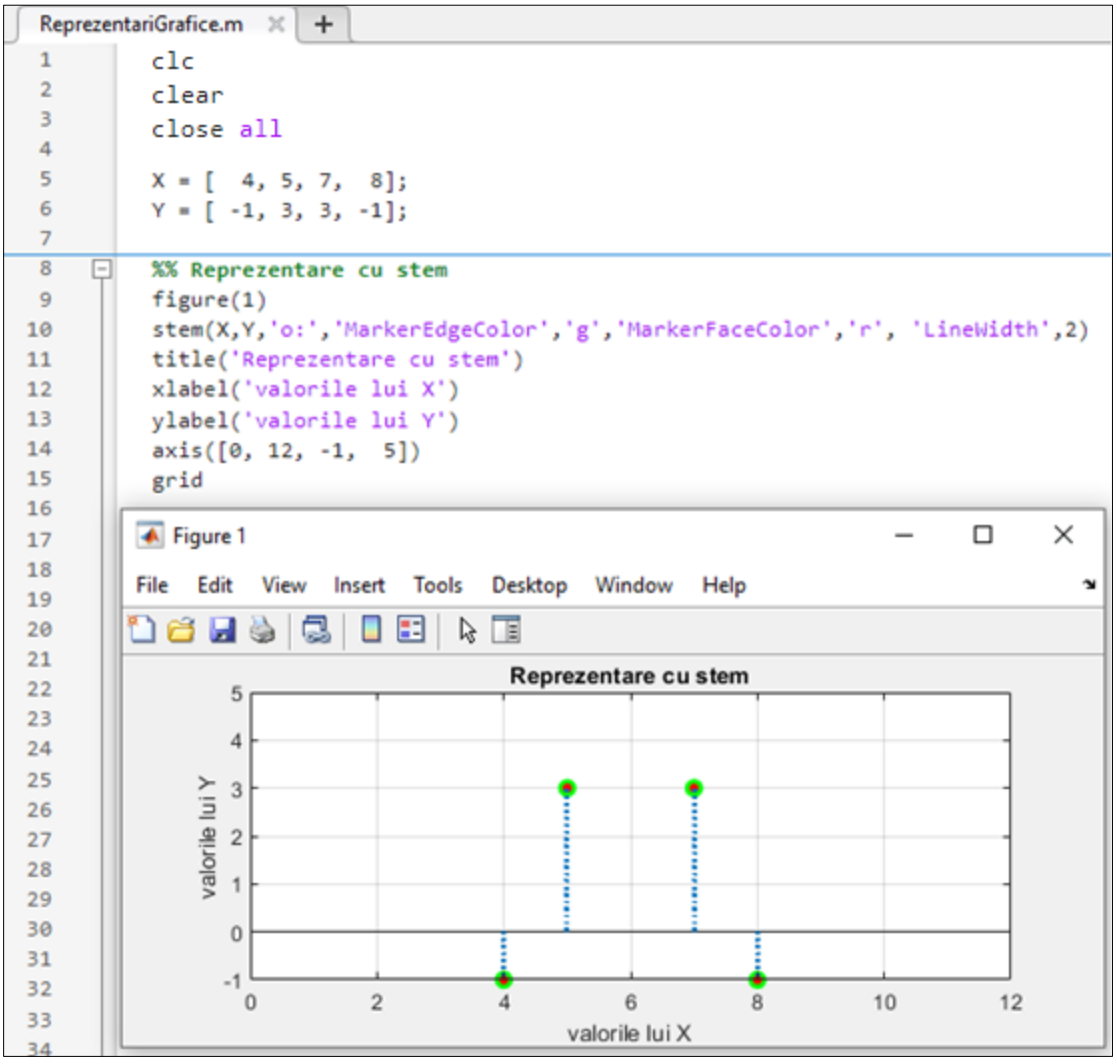


Figura 4.10. Exemplu de reprezentare grafică cu funcția `stem`

4.5. Reprezentarea graficelor în aceeași figură, în același sistem de coordonate

Pentru a reprezenta mai multe grafice în aceeași figură, în același sistem de coordonate, toate reprezentările grafice trebuie să se realizeze între instrucțiunile `hold on ... hold off`

Sintaxă:

```
hold on
    reprezentare grafic_1
    reprezentare grafic_2
    ....
    reprezentare grafic_n
hold off
```

Pentru o mai ușoară identificare a graficelor, se poate adăuga o legendă folosind funcția `legend`.

Sintaxă: `legend('nume_1', 'nume_2',..., 'nume_n','Location',POZ)`

unde:

- `'nume_1'` reprezintă numele ce va apărea în legendă pentru primul grafic reprezentat după `hold on`
- `'nume_2'` reprezintă numele ce va apărea în legendă pentru al doilea grafic reprezentat după `hold on`
- `'nume_n'` reprezintă numele ultimului grafic ce va apărea în legendă pentru ultimul grafic reprezentat după `hold on`
- `POZ` reprezintă localizarea legendei în spațiul figurii și poate fi:

Valoare parametru `POZ`	Descriere
`'north'`	In interiorul graficului, în partea de sus
`'northeast'`	În interiorul graficului, în partea dreapta-sus
`'northwest'`	În interiorul graficului, în partea stânga-sus
`'northoutside'`	În exteriorul graficului, în partea de sus
`'northeastoutside'`	În exteriorul graficului, în partea dreapta-sus
`'northwestoutside'`	În exteriorul graficului, în partea stânga-sus
etc	... pe același model și pentru `east, west, south`

Fie vectorii `X = [4, 5, 7, 8]` și `Y = [-1, 3, 3, -1]`.

Să se reprezinte în aceeași figură, în același sistem de coordonate, atât cu `plot` cât și cu `stem`, graficul determinat de perechile de puncte `{X(i), Y(i)}`. Legenda să fie afișată în interiorul graficului, în partea stânga-sus.

ReprezentariGrafice.m

```
clc
clear
close all
X = [  4, 5, 7,  8];
Y = [ -1, 3, 3, -1];

%%  Reprezentarea mai multor grafice in aceeasi figura,
% in acelasi sistem de coordonate
figure()
hold on
    stem(X, Y, 'r')
    plot(X, Y, 'g')
hold off
title('Reprezentarea mai multor grafice în aceeași fereastră')
xlabel('valori X')
ylabel('valori Y')
axis([0, 12, -1,  5])
grid
legend('reprez. cu stem','reprez. cu plot','location','northeast')
```

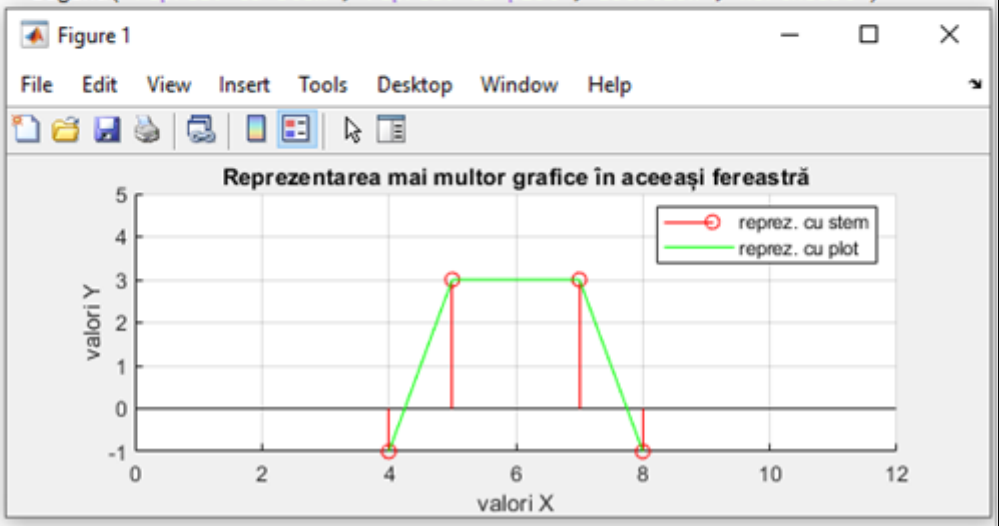


Figura 4.11. Exemplu de reprezentare a mai multor grafice, în aceeași figură, în același sistem de coordonate

4.6. Reprezentarea graficelor în aceeași figură, în sisteme de coordonate diferite

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.

Exemplu: 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)`

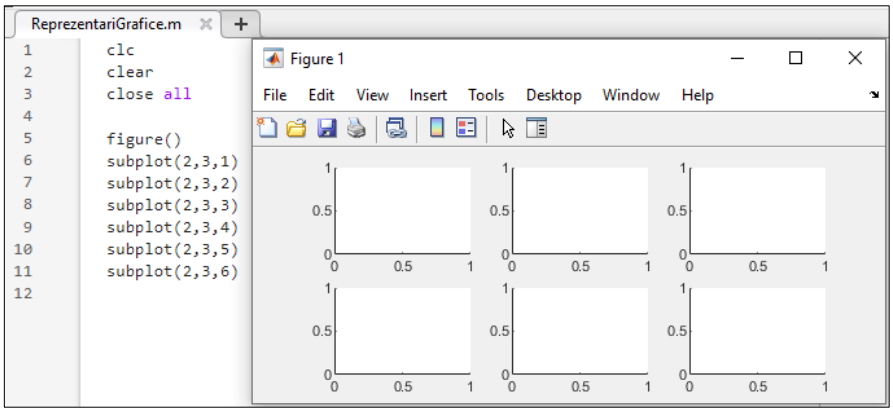


Figura 4.12. Exemplu de reprezentare a mai multor sisteme de coordonate în aceeași figură

Observații:

- `subplot` indică doar zona în care se face afișarea, nu face și afișarea. Pentru afișare se folosește una dintre funcțiile `plot`, `stem` etc.
- Dacă se dorește să se unească de exemplu celulele de pe linia 2 și coloanele 2 și 3, structura va fi următoare:

`subplot(2,3,1)`	`subplot(2,3,2)`	`subplot(2,3,3)`
`subplot(2,3,4)`	`subplot(2,3,[5,6])`	

- pentru a adăuga titlu general tuturor subploturilor se folosește funcția `sgtitle`.

Fie vectorii `X = [4, 5, 7, 8]` și `Y = [-1, 3, 3, -1]`.

Să se afișeze în aceeași figură, în sisteme de coordonate diferite, pe 1 linie și 3 coloane:

- graficul reprezentat cu `plot`
- graficul reprezentat cu `stem`
- graficele reprezentate atât cu `plot` cât și cu `stem`

ReprezentariGrafice.m

```
clc
clear
close all
X = [  4, 5, 7,  8];
Y = [ -1, 3, 3, -1];

%% Reprezentarea mai multor grafice in aceeasi figura
% in sisteme de coordonate diferite
figure()
sgtitle('Exemple subplot')
subplot(1,3,1)
    plot(X,Y,'g','Linewidth',2)
    xlabel('valori X'), ylabel('valori Y')
    title('plot')
    axis([0, 12, -1,  5]), grid
subplot(1,3,2)
    stem(X,Y,'b','MarkerFaceColor','r')
    title('stem')
    xlabel('valori X'), ylabel('valori Y')
    axis([0, 12, -1,  5]), grid
subplot(1,3,3)
    hold on
        plot(X,Y,'g','Linewidth',2)
        stem(X,Y,'b','MarkerFaceColor','r')
    hold off
    xlabel('valori X'), ylabel('valori Y')
    axis([0, 12, -1,  5]), grid
    title('plot și stem')
```

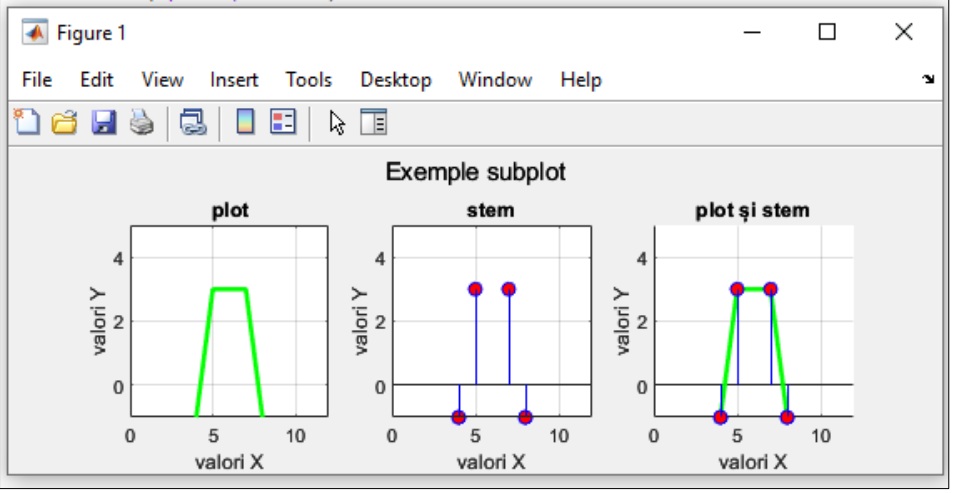


Figura 4.13. Exemplu de reprezentare a mai multor grafice în aceeași figură

Fie funcția:

$$f(x) = x^2 - 2 \cdot x + 1, \; pentru \; x \in [-5, \; 7]$$

Diferența dintre 2 valori consecutive ale lui x este 0.01.

Să se reprezinte graficul funcției $f(x)$, marcând cu roșu punctul de minim.

graficFunctie.m

```
clc
clear
close all

x = -5:0.01:7;
f = x.^2 - 2*x +1;
% determinare valoare minimă
[valMin, pozMin] = min(f);
figure(1)
hold on
    % reprezentarea grafică a funcției
    plot(x, f)
    % afișare valoare minimă pe grafic
    plot(x(pozMin), f(pozMin),'or', 'MarkerFaceColor','r')
    xlabel('valori x'), ylabel('f(x)')
    title('f(x) = x^2 - 2*x +1'), grid
```

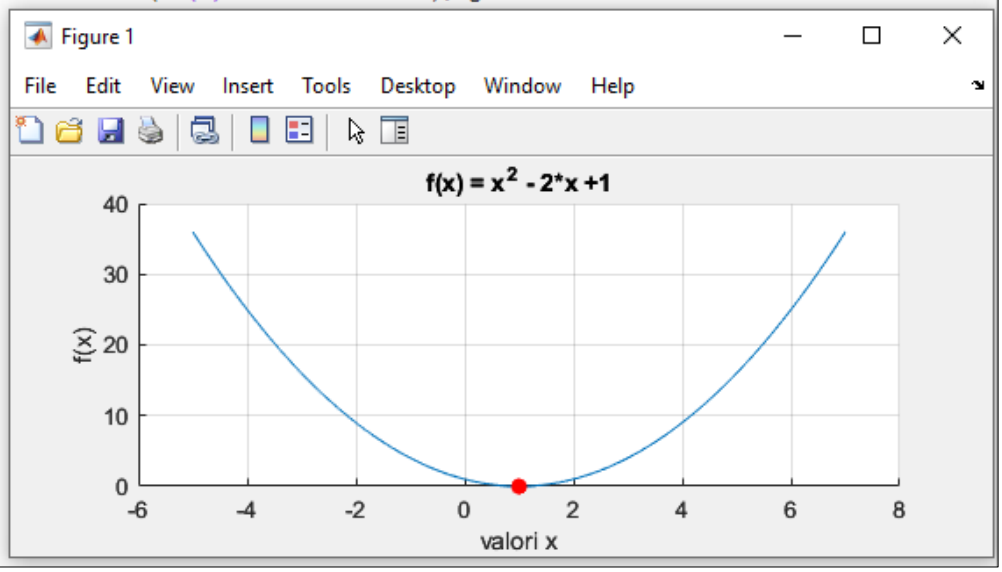


Figura 4.14. Reprezentarea grafică a unei funcții

4.7. Aplicații

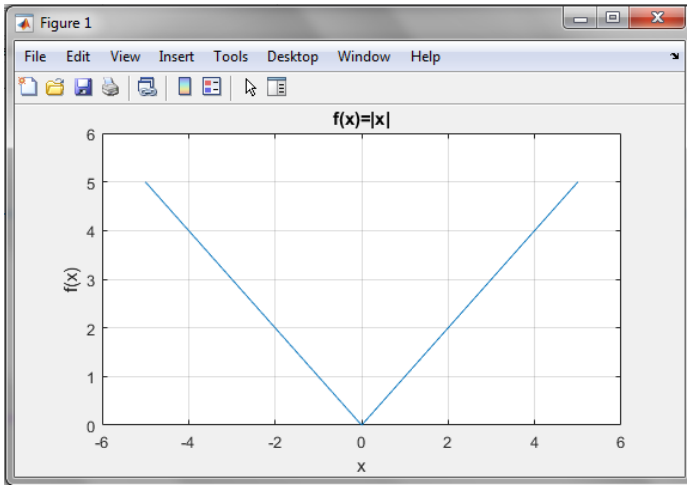
Aplicația 1. Fie funcția:

$$f(x) = |x|,\ pentru\ x \in [-5,\ 5]$$

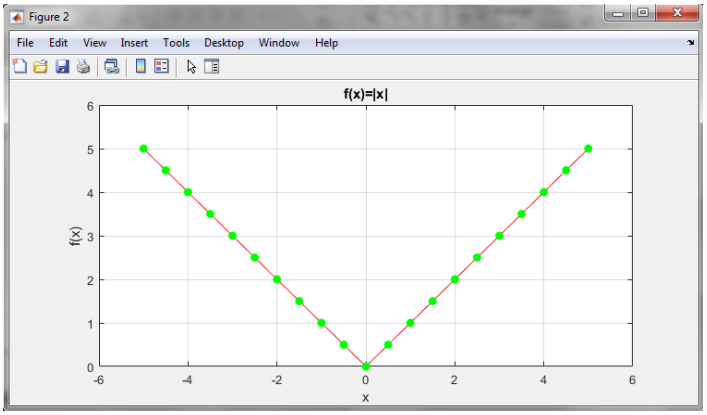
Distanța dintre 2 valori consecutive ale lui x este 0.5.

Cerințe:

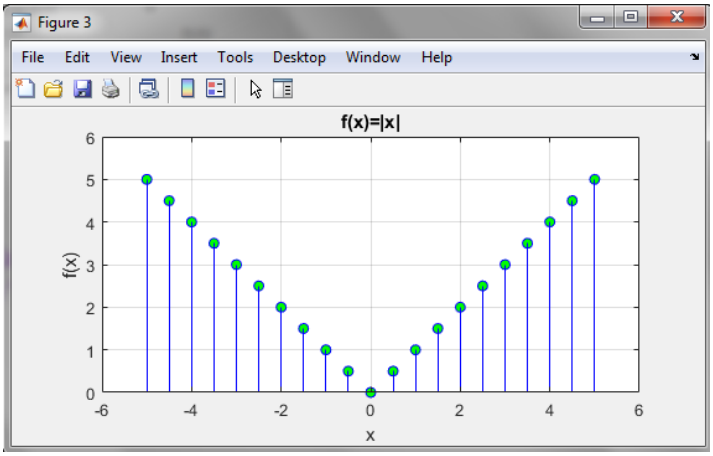
a) Să se reprezinte graficul funcției $f(x)$. Graficul să conțină titlu și semnificațiile axelor O*x* și O*y*



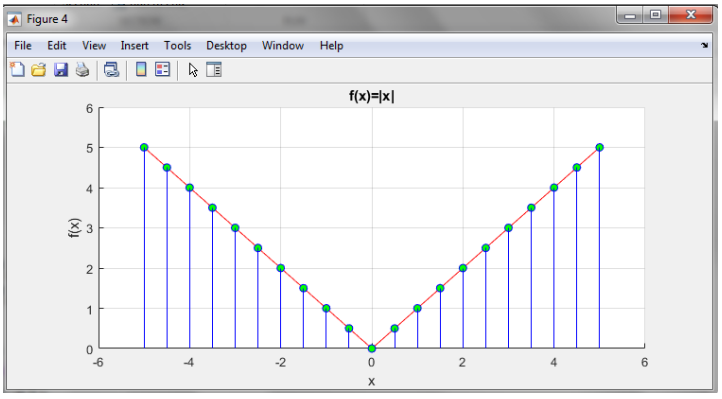
b) Să se reprezinte cu roșu graficul funcției $f(x)$. Toate punctele din care este format graficul să se marcheze cu verde.



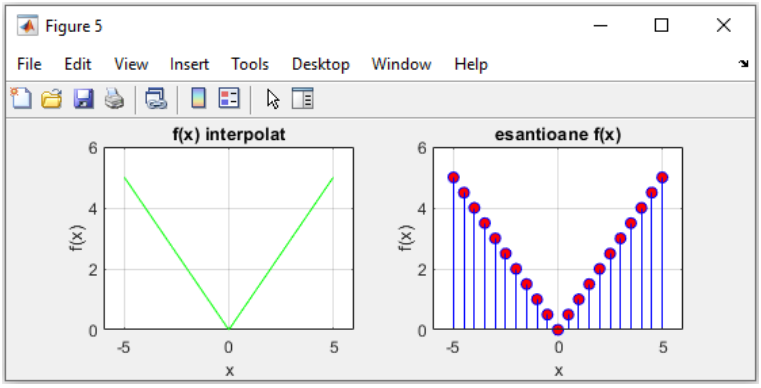
c) Să se reprezinte toate eșantioanele funcției $f(x)$.



d) Să se reprezinte în aceeași figură, **în același sistem de coordonate**, atât graficul funcției $f(x)$ în urma interpolării cât și eșantioanele funcției $f(x)$.



e) Să se reprezinte în aceeași figură, **în sisteme de coordonate diferite**, atât graficul funcției $f(x)$ în urma interpolării cât și eșantioanele funcției $f(x)$.



Aplicația 2. Fie funcția:

$$f(x) = x^3 - 6x^2 + 8x - 6,\ pentru\ x \in [0,\ 5]$$

Distanța dintre 2 valori consecutive ale lui x este 0.01.

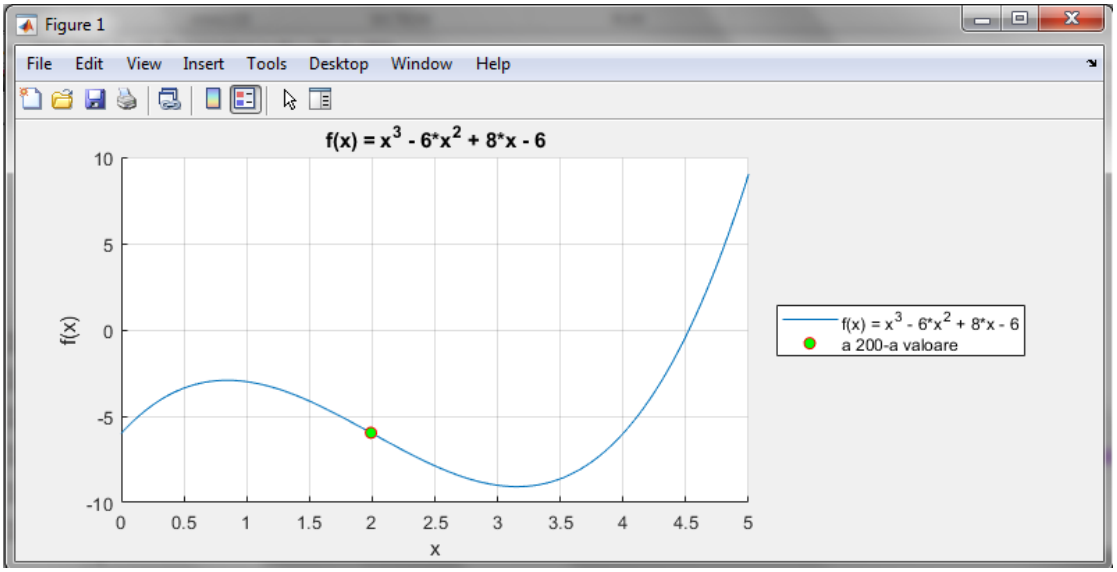
Cerințe:

a) Să se reprezinte graficul funcției $f(x)$.

b) Să se reprezinte cu un punct, eșantionul cu numărul 200.

c) Să se adauge legenda graficelor.

Observație: În urma implementării cerințelor, reprezentarea grafică va arăta astfel:



Aplicația 3. Fie funcția:

$$f(x) = \frac{x}{x^2 + 1}$$

unde $x \in [-4,\ 4]$ și distanța dintre 2 valori consecutive ale lui x este 0.01.

Cerințe:

a) Să se reprezinte graficul funcției $f(x)$.

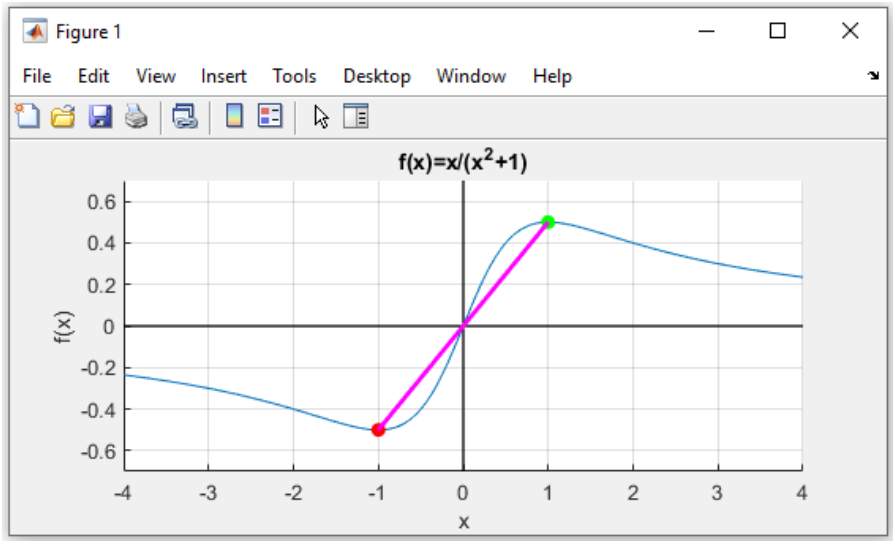
b) Să se traseze cu negru axele O*x* și O*y*.

c) Să se reprezinte cu un punct verde valoarea maximă a funcției $f(x)$.

d) Să se reprezinte cu un punct roșu valoarea minimă a funcției $f(x)$.

e) Să se unească cu un segment punctele de maxim și de minim ale funcției $f(x)$.

Observație: În urma implementării cerințelor, reprezentarea grafică va arăta astfel:



Aplicația 4. Fie funcția:

$$f(x) = \frac{x^2}{5\pi} + |x| - 1,\ pentru\ x \in [-100{,}100]$$

Diferența dintre 2 valori consecutive ale lui x este 0.01.

Cerințe:

a) Să se reprezinte graficul funcției $f(x)$.

b) Să se determine valoarea minimă a funcției $f(x)$ și să se reprezinte pe grafic cu un punct roșu.

Aplicația 5. Fie funcția:

$$f(x) = -\frac{x}{e^x + e^{-x}}$$

unde $x \in [-10, 10]$ și distanța dintre 2 valori consecutive ale lui x este 0.001.

Cerințe:

a) Să se reprezinte graficul funcției $f(x)$.

b) Să se reprezinte pe grafic, cu un punct roșu, valoarea maximă a funcției.

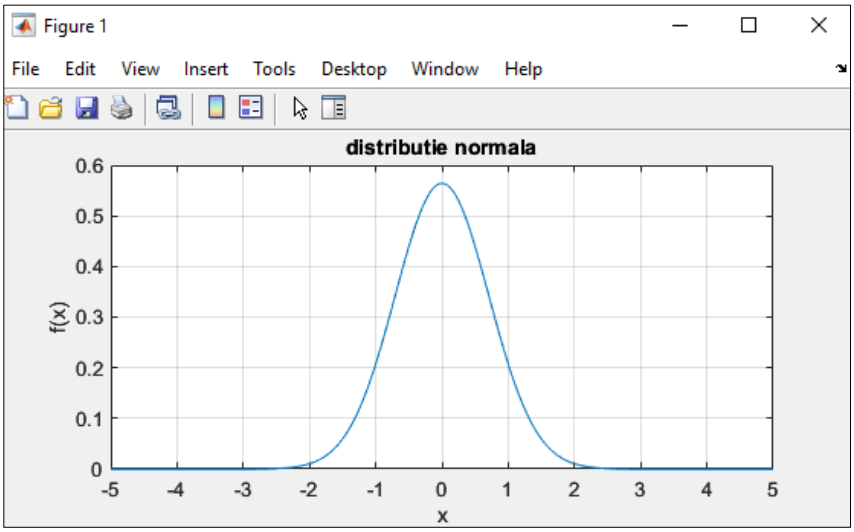
c) Să se traseze cu negru axele O*x* și O*y*.

Aplicația 6. Să se afișeze grafic funcția de distribuție normală:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-m)^2}{2\sigma^2}}$$

pentru $x \in [-5,5]$. Distanța dintre două valori consecutive ale lui x este 0.01.

Se cunosc: media $m = 0$ și varianța $\sigma^2 = 0.5$

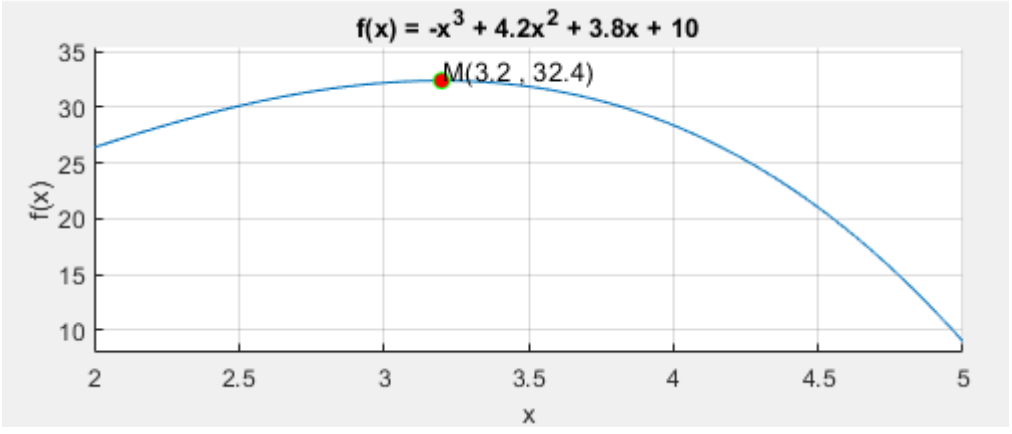


Aplicația 7. Fie funcția:

$$f(x) = -x^3 + 4.2x^2 + 3.8x + 10$$

a) Să se reprezinte graficul funcției $f(x)$.

b) Să se determine valoarea maximă a funcției și să se reprezinte pe grafic punctul `M` de maxim și coordonatele acestuia.



Capitolul 5
Reprezentări grafice 2D
Semnale particulare

Dacă în capitolul anterior am văzut cum se poate reprezenta la modul general orice semnal în spațiul 2D folosind funcțiile `plot` și `stem`, în acest capitol vom discuta mai în detaliu modul de reprezentare a două semnale extrem de importante în inginerie, și anume **semnalele sinusoidale** și **semnalele dreptunghiulare**. Se va explica pas cu pas cum se ajunge de la semnalul analogic la cel numeric și cum poate fi generat acesta în Matlab.

Noțiuni matematice recapitulative

Fie funcția $f: A \to B$

- mulțimea A se numește DOMENIU DE DEFINIȚIE
- mulțimea B se numește CODOMENIU sau DOMENIU DE VALORI

După natura continuă sau discontinuă a domeniului de definiție și a celui de valori, semnalele se pot clasifica conform *Tabelului 5.1*.

Tabel 5.1. Clasificarea semnalelor după natura continuă sau discontinuă a domeniului de definiție și a celui de valori [2]

		Domeniu de valori	
		Continuu	**Discret**
Domeniu de definiție (timpul)	**Continuu**	Semnale continue în timp continuu *(analogice)*	Semnale discrete în timp continuu
	Discret	Semnale continue în timp discret	Semnale discrete în timp discret *(numerice)* **Cu aceste semnale lucrăm în Matlab!**

Funcțiile trigonometrice sin și cos

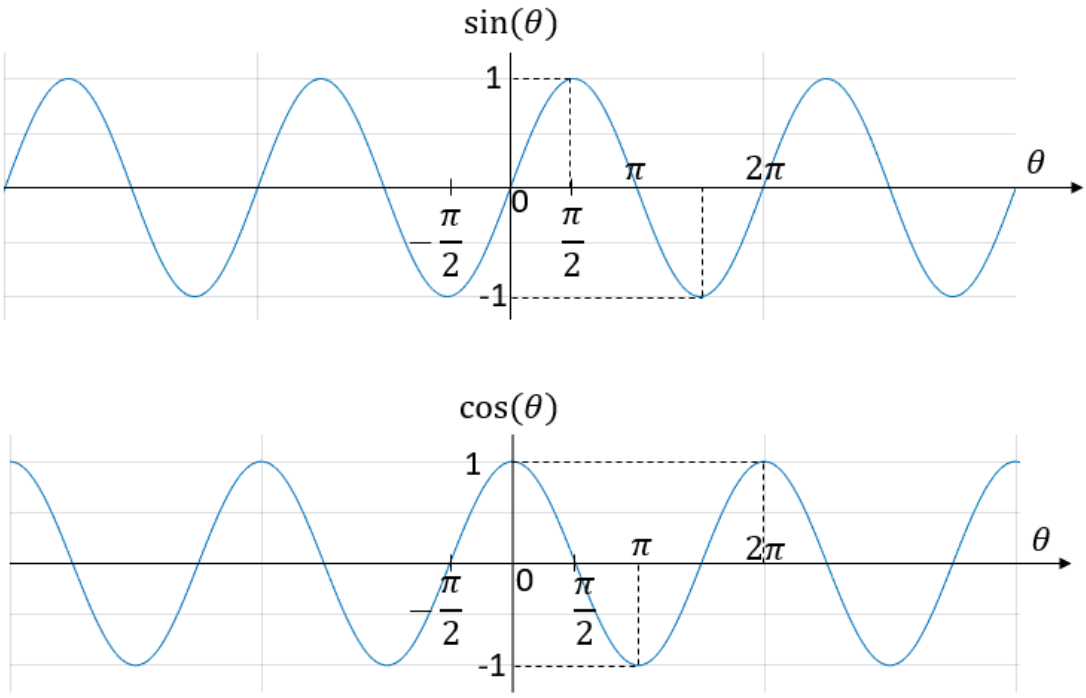
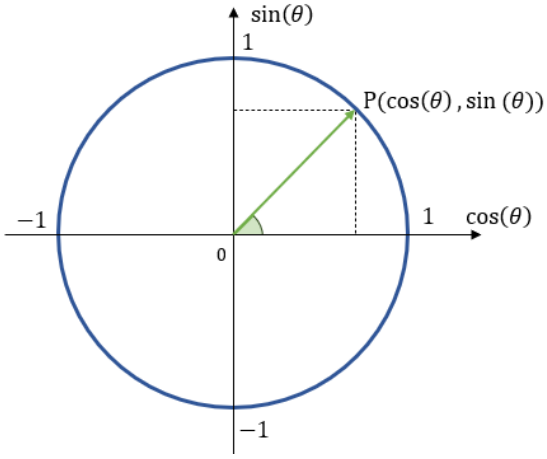


Figura 5.1. Funcțiile trigonometrice sin și cos

Observație: Un sinus defazat cu $\frac{\pi}{2}$ devine un cosinus.



θ [rad]	$\sin(\theta)$	$\cos(\theta)$
0	0	1
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$
$\frac{\pi}{2}$	1	0
π	0	-1
2π	0	1

Figura 5.2. Cercul trigonometric

5.1. Semnal sinusoidal continuu. *Considerente matematice*

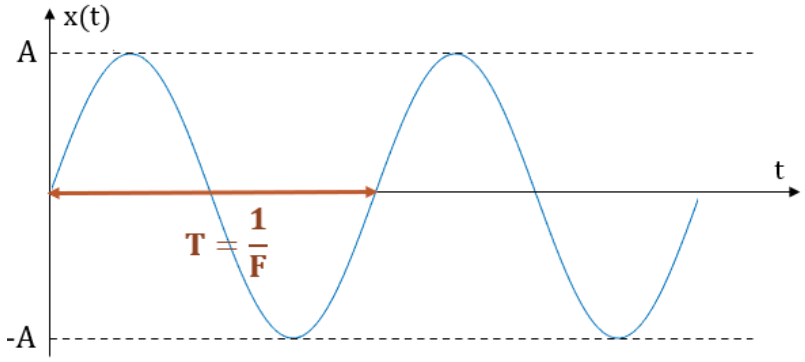


Figura 5.3. Semnal sinusoidal continuu

Un semnal sinusoidal poate fi privit ca un sinus care poate fi amplificat, defazat și căruia i se poate modifica frecvența de repetiție.

Pentru orice semnal periodic:

- **perioada T** a semnalului reprezintă intervalul de timp după care semnalul se repetă (durata unei oscilații complete).
- **frecvența de repetiție F** a semnalului reprezintă numărul de perioade (oscilații) ale semnalului dintr-o secundă ➔ **F = 1/T**.

Formula matematică a unui semnal sinusoidal continuu este:

$$x(t) = A \cdot \sin(\omega t + \varphi_0) \qquad (1)$$

unde:

- A = amplitudinea maximă
- φ = faza; $\varphi = \omega t + \varphi_0$; $[\varphi]_{S.I} = 1\ rad$
- ω = pulsația; $\omega = 2\pi \cdot F$; $[\omega]_{S.I} = 1\frac{rad}{s}$
- φ_0 = faza inițială; $[\varphi_0]_{S.I} = 1\ rad$

Înlocuind $\omega = 2\pi \cdot F$, obținem:

$$x(t) = A \cdot \sin(2\pi \cdot F \cdot t + \varphi_0) \qquad (2)$$

5.2. Semnal sinusoidal numeric. *Considerente matematice*

În urma eșantionării rezultă semnalul numeric (vectorul):

$$x[n \cdot Ts] = A \cdot \sin(2\pi \cdot F \cdot nTs + \varphi_0) \qquad (3)$$

- **Ts** (perioada de eșantionare) = intervalul de timp dintre două eșantioane consecutive
- **Fs** (frecvența de eșantionare) = numărul de eșantioane dintr-o secundă

$$Fs = \frac{1}{Ts}$$

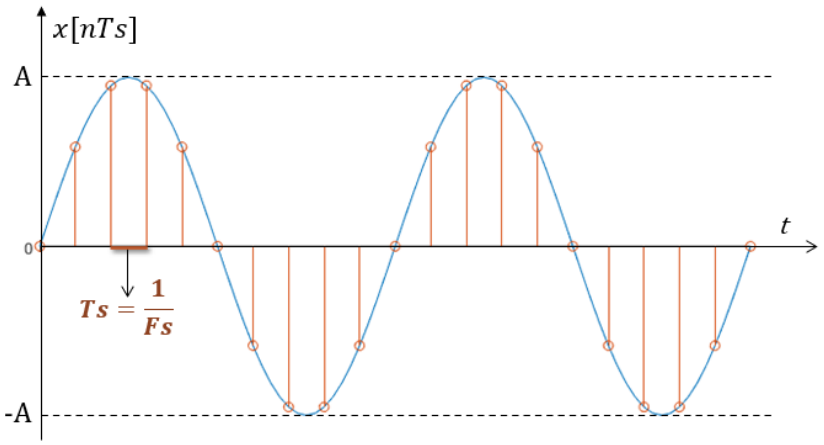


Figura 5.4. Semnalul sinusoidal continuu (cu albastru)
Eșantioanele semnalului sinusoidal (cu roșu)

Formula sinusoidei discrete se mai poate scrie în funcție de frecvența de eșantionare:

$$x[n] = A \cdot \sin\left(2\pi \cdot n \cdot \frac{F}{Fs} + \varphi_0\right) \qquad (4)$$

- pentru un semnal periodic, **perioada T** a semnalului reprezintă cel mai scurt interval de timp după care semnalul se repetă
- pentru un semnal periodic, **frecvența de repetiție F** a semnalului reprezintă numărul de perioade ale semnalului dintr-o secundă, **F = 1/T**
- intervalul de timp dintre două eșantioane consecutive se numește **perioadă de eșantionare** și se notează **Ts** (*s* de la *sampling* = eșantionare în engleză)
- **frecvența (rata) de eșantionare** reprezintă numărul de eșantioane obținute în unitatea de timp (o secundă), **Fs = 1/Ts**

5.3. Semnal sinusoidal. *Implementare în Matlab*

Pentru a genera o sinusoidă în Matlab se poate folosi următoarea sintaxă:

Sintaxă: `s = A*sin(2*pi*F*t+faza0)`

unde:

- `A` = amplitudinea maximă a semnalului sinusoidal
- `F` = frecvența de repetiție a semnalului sinusoidal (în *Hz*)
- `faza0` = faza inițială a semnalului sinusoidal (în *radiani*)
- `t` = vectorul momentelor de timp obținute în urma eșantionării

  vectorul `t` se poate genera cu linia de cod:

  ```
  t = 0:1/Fs:durata;
  ```

  unde:

  - `Fs` = frecvența de eșantionare (în *Hz*)
  - `durata` = durata semnalului (în *secunde*)

Observații:

- semnalul generat cu sintaxa de mai sus va avea valoarea maximă `+A` și valoarea minimă `-A`
- dacă faza inițială este nulă (`faza0 = 0`), atunci la momentul de timp inițial semnalul va avea valoarea `0`

Să se genereze și să se afișeze în Matlab următoarea sinusoidă.

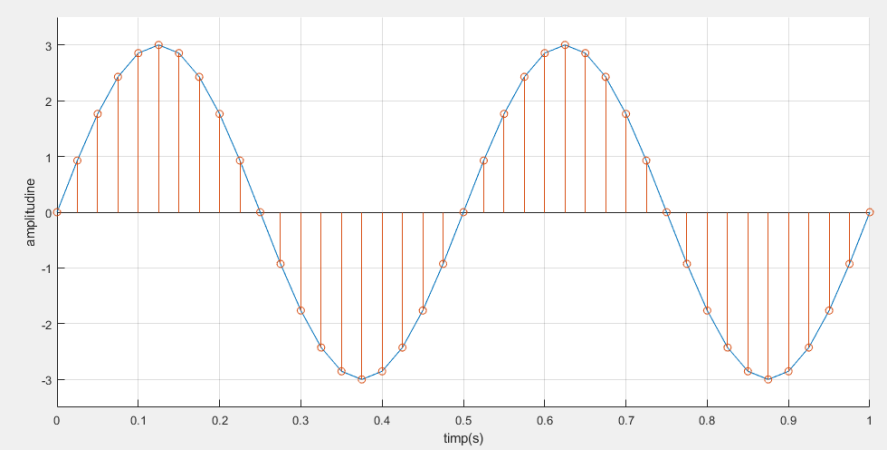


Figura 5.5. Sinusoidă generată în Matlab

Pentru semnalul din *Figura 5.5*, parametrii sunt:

- amplitudine $A = 3$
- frecvență $F = 2\text{Hz}$ (perioada $T = 0.5\text{s}$)
- fază inițială $\varphi_0 = 0\ rad$
- durata = 1s
- frecvența de eșantionare $Fs = 40\text{Hz}$ (deoarece sunt 40 de eșantioane într-o secundă)

Pași de urmat pentru generarea unei sinusoide în Matlab

Pas 1: scrierea celor 5 parametrii ai sinusoidei.

Pas 2: generarea vectorului `t` care să conțină momentele de timp obținute în urma eșantionării.

Pas 3: calcularea valorilor sinusoidei folosind *formula 4* pentru fiecare moment de timp din vectorul `t`.

Pas 4: afișarea sinusoidei în funcție de timp; pe axa O*x* sunt momentele de timp iar pe axa O*y* sunt reprezentate valorile sinusoidei.

semnalSinusoidal.m

```
clc
clear
close all

%% semnal sinusoidal
% parametrii sinusoidei:
A = 3; % amplitudine
F = 2; % frecventa de repetitie
faza0 = 0; % faza initiala
durata = 1; % durata semnalului
Fs = 40; % frecventa de esantionare

% generare sinusoida
% vectorul t este momentelor de timp obținute în urma eșantionării cu Fs
t = 0:1/Fs:durata;
s = A*sin(2*pi*F*t+faza0);
figure()
hold on
    plot(t,s,'b')
    stem(t,s,'r')
hold off
grid
xlabel('timp(s)'), ylabel('amplitudine')
title('sinusoida')
ylim([min(s)-0.2, max(s)+0.2])
```

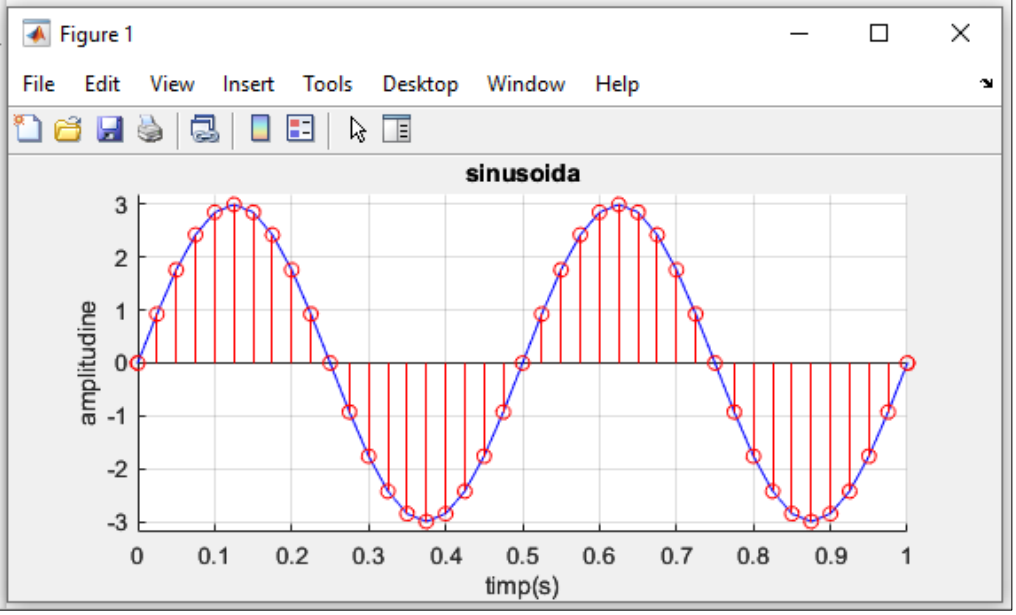


Figura 5.6. Generarea și afișarea unei sinusoide în Matlab

5.4. Semnal dreptunghiular continuu. *Considerente matematice*

Un semnal dreptunghiular este un semnal periodic care poate lua doar 2 valori posibile.

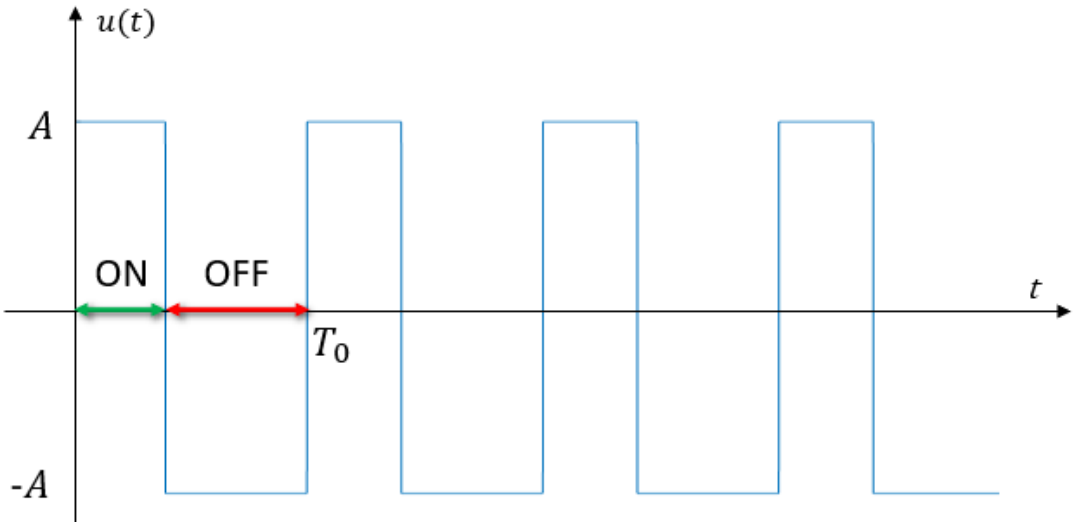


Figura 5.7. Semnal dreptunghiular

Pentru semnalul dreptunghiular se definește **factorul de umplere (q)** ca valoarea procentuală a duratei în care semnalul are valoarea maximă din durata totală a perioadei.

$$q = \frac{t_{ON}}{t_{ON} + t_{OFF}} \cdot 100\ [\%]$$

Dacă semnalul dreptunghiular are factor de umplere de 50% și semnalul are valorile `A` și `−A`, atunci forma lui generală este cea de mai jos.

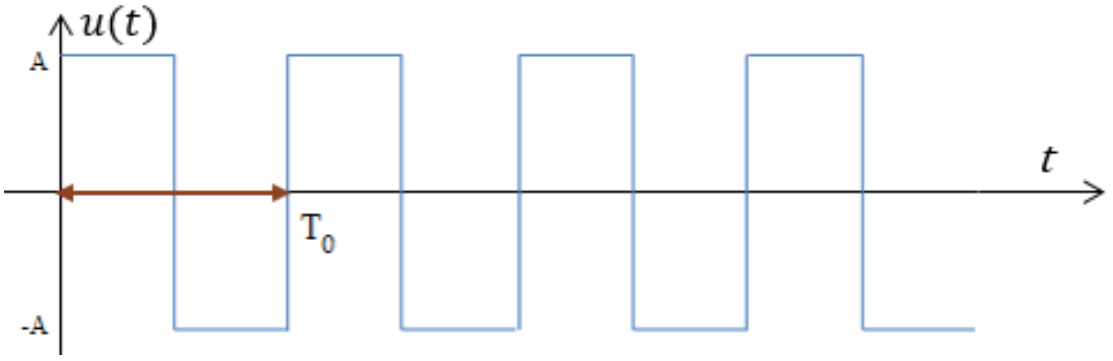


Figura 5.8. Semnal dreptunghiular de amplitudine *A* și perioadă T_0

5.5. Semnal dreptunghiular numeric. *Considerente matematice*

Atunci când se eșantionează un semnal dreptunghiular, trebuie ca distanța dintre două eșantioane consecutive (perioada de eșantionare `Ts`) să fie cât mai mică, pentru a se păstra cât mai multe eșantioane dintr-o perioadă. Dacă nu sunt suficiente eșantioane (cu alte cuvinte dacă frecvența de eșantionare `Fs` este prea mică) cu atât semnalul obținut prin interpolarea eșantioanelor va arăta mai mult cu un "semnal trapezoidal".

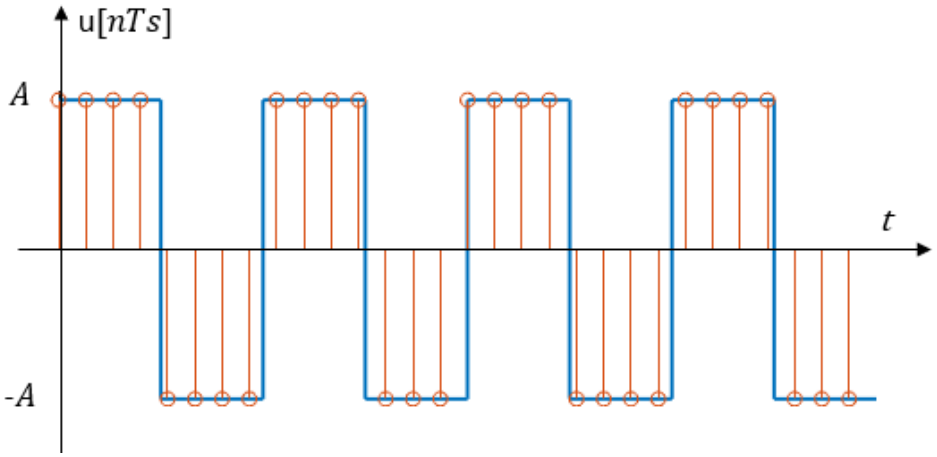


Figura 5.9. Semnalul dreptunghiular continuu (cu albastru)
Eșantioanele semnalului dreptunghiular (cu roșu)

5.6. Semnal dreptunghiular. *Implementare în Matlab*

Funcția Matlab care generează un semnal dreptunghiular este funcția `square`.

Sintaxă: `s = A*square(2*pi*F*t + faza0, q)`

unde:

- `A` = amplitudinea semnalului dreptunghiular
- `F` = frecvența de repetiție a semnalului dreptunghiular (în *Hz*)
- `faza0` = faza inițială a semnalului dreptunghiular (în *radiani*)
- `q` = factor de umplere (de exemplu dacă este 25% se scrie doar 25)
- `t` = vectorul momentelor de timp obținute în urma eșantionării

Vectorul `t` se poate genera cu linia de cod:

```
t = 0:1/Fs:durata;
```

unde:

- `Fs` = frecvența de eșantionare (în *Hz*)
- `durata` = durata semnalului (în *secunde*)

Observații:

- semnalul dreptunghiular generat cu sintaxa de mai sus va avea valorile `A` și `-A`
- dacă faza inițială este nulă (`faza0 = 0`), atunci la momentul de timp inițial semnalul va avea valoarea `+A`
- factorul de umplere (parametrul `q`) este opțional; dacă lipsește se consideră că are valoarea `50`

Să se genereze și să se afișeze în Matlab următorul semnal dreptunghiular.

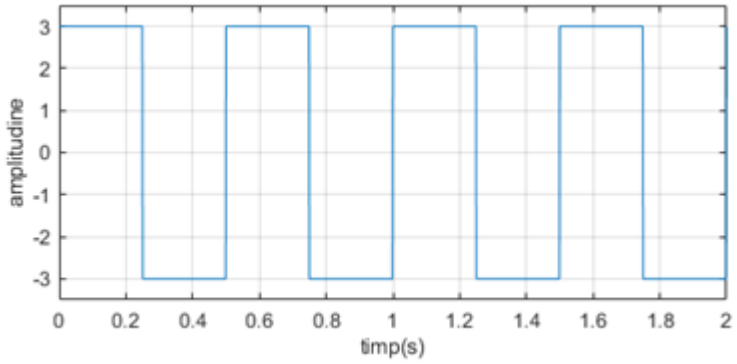


Figura 5.10. Semnalul dreptunghiular

Pentru semnalul de mai sus, parametrii sunt:

- amplitudine `A = 3`
- frecvență `F = 2`Hz (perioada T = 0.5s)
- fază inițială $faza0 = 0\ rad$
- factor de umplere `q` = 50%
- frecvența de eșantionare Fs = 1000Hz (ceea ce înseamnă că Ts = 0.001s, deci într-o perioadă sunt 500 de eșantioane; doar din graficul de mai sus nu putem determina valoarea lui `Fs`, însă încercăm să-i dăm o valoare astfel încât să prindem cât mai multe eșantioane într-o perioadă)
- `durata = 2`s

semnalDreptunghiular.m

```
clc
clear
close all

%% semnal dreptunghiular
% parametrii semnal dreptunghiular:
A = 3;       % amplitudine
F = 2;       % frecventa în Hz
durata = 2;  % durata în secunde
faza0 = 0;   % faza initiala în radiani
Fs = 1000;   % frecventa de esantionare în Hz
duty = 50;   % factor de umplere

% generare semnal dreptunghiular
% vectorul t conține momentele de timp obținute în urma eșantionării cu Fs
t = 0:1/Fs:durata;
s = A*square(2*pi*F*t+faza0,duty);
figure()
plot(t,s)
ylim([min(s)-0.5, max(s)+0.5])
grid
xlabel('timp(s)'), ylabel('amplitudine')
title('semnal dreptunghiular')
```

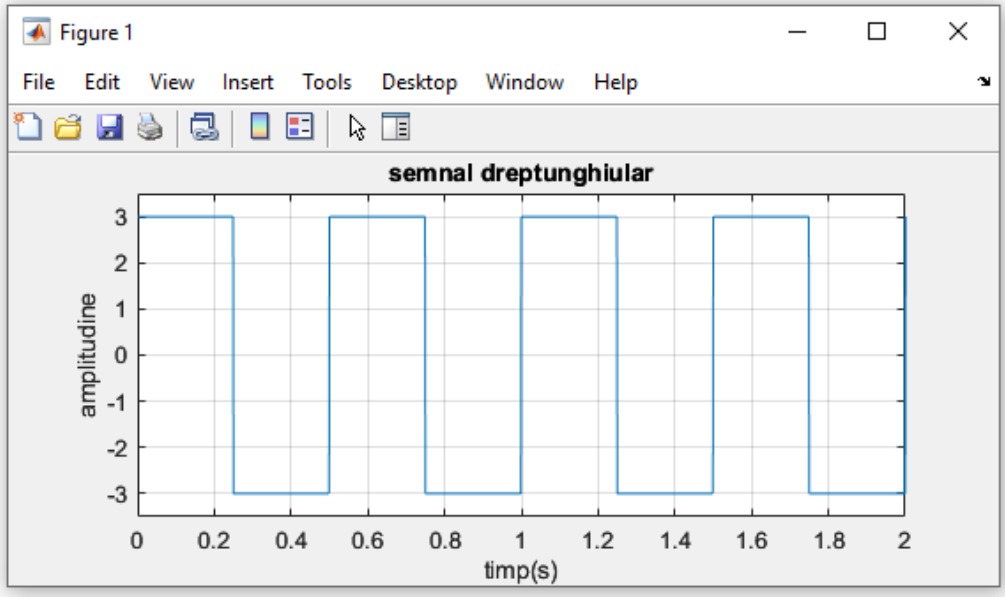


Figura 5.11. Generarea unui semnal dreptunghiular cu Amax = |Amin| = A

📣 Să se genereze și să se afișeze în Matlab un semnal dreptunghiular cu următorii parametri:

- amplitudine minimă `Amin` = -2
- amplitudine maximă `Amax` = 6
- frecvență `F = 3`Hz
- faza inițială $faza0 = 0$ rad
- durata = `2` secunde
- factor umplere `q` = 50%

Observație: Știm că folosind sintaxa `s = A*square(2*pi*F*t+faza0, q)` se generează un semnal dreptunghiular cu valorile `+A` și `-A`. Dacă se dorește ca valorile semnalului să fie `Amin` și `Amax`, cu Amax ≠ |Amin|, atunci raționamentul este următorul:

- se calculează jumătatea distanței de la valoarea maximă la cea minimă

```
amp = (Amax - Amin)/2;
```

- se generează un semnal dreptunghiular cu amplitudinea egală cu `amp`

```
s = amp*square(2*pi*F*t+faza0, q);
```

- se determină componenta continuă cu care trebuie adunat semnalul

```
cc = Amax - amp;
```

- se adună la semnalul dreptunghiular componenta continuă

```
s_drept = s + cc;
```

- Faza inițială deplasează semnalul pe axa O*x*
- Un semnal adunat cu o componentă continuă are ca rezultat deplasarea pe axa O*y*

semnalDreptunghiular_2.m

```
clc
clear
close all

%% semnal dreptunghiular
% parametrii semnalului dreptunghiular:
Amin = -2;      % amplitudine minimă
Amax = 6;       % amplitudine maximă
F = 3;          % frecvența în Hz
durata = 2;     % durata în secunde
faza0 = 0;      % faza inițială în radiani
Fs = 1000;      % frecvența de eșantionare în Hz
q = 50;         % factor de umplere

% generare semnal dreptunghiular
% vectorul t conține momentele de timp obținute în urma eșantionării cu Fs
t = 0:1/Fs:durata;
amp = (Amax - Amin)/2;
cc = Amax - amp;   % cc = componenta continuă
s = amp*square(2*pi*F*t+faza0, q);
s_final = s + cc;
figure()
plot(t,s_final)
ylim([min(s_final)-0.5, max(s_final)+0.5])
grid
xlabel('timp(s)'), ylabel('amplitudine')
title('semnal dreptunghiular')
```

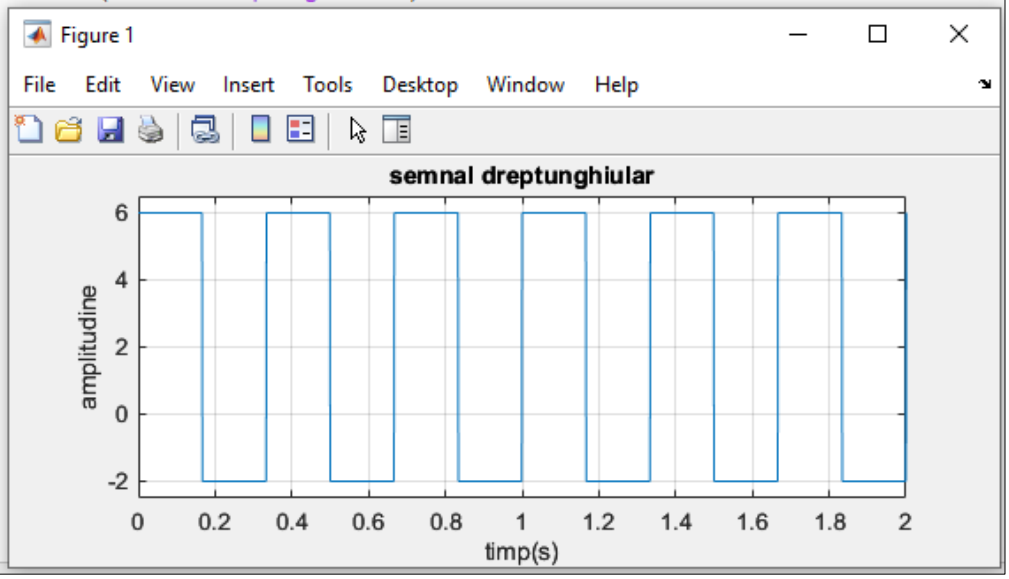
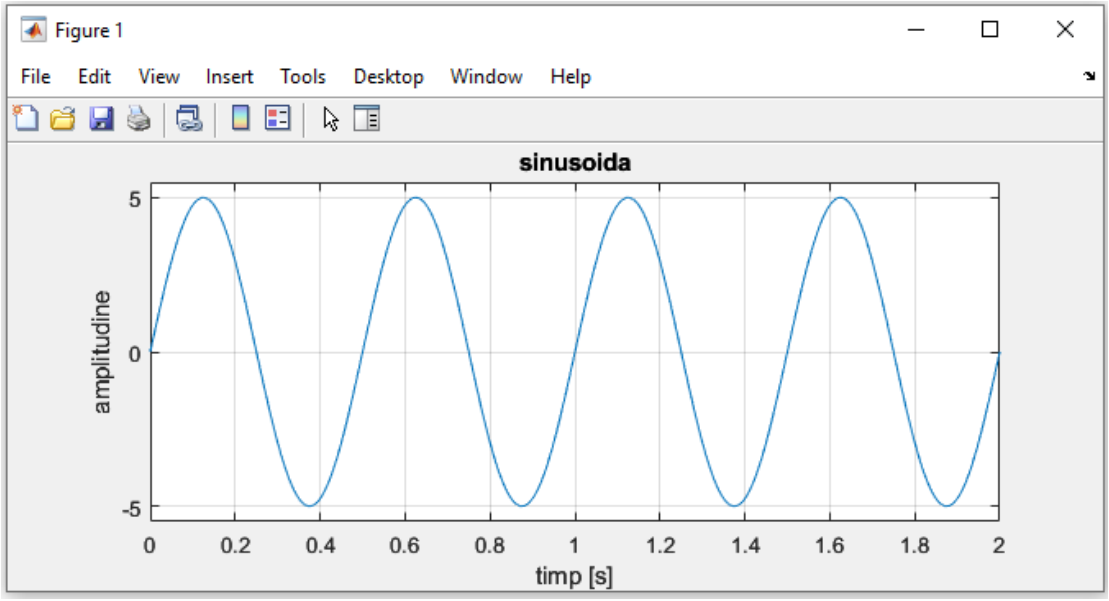


Figura 5.12. Generarea unui semnal dreptunghiular cu valorile `Amax` și `Amin`, Amax ≠ |Amin|

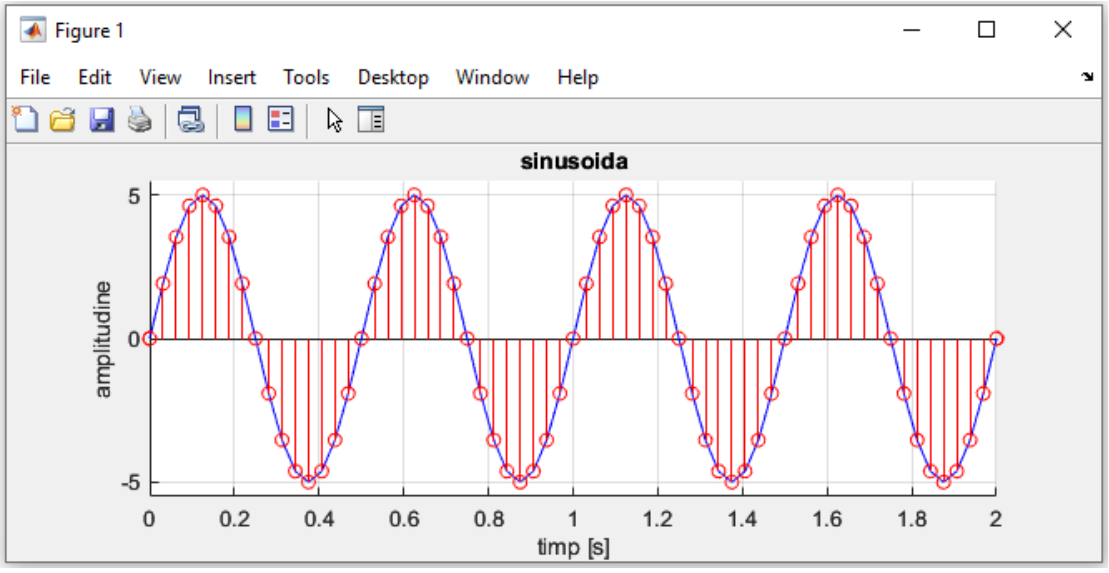
5.7. Aplicații

Aplicația 1. Să se genereze și să se afișeze o sinusoidă urmând pașii:

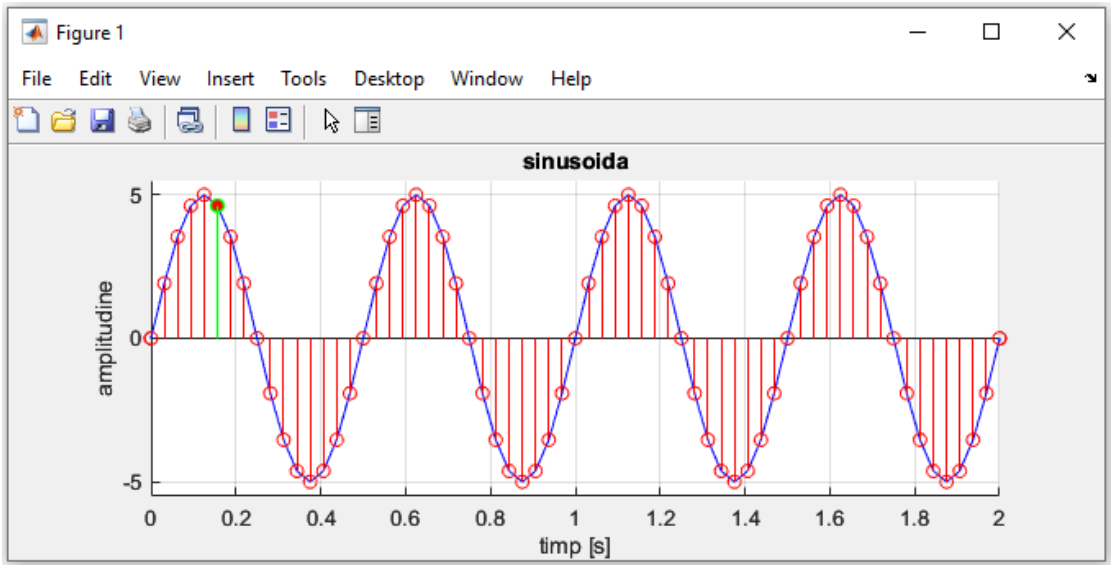
Pas 1. Se generează și se afișează sinusoida de mai jos.



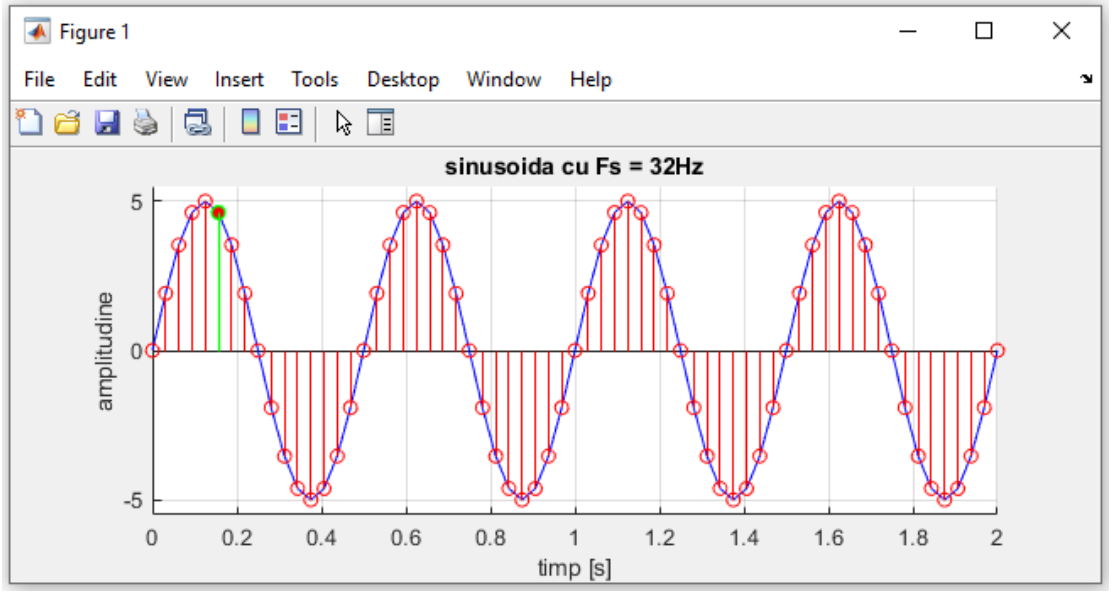
Pas 2. Pentru semnalul generat la *Pas1* să se adauge pe grafic și eșantioanele.



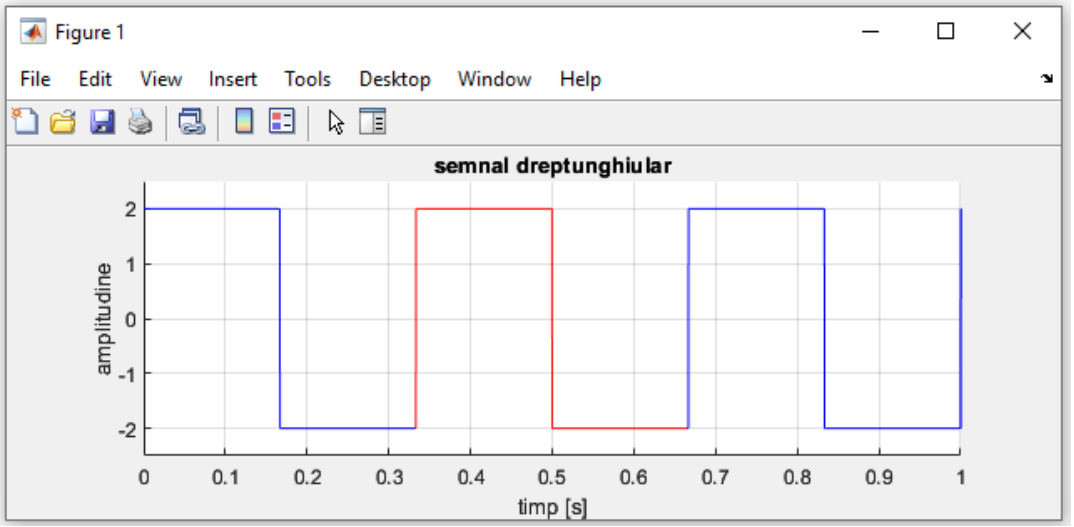
Pas 3. Să se marcheze cu verde eșantionul specificat.



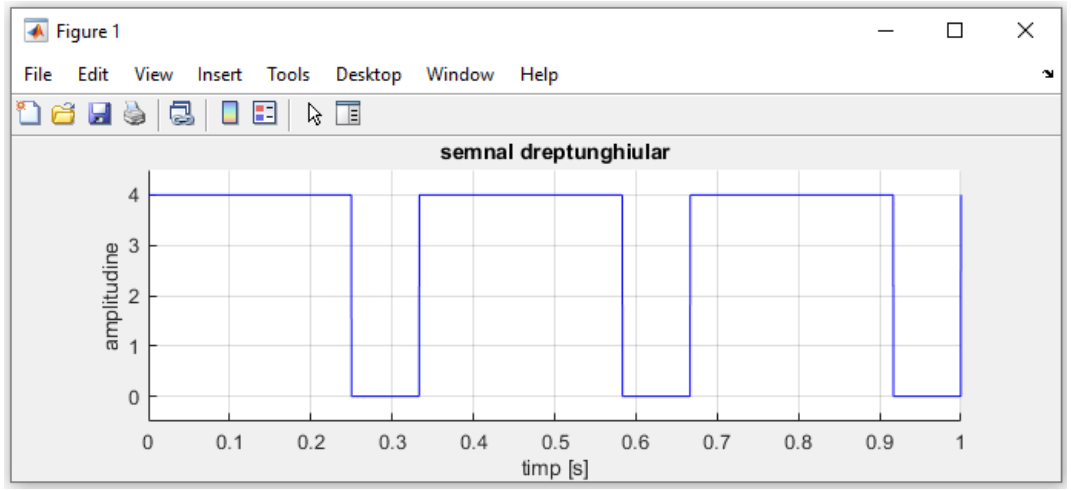
Pas 4. Titlul să fie generat dinamic în funcție de valoarea lui Fs.



Aplicația 2. Să se genereze și să se afișeze următorul semnal dreptunghiular marcând cu roșu a doua perioadă.



Aplicația 3. Să se genereze și să se afișeze următorul semnal dreptunghiular.



Aplicația 4. Fie un semnal sinusoidal `s` cu următorii parametri: frecvența F = 3Hz, frecvența de eșantionare Fs = 1000Hz, durata de 1s, amplitudinea A = 100 și fază inițială $\varphi_0 = 0$.

a) Să se reprezinte grafic semnalul sinusoidal în funcție de timp.

b) Să se copieze în semnalul `z(t)` a 3-a perioadă din semnalul `s(t)` și să se reprezinte grafic.

Capitolul 6
Instrucțiuni decizionale și repetitive

În limbajul 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`).

6.1. Instrucțiunea `if`

Execută un set de instrucțiuni când o expresie este adevărată.

Sintaxă:

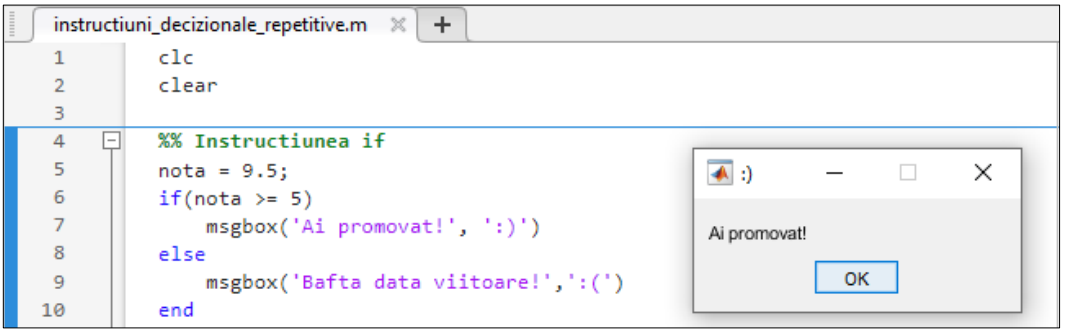
```
if (expresie)
     instrucțiuni_1
else
     instrucțiuni_2
end
```

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

Dacă variabila `nota` are o valoare mai mare sau egală cu 5:

- atunci să se afișeze mesajul "*Ai promovat*!"
- altfel să se afișeze mesajul "*Bafta data viitoare*!"

Afișarea mesajului să se realizeze cu funcția `msgbox`.



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

Sintaxă:

```
if(expresie_1)
      instrucțiuni_1
elseif(expresie_2)
      instrucțiuni_2
else
      instrucțiuni_3
end
```

Folosind funcția `input` să se introducă valoarea variabilei `a` din fereastra *Command Window*. Să se calculeze valoarea variabilei `b` astfel:

$$b = \begin{cases} -1, dacă\ a < 0 \\ 1, dacă\ a > 0 \\ 0,\ dacă\ \ a = 0 \end{cases}$$

instructiuni_decizionale_repetitive.m

```
1   clc
2   clear
3
4   %%  Instructiunea if ... elseif
5   a = input('Introduceti valoarea lui a: ');
6   if(a < 0)
7       b = -1;
8       disp(['b = ', num2str(b)])
9   elseif(a > 0)
10      b = 1;
11      disp(['b = ', num2str(b)])
12  else
13      b = 0;
14      disp(['b = ', num2str(b)])
15  end
16
```

Command Window

```
Introduceti valoarea lui a: 3
b = 1
```

6.2. Instrucțiunea `switch`

Se folosește atunci când se dorește implementarea unor secvențe imbricate mai complexe.

Sintaxă:

```
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 `filtruAles` are valoarea `FTJ` (Filtru Trece Jos) sau `FTS` (Filtru Trece Sus).

instructiuni_decizionale_repetitive.m

```
1   clc
2   clear
3
4   %% Instrucțiunea switch
5   % filtre disponibile: FTJ, FTS
6   % parametrul dat trebuie să fie între apostroafe
7   filtruAles = input('Introduceti filtru: ');
8   switch(filtruAles)
9       case('FTJ')
10           disp('Filtru Trece Jos')
11      case('FTS')
12          disp('Filtru Trece Sus')
13      otherwise
14          disp('Atentie! Filtru necunoscut.')
15  end
```

Command Window

```
Introduceti filtru: 'FTS'
Filtru Trece Sus
```

6.3. Instrucțiunea `for`

Ciclul `for` este utilizat pentru a executa un bloc de cod de mai multe ori.

Sintaxă:

```
for index = val_start : pas : val_stop
          instrucțiuni
end
```

În timpul fiecărei iterații a ciclului `for`, variabila de control (`index`) ia valoarea următoare din vectorul `[val_start : pas : val_stop]` și blocul de `instrucțiuni` este executat cu acea valoare. Dacă variabila `pas` lipsește, se consideră că aceasta are valoarea implicită 1.

Să se ridice la pătrat toate elementele unui vector cu 1 miliar de elemente generate pseudorandom în intervalul [0, 10]. Se va compara timpul de calcul al implementării folosind `for` cu timpul de calcul al implementării fără `for`.

instructiuni_decizionale_repetitive.m

```
1   clc
2   clear
3
4   %% Instructiunea for
5   X = randi([1,10],1,1000000000);
6
7   % implementare fără for
8   tic
9   Y1 = X.^2;
10  disp('timp rulare fără for:')
11  toc
12  disp(newline)
13  % implementare folosind for
14  tic
15  Y2 = zeros(1, length(X));
16  for i=1:length(X)
17      Y2(i) = X(i)^2;
18  end
19  disp('timp rulare folosind for:')
20  toc
21
22
```

Command Window

```
timp rulare fără for:
Elapsed time is 13.864171 seconds.
timp rulare folosind for:
Elapsed time is 28.468734 seconds.
```

Observații:

- pentru a determina timpul de execuție a unei porțiuni de cod se folosește setul de comenzi `tic-toc` astfel: `tic` la începutul codului și `toc` la sfârșitul codului
- așa cum se poate observa în exemplul de mai sus, în Matlab versiunea de implementare folosind ciclul `for` ia mai mult timp la execuție decât varianta fără `for`. În Matlab este de preferat ca atunci când este posibil să se implementeze codul folosind cât mai puțin bucle `for`.

Să se ridice la pătrat doar elementele negative dintr-o matrice $A_{2\times3}$.

instructiuni_decizionale_repetitive.m

```
clc
clear

%% Instrucțiunea for
A = randi([-10,10], 2, 3);
[M, N] = size(A);
for i  = 1 : M %  linia i
    for j = 1 : N %  coloana j
        if(A(i, j) < 0)
            B(i, j) = A(i, j)^2;
        else
            B(i, j) = A(i, j);
        end
    end
end
A
B
```

Command Window

```
A =

     6   -10     8
    -5     2     1


B =

     6   100     8
    25     2     1
```

6.4. Instrucțiunea `while`

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

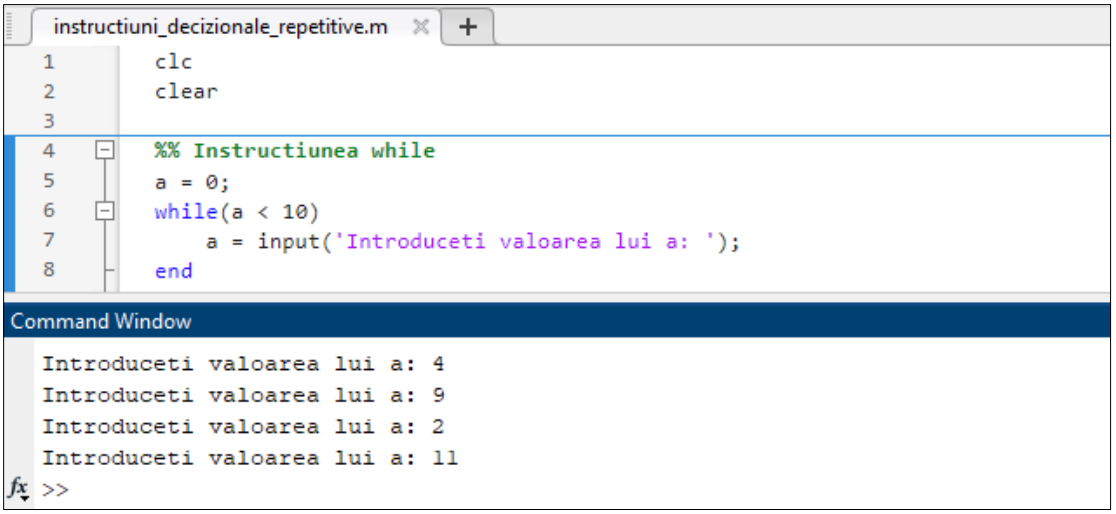
Sintaxă:

```
while (expresie)
      instrucțiuni
end
```

Blocul de `instrucțiuni` dintre `while` și `end` este executat repetat atâta timp cât `expresie` este adevărată.

Este important să se modifice variabilele din condiția ciclului `while` în așa fel încât să existe o cale de ieșire din ciclu pentru a evita bucle infinite.

Să se introducă valoarea unui număr **a** din *Command Window* atâta timp cât **a**<10.



Observații:

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

6.5. Aplicații

Aplicația 1. Fie variabila `numar` a cărei valoare se introduce din fereastra *Command Window*, cu ajutorul funcției `input`.

În funcție de paritatea variabilei `numar`, să se afișeze în *Command Window* mesajul "*numar par*" sau "*numar impar*".

Aplicația 2. Să se genereze pseudorandom un vector linie `X` cu `N = 5` valori reale în intervalul (0, 1). Să se calculeze vectorul `Y` după formula matematică:

$$Y(i) = \begin{cases} 1, \ daca\ X(i) > 0.5 \\ 0, \ daca\ X(i) = 0.5 \\ -1, \ daca\ X(i) < 0.5 \end{cases} \quad \text{pentru } i = 1 \ldots N$$

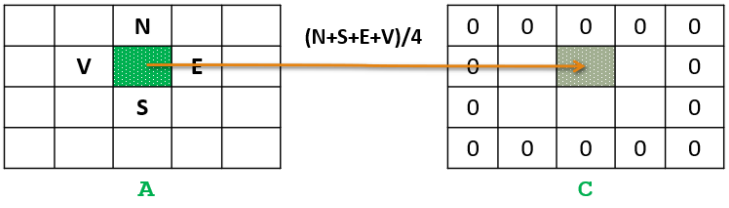
Aplicația 3. Să se genereze pseudorandom o matrice `A` cu `M = 4` linii și `N = 5` coloane cu valori întregi în intervalul [0, 10].

a) Să se calculeze matricea `B` după formula:

$$B(i,j) = \begin{cases} 1, & \text{daca } A(i,j) \geq 5 \\ -1, & \text{daca } A(i,j) < 5 \end{cases}, \text{ pentru } i = 1 \ldots M, \ j = 1 \ldots N$$

b) Să se calculeze în variabila `sumaImpare`, suma elementelor impare din `A`

c) Să se genereze matricea `C` (de aceeași dimensiune cu `A`) care să conțină media vecinilor elementelor din `A`



Aplicația 4. Fie o matrice `A` cu `M = 5` linii și `N = 5` coloane, cu valori întregi generate pseudorandom, cu distribuție uniformă în intervalul [0, 10].

a) Să se calculeze matricea `B` după formula:

$$B(i,j) = \begin{cases} 0, \ dacă \ A(i,j) \leq 3 \\ 1, \ dacă \ 3 < A(i,j) \leq 7 \\ 2, \ daca \ A(i,j) > 7 \end{cases} \text{ unde } i = 1...M, j = 1...N$$

b) Să se salveze în vectorul coloană C toate elementele din `A` mai mari de 5.

c) Să se salveze în variabila D numărul de elemente din `A` mai mari de 5.

Aplicația 5. Să se introducă de la tastatură și să se salveze în vectorul `Z` numere atâta timp cât acestea sunt în intervalul [0,10].

Aplicația 6. Fie un vector linie `X` cu `N = 101` elemente numere întregi generate pseudorandom în intervalul [-10, 10].

a) Să se salveze in variabila `valMedie` media elementelor din `X` de pe pozițiile pare (cu index par).

b) Între elementele de pe pozițiile 10 și 11 din `X` să se insereze 100 de valori de 3. Rezultatul să se salveze în vectorul `extraX`.

c) Să se construiască vectorul linie `Z` astfel:

$$Z(i) = \begin{cases} 1, \ daca \ (X(i) - medie) > 0 \\ 0, daca \ (X(i) - medie) = 0 \\ -1, \ daca \ (X(i) - medie) < 0 \end{cases} \quad pentru \ i = 1 \ldots N$$

unde *medie* reprezintă media tuturor elementelor din `X`.

d) Să se salveze în variabila `suma`, suma elementelor din `X` strict mai mari decât 5

e) Să se genereze vectorul `Y` care să conțină elementele negative din `X` ridicate la puterea a 2-a și apoi ordonate crescător.

Aplicația 7. Fie un vector coloană `X` cu `N = 21` elemente numere întregi generate pseudorandom în intervalul [0, 100].

a) Să se salveze în variabila `valMedie` media celor mai mari 8 elemente din `X`

b) Să se construiască vectorul linie `Z` astfel:

$$Z(i) = \begin{cases} X(i), \ daca \ X(i) < X(i+1) \\ 0, in \ rest \end{cases} \quad \text{pentru } i = 1 \ldots N - 1$$

c) Să se salveze în variabila `multipli`, câți multipli ai lui 5 sunt în vectorul `X`

Aplicația 8. Fie o matrice `X`, având 5 linii și 5 coloane, cu valori întregi pseudorandom distribuite uniform în intervalul [-5, 5].

a) Să se calculeze suma elementelor aflate pe diagonala principală din `X`.

b) Să se calculeze suma elementelor aflate strict deasupra diagonalei principale din `X`.

c) Să se calculeze suma elementelor aflate strict sub diagonala principală din `X`.

d) Să se calculeze suma elementelor aflate pe diagonala secundară din `X`.

Aplicația 9. Fie o matrice `T` cu 5 linii și 5 coloane, cu valori întregi generate pseudorandom, cu distribuție uniformă în intervalul [-5, 5].

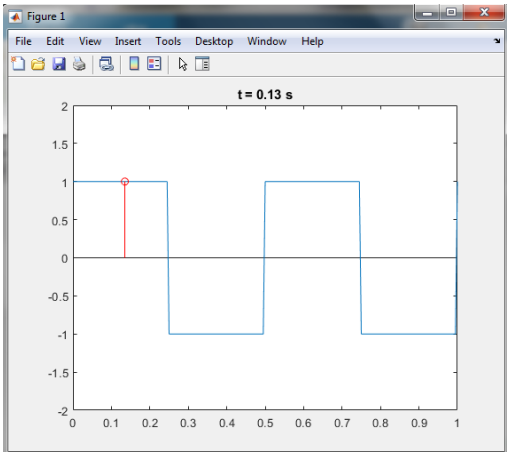
a) Să se scrie codul Matlab care să calculeze matricea `U` după formula:

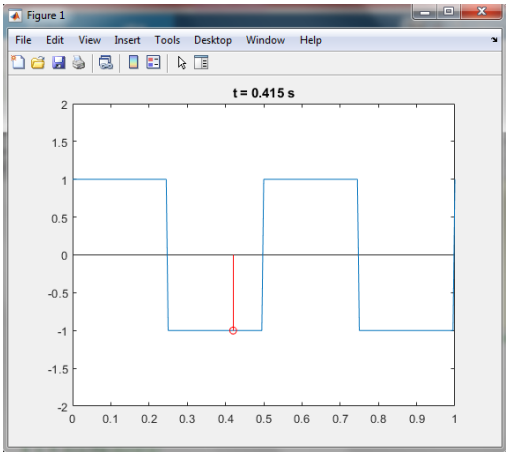
$$U(i,j) = \begin{cases} 1, dacă: \ (T(i,j) \geq medieDiag) şi (T(i,j) \in diagonalei \ principale) \\ 2, dacă: \ (T(i,j) < medieDiag) şi (T(i,j) \in diagonalei \ principale) \\ 0, \ în \ rest \end{cases}$$

pentru $i, j = 1 \ldots 5$, unde `medieDiag` reprezintă media elementelor de pe diagonala principală din `T`.

b) Elementul de pe poziția centrală din matricea `T` să fie înlocuit cu suma tuturor elementelor din `T`. Matricea rezultată să fie salvată în variabila `Tmodif`.

Aplicația 10. Să se genereze un semnal dreptunghiular și să se afișeze rând pe rând eșantioanele sale. Titlul să se genereze dinamic și să conțină momentul de timp al fiecărui eșantion. Se cunoaște Fs = 1000Hz.





Aplicația 11. Fie un semnal dreptunghiular x cu următorii parametri: Fs = 1000Hz, durata = 1s, A = 100, $\varphi_0 = 0$, F = 2Hz. Să se genereze semnalul y după regula:

$$y(n) = \begin{cases} 1, dacă\ x(n) \geq 1 \\ -1, dacă\ x(n) \leq -1 \\ x(n), altfel \end{cases}$$

unde $n = 1 \ldots N$, N = numărul de eșantioane din y.

Să se reprezinte grafic semnalele $x(t)$ și $y(t)$.

Aplicația 12. Să se genereze semnalul u format din suma a N sinusoide, astfel:

$$u(t) = \sum_{k=1}^{N} S_k \cdot \sin(2\pi \cdot k \cdot F \cdot t)$$

unde F = 2Hz, durata fiecărei sinusoide este de 2 secunde, Fs = 1000Hz și amplitudinile sinusoidelor sunt:

$$S_k = \begin{cases} \frac{4}{k\pi}, & pentru\ k\ impar \\ 0, & pentru\ k\ par \end{cases}$$

Să se afișeze:

- semnalul $u(t)$ rezultat din suma a N = 5 sinusoide
- semnalul $u(t)$ rezultat din suma a N = 500 sinusoide

Capitolul 7
Lucrul cu fișiere externe

În acest capitol se dorește o scurtă prezentare a modului în care se poate lucra în Matlab cu imagini, semnale audio, fișiere de tip text și Excel.

7.1. Lucrul cu imagini în Matlab

Așa cum s-a mai specificat deja, limbajul Matlab este orientat pentru lucrul cu structuri matriceale. Astfel, o imagine grayscale digitală este reprezentată ca o matrice `A` de `M` linii si `N` coloane:

$$A = \begin{bmatrix} a_{1,1}\ a_{1,2}\ a_{1,3} \dots\ a_{1,N} \\ a_{2,1}\ \ a_{2,2}\ a_{2,3} \dots\ a_{2,N} \\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots. \\ a_{M,1}\ a_{M,2}\ a_{M,3} \dots\ a_{M,N} \end{bmatrix}$$

Elementul de bază al imaginii digitale se numește *pixel*. Un pixel `(a`$_{i,j}$`)` este definit prin coordonatele sale spațiale (linia `i`, coloana `j`) și valoarea luminanței (valoarea nivelului de gri) asociată coordonatelor `i` și `j`. De exemplu, o imagine *grayscale* având dimensiunea de 200 x 300 pixeli nu este altceva decât o matrice cu 200 de linii și 300 de coloane. Dacă imaginea este color, atunci matricea în care este stocată va avea 3 straturi (pentru planurile de culoare Roșu, Verde, Albastru).

7.1.1. Citirea unei imagini

Funcția `imread` permite citirea imaginilor și salvarea acestora într-o matrice.

Sintaxă: `A = imread('nume_imagine.ext')`

Observații:

- parametrul `'nume_imagine.ext'` reprezintă numele imaginii, incluzând și extensia care poate fi `jpg`, `png`, `bmp`, `tiff` etc
- 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 straturile de culoare Roșu, Verde, Albastru

7.1.2. Afișarea unei imagini

Funcția `imshow` permite afișarea conținutului unei matrice ca o imagine, asociind un nivel de gri fiecărei valori din matrice.

Sintaxă: `imshow(A)`

Observații:

- Variabila `A` este matricea care se dorește să fie afișată ca imagine
- Pentru ca funcția `imshow` să afișeze corect o imagine, trebuie să se respecte corespondența dintre tipul de date al variabilei în care este stocată imaginea și intervalul de valori al intensității pixelilor din imagine.
- Dacă se dorește să se schimbe tipul de date al unei variabile în care este stocată o imagine se poate folosi una dintre funcțiile următoare, depinzând de situație.

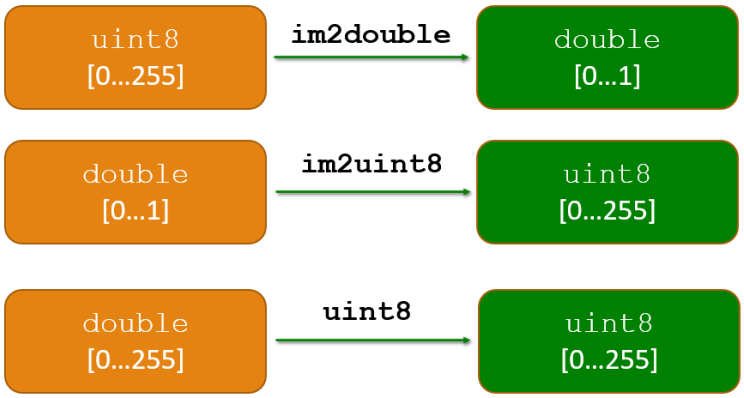


Figura 7.1. Tipuri de date utile în Matlab pentru lucrul cu imagini

7.1.3. Salvarea unei imagini

Salvarea unei matrice ca o imagine se realizează cu funcția `imwrite`.

Sintaxă: `imwrite(A, 'nume_imagine.ext')`

Observații:

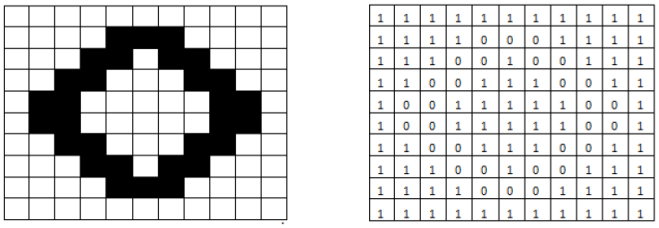
- Se salvează matricea `A` ca imagine cu numele `nume_imagine.ext`
- Dacă numele imaginii conține și calea către un folder, atunci imaginea va fi salvată în acel folder; dacă nu, imaginea va fi salvată în folderul curent
- Dacă se dorește ca valorile pixelilor dintr-o imagine să nu fie distorsionate în urma salvării, se folosesc extensiile *png* (PNG - *Portable Network Graphics*) sau *tiff* (TIFF - Tagged Image File Format). Formatul *jpeg* (JPEG - *Joint Photographic Experts Group*) în care sunt salvate majoritatea imaginilor este un fomat care realizează compresie.

7.1.4. Tipuri de imagini

7.1.4.1. Imagine alb-negru

O imagine alb-negru este o imagine cu pixeli care pot fi albi sau negri. În funcție de tipul de date al variabilei în care este stocată imaginea, în Matlab putem avea:

- tip de date `double` ➔ doar valorile 0 (negru) și 1 (alb)
- tip de date `uint8` ➔ doar valorile 0 (negru) și 255 (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

a b

Figura 7.2. a) imagine binară b) reprezentarea matriceală a imaginii binare

Să se genereze o imagine cu 100 de linii și 100 de coloane care să conțină în jumătatea de sus doar pixeli albi, iar în jumătatea de jos doar pixeli negri.

```
%% imagine alb - negru
% generare imagine: jumatate alba-jumatate neagra
W = ones(50,100);  % jumatatea gri a imaginii
B = zeros(50,100); % jumatatea neagra a imaginii
imagBW = [W; B];   % concatenarea celor doua jumatatie
figure('Color', [0.8 0.8 0.8]) % background gri
imshow(imagBW), title('imagine alb-negu')
```

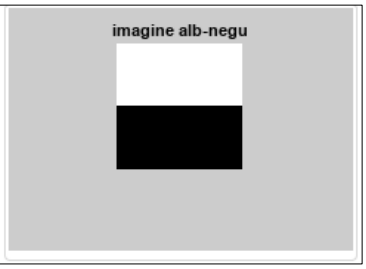


Figura 7.3. Generarea unei imagini *alb - negru*

7.1.4.2. Imagine grayscale

Imaginile cu niveluri de gri (imagini *grayscale*) pot fi 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.

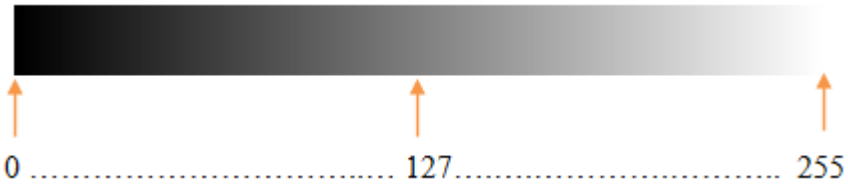
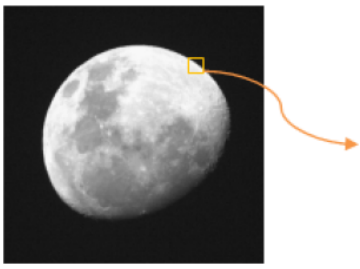


Figura 7.4. Cele 256 niveluri de gri

- tip de date `double` ➔ valori reale în intervalul [0, 1]
- tip de date `uint8` ➔ valori naturale în intervalul [0, 255]



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 7.5. Exemplu de imagine grayscale

Fie o imagine grayscale cu 50 de linii și 256 de coloane. Pe fiecare coloană imaginea conține câte un nivel diferit de gri, de la negru la alb. Să se scrie codul Matlab care generează și afișează imaginea grayscale.

```
1  %% imagine grayscale
2  NrLinii = 50;
3  ValoriPeLinie = 0:255;
4  imagGray = repmat(ValoriPeLinie,NrLinii,1);
5  imagGray = uint8(imagGray);
6  figure(), imshow(imagGray)
7  title('imagine grayscale')
```

Figura 7.6. Generarea unei imagini *grayscale*

7.1.4.3. Imagini color

Conținutul unei imagini digitale color în format RGB reprezintă o matrice tridimensională cu trei straturi (stratul de roșu-*Red*, stratul de verde-*Green* și stratul de albastru-*Blue*) fiecare strat fiind o matrice cu aceeași dimensiune cu cea a imaginii.

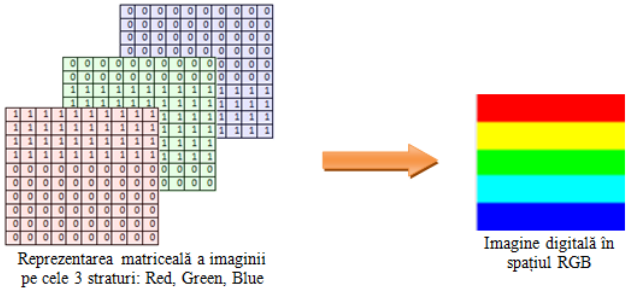


Figura 7.7. Reprezentarea matriceală a unei imagini digitale color

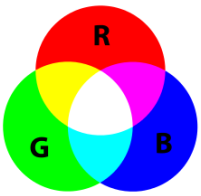


Figura 7.8. Aditivitatea culorilor

Observație: dacă cele trei straturi de culoare sunt identice, atunci imaginea va arăta precum o imagine grayscale.

- tip de date `double` ➔ valori reale în intervalul [0, 1]
- tip de date `uint8` ➔ valori naturale în intervalul [0, 255]

Tabel 7.1. Principalele culori și codurile acestora

Culoare	Cod culoare `double`	Cod culoare `uint8`	Reprezentare grafică
red	[1, 0, 0]	[255, 0, 0]	
green	[0, 1, 0]	[0, 255, 0]	
blue	[0, 0, 1]	[0, 0, 255]	
cyan	[0, 1, 1]	[0, 255, 255]	
magenta	[1, 0, 1]	[255, 0, 255]	
yellow	[1, 1, 0]	[255, 255, 0]	
black	[0, 0, 0]	[0, 0, 0]	
white	[1, 1, 1]	[255, 255, 255]	

<table>
<tr><th>Valori pixel</th><th>Afișare</th><th>Observații</th></tr>
<tr><td colspan="3">Generare pătrat roșu</td></tr>
<tr><td>

```
pixel = uint8(zeros(50,50,3));
pixel(1:50,1:50,1) = 255;
pixel(1:50,1:50,2) = 0;
pixel(1:50,1:50,3) = 0;
imshow(pixel)
```

</td><td>  </td><td>Pătrat roșu
dimensiune:50x50x3
tip date: <code>uint8</code></td></tr>
<tr><td colspan="3">Generare pătrat magenta</td></tr>
<tr><td>

```
pixel = uint8(zeros(50,50,3));
pixel(1:50,1:50,1) = 255;
pixel(1:50,1:50,2) = 0;
pixel(1:50,1:50,3) = 255;
imshow(pixel)
```

</td><td>  </td><td>Pătrat magenta
dimensiune:50x50x3
tip date: <code>uint8</code></td></tr>
<tr><td colspan="3">Generare pătrat gri</td></tr>
<tr><td>

```
pixel = uint8(zeros(50,50,3));
pixel(1:50,1:50,1) = 155;
pixel(1:50,1:50,2) = 155;
pixel(1:50,1:50,3) = 155;
imshow(pixel)
```

</td><td>  </td><td>Pătrat gri
dimensiune:50x50x3
tip date: <code>uint8</code></td></tr>
<tr><td colspan="3">Generare pătrat galben</td></tr>
<tr><td>Varianta greșită!

```
pixel = uint8(zeros(50,50,3));
pixel(1:50,1:50,1) = 1;
pixel(1:50,1:50,2) = 1;
pixel(1:50,1:50,3) = 0;
imshow(pixel)
```

</td><td>  </td><td>Pătrat aproape negru
dimensiune:50x50x3
tip date: <code>uint8</code></td></tr>
<tr><td colspan="3">Dacă dorim să afișăm un pătrat galben, atunci fie schimbăm tipul de date să fie <code>double</code>, fie schimbăm codul de culoare să fie: [255, 255, 0] și tipul de date rămâne <code>uint8</code>.</td></tr>
<tr><td>Varianta corectă!

```
pixel = double(zeros(50,50,3));
pixel(1:50,1:50,1) = 1;
pixel(1:50,1:50,2) = 1;
pixel(1:50,1:50,3) = 0;
imshow(pixel)
```

</td><td></td><td>Pătrat gri
dimensiune:50x50x3
tip date: <code>double</code></td></tr>
</table>

Figura 7.9. Exemple de generare imagini color și afișare cu `imshow`

7.1.5. Transformări simple ale imaginilor

7.1.5.1. Transformarea din imagine color în imagine grayscale

Nivelul de gri corespunzător culorii unui pixel se obţine prin aducerea la aceeaşi intensitate a celor trei componente ale culorii pixelului respectiv (*R* - roşu, *G* - verde, *B* - albastru). Procedeul se aplică pentru toți pixelii din imagine.

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Observaţie: începând cu versiunea `R2020b`, în Matlab există funcția `im2gray` pentru conversia unei imagini din color în grayscale.

Să se realizeze conversia din color în grayscale folosind formula de mai sus.

aplicatii_imagini.m

```
clc
clear
close all

imag = imread('semnale/casuta.jpg');
% formula pentru conversie I = 0.299 · R + 0.587 · G + 0.114 · B
stratR = imag(:,:,1);
stratG = imag(:,:,2);
stratB = imag(:,:,3);
imagGray = 0.299*stratR + 0.587*stratG + 0.114*stratB;
figure()
subplot(1,2,1), imshow(imag), title('Imagine color')
subplot(1,2,2), imshow(imagGray), title('Imagine grayscale')
```

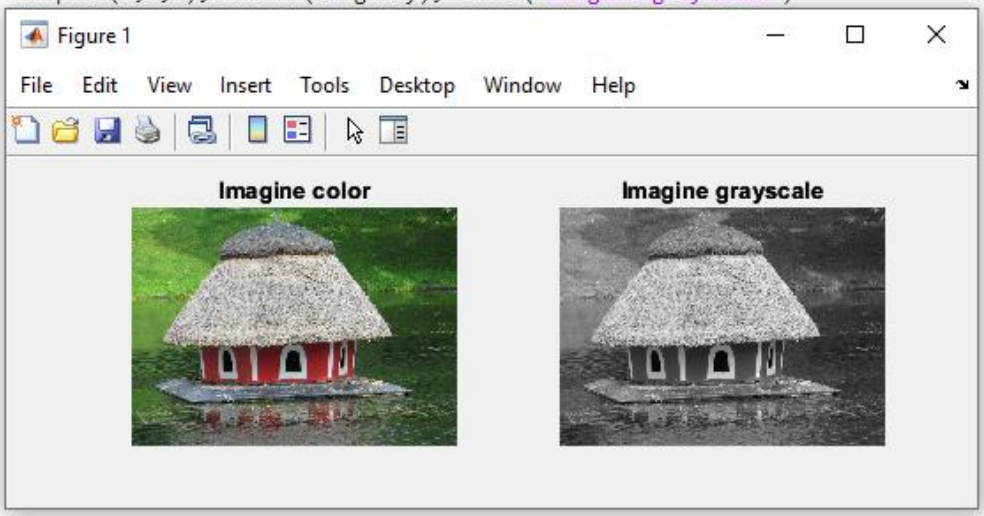


Figura 7.10. Transformarea unei imagini color într-o imagine grayscale

7.1.5.2. Complementul unei imagini (imaginea negativă)

Complementul unei imagini binare

Complementul I_c al unei imagini binare I se calculează astfel:

$$I_C = 1 - I$$

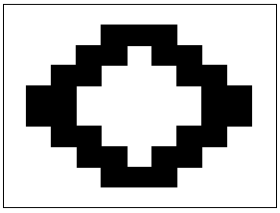
În cazul unei imagini binare, prin negativare se inversează albul cu negru.

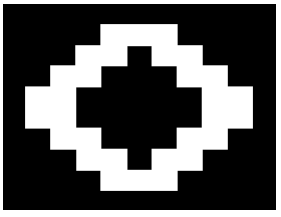
Exemple:

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

$Ic = 1 - I$

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





a) Imagine binară b) Complementul imaginii binare

Figura 7.11. Negativa (complementul) unei imagini binare

Complementul unei imagini grayscale

Complementul I_C al unei imagini grayscale I pe 8 biți se calculează astfel:

$$I_C = 255 - I$$

În imaginea finală (imaginea negativă) zonele întunecate devin mai luminoase iar zonele mai luminoase devin mai întunecate.

Exemple:

100	120	200	255
50	100	0	200
50	150	255	0

$Ic = 255 - I$

155	135	55	0
205	155	255	55
205	105	0	255

a) Imaginea originală b) Negativa imaginii

Figura 7.12. Negativa (complementul) unei imagini grayscale

Să se realizeze complementul unei imagini grayscale.

aplicatii_imagini.m

```
clc
clear
close all

%% Complementul unei imagini grayscale
imag = imread('semnale/luna.jpg');
imag = im2double(imag);
imagComplement = 1 - imag;
figure()
subplot(1,2,1), imshow(imag), title('Imagine originală')
subplot(1,2,2), imshow(imagComplement), title('Imagine complement')
```

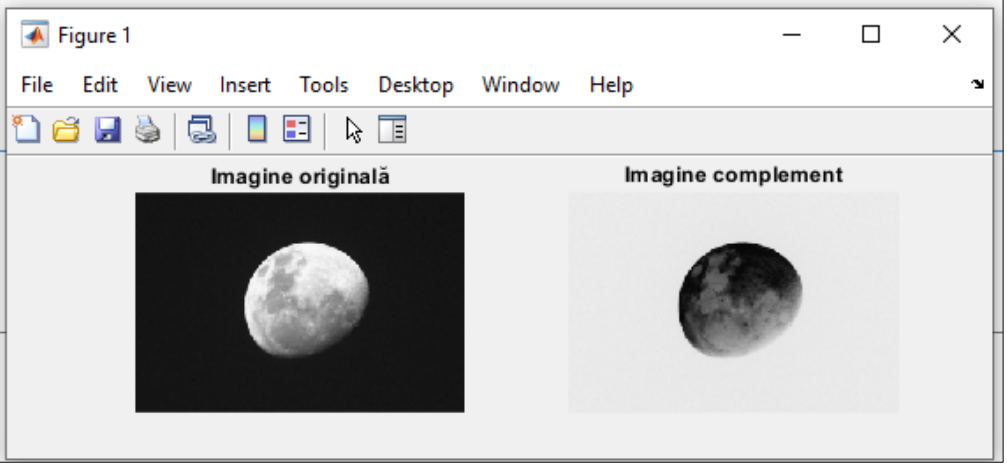


Figura 7.13. Negativa (complementul) unei imagini grayscale

- **Complementul unei imagini color**

Pentru o imagine color RGB pe 8 biți, complementul se realizează pentru fiecare strat în parte, astfel:

$$Rc = 255 - R, Gc = 255 - G, Bc = 255 - B$$

a) Imagine originală B) Imaginea negativă

Figura 7.14. Complementul (imaginea negativă) a unei imagini color

7.2. Lucrul cu semnale audio în Matlab

7.2.1. Citirea unui semnal audio

Citirea unui semnal audio se realizează cu funcția `audioread`.

Sintaxă: `[Y, Fs] = audioread('nume_fișier.ext')`

- `'nume_fișier.ext'` = numele fișierului audio ce se dorește a se citi, , incluzând și extensia care poate fi *wav, mp3, mp4* etc
- `Y` = eșantioanele semnalului audio
- `Fs` = frecvența de eșantionare a semnalului audio

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`.

7.2.2. Scrierea unui semnal audio

Scrierea fișierelor de tip audio se poate realiza folosind funcția `audiowrite`.
Sintaxă: `audiowrite('nume_fisier.ext', Y, Fs)`

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

7.2.3. 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
- `Fs` este frecvența de eșantionare a semnalului audio

Să se citească un semnal audio.

a) Să se calculeze durata semnalului în secunde.

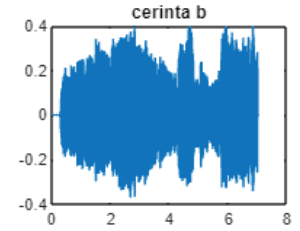
b) Să se reprezinte semnalul în domeniul timp.

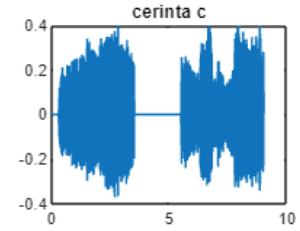
c) Să se adauge 2 secunde de liniște la mijlocul semnalului audio.

d) Să se marcheze cu verde cea de-a doua secundă din semnalul audio.

```
clc
clear
close all
% citire semnal audio
[y,Fs] = audioread('semnale/semnalAudio.mp4');

% a) Să se calculeze durata semnalului în secunde
durata = (length(y)-1)/Fs;
disp("durata = " + durata + "sec")

% b) Să se reprezinte semnalul în domeniul timp
t = 0:1/Fs:durata;
figure()
plot(t,y), title('cerinta b')

% c) Să se adauge 2 secunde de liniște
% la mijlocul semnalului audio
jum = round(length(y)/2);
primaJum = y(1:jum);
aDouaJum = y(jum+1:end);
liniste = zeros(1, 2*Fs)';
yCuLiniste = [primaJum;liniste;aDouaJum];
durataCuLiniste = (length(yCuLiniste)-1)/Fs;
tCuLiniste = 0:1/Fs:durataCuLiniste;
figure()
plot(tCuLiniste, yCuLiniste), title('cerinta c')

% d) Să se marcheze cu verde cea de-a
% doua secundă din semnalul audio
figure()
hold on
    plot(t,y,'b')
    plot(t(Fs:2*Fs),y(Fs:2*Fs),'g')
hold off
title('cerinta d')
```

```
durata = 7.0124sec
```





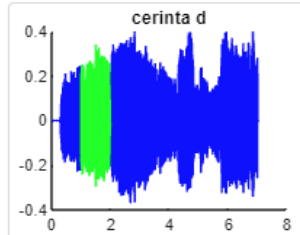


Figura 7.15. Lucrul cu semnale audio în Matlab

7.3. Lucrul cu fișiere text în Matlab

Lucrul cu fișiere **.txt* se dovedește foarte util atunci când se dorește de exemplu salvarea rezultatelor diverselor simulări în Matlab.

7.3.1. Deschiderea unui fișier **.txt*

Sintaxă: `fileID = fopen(FILENAME,PERMISIUNE)`

Deschide fișierul cu numele `FILENAME` în modul specificat de `PERMISIUNE`.

Tabel 7.2. Principalele tipuri de permisiuni la deschiderea unui fișier **.txt*

Tip permisiune	Descriere
`'r'`	Deschis pentru citit.
`'w'`	Deschis pentru scriere; se șterge ce era scris anterior în fișier.
`'a'`	Deschiderea sau crearea unui fișier pentru scriere. Se adaugă datele la sfârșitul fișierului.
`'r+'`	Deschiderea unui fișier pentru citire și scriere.
`'w+'`	Deschiderea sau crearea unui fișier pentru citire și scriere. Se șterge ce era scris anterior în fișier.
`'a+'`	Deschiderea sau crearea unui fișier pentru citire și scriere. Se adaugă datele la sfârșitul fișierului.

7.3.2. Scrierea într-un fișier **.txt*

Sintaxă: `fprintf(fileID, FORMAT, date)`

unde:

- `fileID` = identificatorul fișierului, obținut de obicei cu funcția `fopen`
- `FORMAT` = specifică tipul datelor ce vor fi scrise în fișier. Cele mai uzuale tipuri sunt cele din tabelul următor.

Tip de date	FORMAT
întregi cu semn	`'%d'` sau `'%i'`
întregi fără semn	`'%u'`
double	`'%f'`
șiruri de caractere	`'%s'`

Observații:

- dacă se dorește trecerea pe o nouă linie se folosește caracterul special `\n`, după tipul de `FORMAT`, dar între aceleași apostroafe. *Exemplu*: `'%s\n'`.
- dacă se dorește ca o valoare numerică să fie scrisă într-un anumit format (cu un număr impus de cifre pentru partea întreagă și cea zecimală), acest lucru se specifică imediat înaintea literei de la tipul de `FORMAT`. *Exemplu*: `'%2.3f'` specifică faptul că se va scrie o valoare numerică de tip `double`, cu 2 cifre la partea întreagă și 3 zecimale.
- după terminarea scrierii într-un fișier text, trebuie ca acesta să fie închis cu funcția `fclose(fileID)`.

Să se scrie un program care să genereze 5 numere pseudorandom în intervalul (0...1) și să le salveze în fișierul *dateRandom.txt*. Salvarea numerelor se va face cu 2 zecimale.

fisiere_txt.m

```
1   clc
2   clear
3   fileID = fopen('dateRandom','w+');
4   for i = 1:5
5       mesaj = "iteratia " + i + ": ";
6       nr = rand();
7       fprintf(fileID,'%s',mesaj);
8       fprintf(fileID,'%1.2f\n',nr);
9   end
10  fclose(fileID);
```

Figura 7.16. Scrierea datelor într-un fișier text

Conținutul fișierului *dateRandom.txt* obținut în urma rulării codului de mai sus este:

dateRandom

```
1   iteratia 1: 0.95
2   iteratia 2: 0.21
3   iteratia 3: 0.71
4   iteratia 4: 0.24
5   iteratia 5: 0.12
```

Figura 7.17. Cele 5 valori generate pseudorandom

7.4. Lucrul cu fișiere Excel în Matlab

Înainte de a începe prezentarea modului în care se poate lucra cu un fișier Excel în Matlab, se va prezenta mai întâi tipul de date `cell`, deoarece fiecare celulă dintr-un fișier Excel încărcat în Matlab va avea acest tip de date.

7.4.1. Tipul de date `cell`

O variabilă de tip `cell` conține mai multe celule, fiecare celulă putând fi de orice tip de date. Pentru a defini o variabilă de tip `cell`, conținutul ei trebuie scris între acolade.

Sintaxă: `varCell = {var1, var2, … , varN}`

unde `varN` este o variabilă de orice tip de date

Să se definească o variabilă `data` de tip `cell` care să conțină un scalar, un vector, o matrice și un șir de caractere.

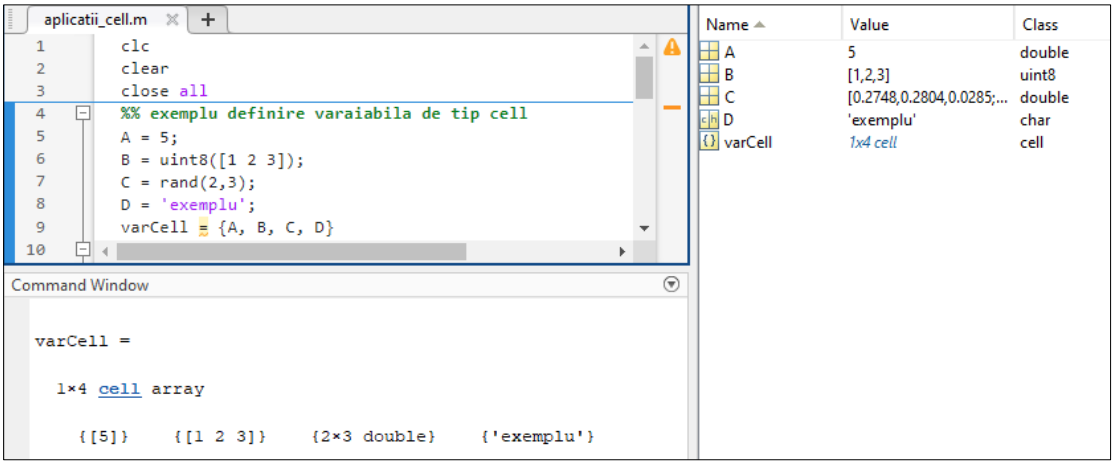


Figura 7.18. Generarea variabilelor de dip `cell`

Observații:

- Pentru a accesa un element al variabilei de tip `cell`, **ca celulă**, se folosește sintaxa: `varCell(index)`
- Pentru a accesa **conținutul unei celule** dintr-o variabilă de tip `cell`, se folosește sintaxa: `varCell{index}`

Să se acceseze al doilea element al variabilei `varCell` de la aplicația anterioară. Elementul va fi accesat atât ca celulă cât și ca conținut.

aplicatii_cell.m

```
1   clc
2   clear
3   close all
4   %% exemplu definire varaiabila de tip cell
5   A = 5;
6   B = uint8([1 2 3]);
7   C = rand(2,3);
8   D = 'exemplu';
9   varCell = {A, B, C, D};
10  % accesare elementului ca celula
11  valoareCelula = varCell(2)
12  % accesare ca si continut
13  continutCelula = varCell{2}
```

Command Window

```
valoareCelula =

  1×1 cell array

    {[1 2 3]}


continutCelula =

  1×3 uint8 row vector

   1   2   3
```

Figura 7.19. Lucrul cu variabile de dip `cell`

Pentru a converti o variabilă de tip `cell` într-un vector sau matrice se folosește funcția `cell2mat`.

Sintaxă: `A = cell2mat(C)`

convertește o variabilă `C` de tip `cell` într-un vector/matrice `A`

7.4.2. Citirea datelor dintr-un fișier Excel

În continuare se va lucra cu fișierul Excel cu numele *Regiuni.xlsx*. Conținutul acestuia arată astfel:

	A	B	C
1	Provincia	Suprafața (x1000 km²)	Populație (mil.)
2	Crișana	17,717	0,60
3	Muntenia	52,486	2,93
4	Basarabia	44,422	1,10
5	Maramureș	8,283	0,46
6	Oltenia	24,095	2,08
7	Dobrogea	23,300	0,97
8	Ardeal	57,807	6,79
9	Banat	18,393	1,23
10	Moldova	38,224	3,55
11	Bucovina	10,442	1,20

Figura 7.20. Conținutul fișierului Excel *Regiuni.xlsx*

Pentru a încărca datele dintr-un fișier Excel în Matlab se poate folosi funcția `readcell`.

Sintaxă: `fisier = readcell('NumeFisier.xlsx')`

Să se citească în Matlab datele din fișierul *Regiuni.xlsx*

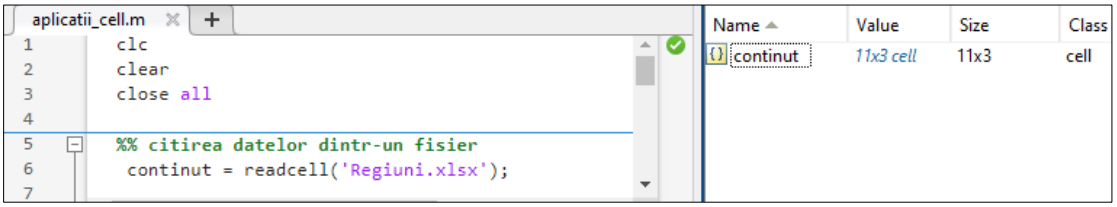


Figura 7.21. Citirea datelor din fișierul Excel

Dând dublu-click pe variabila `continut` din fereastra *Workspace* se va deschide un tabel cu conținutul citit din fișierul Excel.

11x3 cell

	1	2	3
1	'Provincia'	'Suprafața ...	'Populație (...
2	'Crișana'	17.7170	0.6000
3	'Muntenia'	52.4860	2.9300
4	'Basarabia'	44.4220	1.1000
5	'Maramureș'	8.2830	0.4600
6	'Oltenia'	24.0950	2.0800
7	'Dobrogea'	23.3000	0.9700
8	'Ardeal'	57.8070	6.7900
9	'Banat'	18.3930	1.2300
10	'Moldova'	38.2240	3.5500
11	'Bucovina'	10.4420	1.2000

Figura 7.22. Datele din Excel citite în Matlab

7.4.3. Scrierea datelor într-un fișier Excel

Pentru a adăuga date într-un fișier Excel se poate folosi funcția `writematrix`.

Sintaxă:

```
writematrix(valoare,'NumeFisier.xlsx','Sheet',NrSheet,…,
'Range', 'NrCelula')
```

unde:

- `valoare` = informația care se dorește să se scrie în fișierul Excel
- `'NumeFisier.xlsx'` = numele fișierului în care se dorește scrierea informațiilor
- `'Sheet'` = cuvânt cheie care anunță faptul că următorul parametru va fi numărul paginii din Excel în care se adaugă informația
- `NrSheet` = numărul paginii din Excel în care se adaugă informația
- `'Range'` = cuvânt cheie care anunță faptul că următorul parametru va fi celula din Excel în care se adaugă informația
- `'NrCelula'` = numărul celulei în care se va scrie informația

Să se introducă în fișierul Excel o nouă coloană care să conțină clasamentul provinciilor în funcție de suprafață. Această coloană să se numească *ClasamentSuprafata*.

aplicatii_cell.m

```
clc
clear
close all

%% citirea datelor dintr-un fisier
 continut = readcell('Regiuni.xlsx');

%% scriere date in Excel
% scriere cap de tabel pentru noua coloana
writematrix('Clasament Suprafata','Regiuni.xlsx','Sheet',1,'Range','D1');
% salvare coloana cu Provincii
Provincii = continut(2:end,1);
% salvare coloana cu Suprafata
Suprafete = cell2mat(continut(2:end,2));
% ordonarea suprafetelor descrescator
[val, index] = sort(Suprafete,'descend');
for i = 1:length(index)
    % identificare celula in care trebuie scris locul in clasament
    celula = ['D',num2str(index(i)+1)];
    % scriere in fisierul Excel in celula gasita
    writematrix(i,'Regiuni.xlsx','Sheet',1,'Range',celula);
end
```

Figura 7.23. Codul Matlab care realizează sortarea datelor dintr-un Excel

În urma scrierii datelor, conținutul fișierului Excel arată astfel:

	A	B	C	D
1	**Provincia**	**Suprafața (x1000 km²)**	**Populație (mil.)**	Clasament Suprafata
2	Crișana	17,717	0,60	8
3	Muntenia	52,486	2,93	2
4	Basarabia	44,422	1,10	3
5	Maramureș	8,283	0,46	10
6	Oltenia	24,095	2,08	5
7	Dobrogea	23,300	0,97	6
8	Ardeal	57,807	6,79	1
9	Banat	18,393	1,23	7
10	Moldova	38,224	3,55	4
11	Bucovina	10,442	1,20	9

Figura 7.24. Fișierul Excel după adăugarea din Matlab a clasamentului

Să se afișeze provinciile din fișierul Excel ordonate crescător în funcție de suprafață cu funcția `pie` (reprezentare grafică procentuală).

aplicatii_cell_pie.m

```
clc
clear
close all

%% citirea datelor dintr-un fisier
continut = readcell('Regiuni.xlsx');
Provincii = continut(2:end,1);
Suprafete = cell2mat(continut(2:end,2));

%% Reprezentarre cu pie
[SuprafeteSortate, index] = sort(Suprafete,'descend');
figure()
pie(SuprafeteSortate)
legend(Provincii(index),'Location','eastoutside')
```

Figura 7.25. Codul Matlab care realizează sortarea datelor dintr-un Excel și afișarea cu `pie`

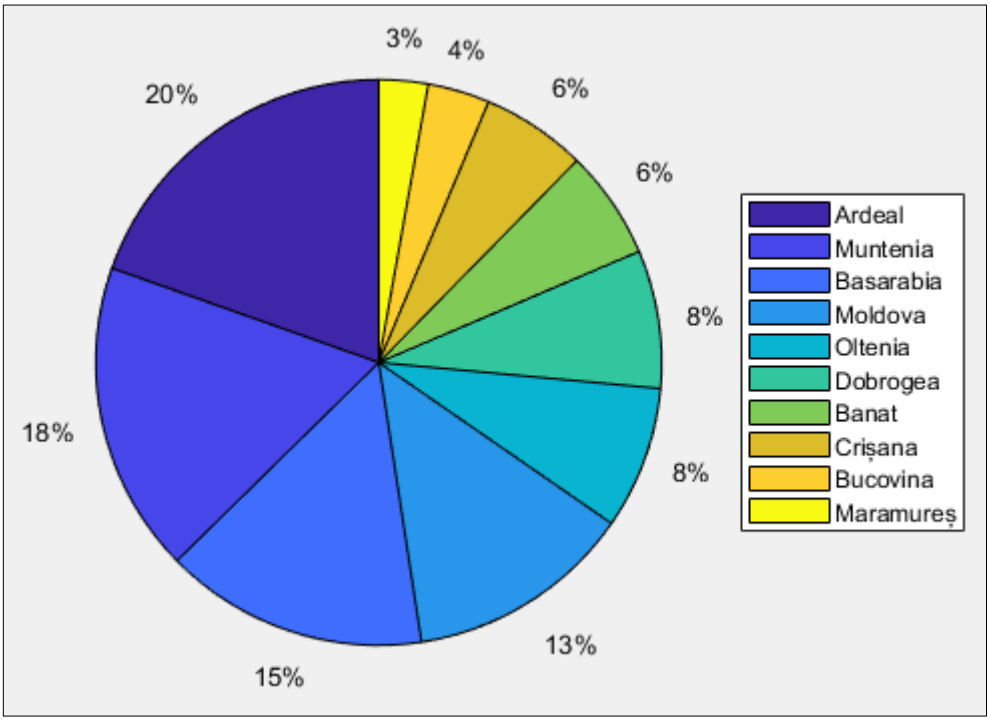


Figura 7.26. Reprezentare grafică cu funcția `pie`

7.5. Lucrul cu mai multe fișiere

În practică este nevoie de multe ori să se lucreze cu baze de date care conțin mai multe fișiere, iar în acest caz este util să le putem încărca automat pe toate. Înainte de a începe însă prezentarea modului în care se pot citi pe rând toate fișierele dintr-un folder, se va prezenta mai întâi tipul de date `struct`, deoarece fișierele dintr-un folder atunci când sunt citite în Matlab sunt organizate într-o structură.

7.5.1. Tipul de date `struct`

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

Sintaxă: `numeStruct = struct('param1',VAL1,'param2',VAL2,..)`

Să se creeze o structură care să conțină soluțiile `x` și `y` ale unei ecuații și valorile acestora.

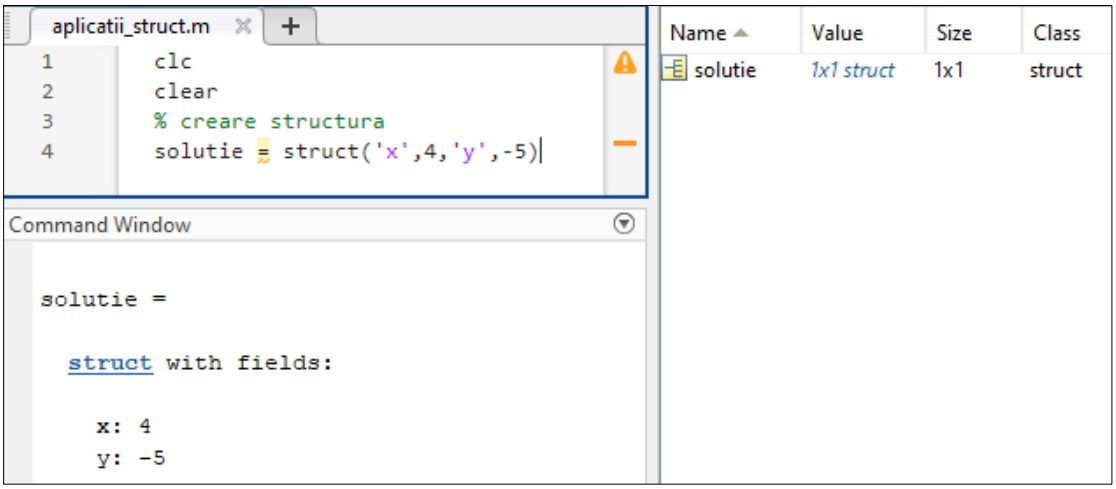


Figura 7.27. Generarea variabilelor de tip `struct`

Conținutul variabilei `solutie` va fi:

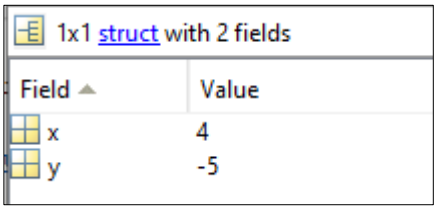


Figura 7.28. Variabilă de tip `struct`

Pentru a accesa un element al structurii se folosește sintaxa:

Sintaxă: `val = numeStruct.param`

Pentru structura creată anterior care conține soluțiile `x` și `y` ale unei ecuații și valorile acestora, să se salveze valoarea lui `x` în variabila `val_x` și valoarea lui `y` în variabila `val_y`.

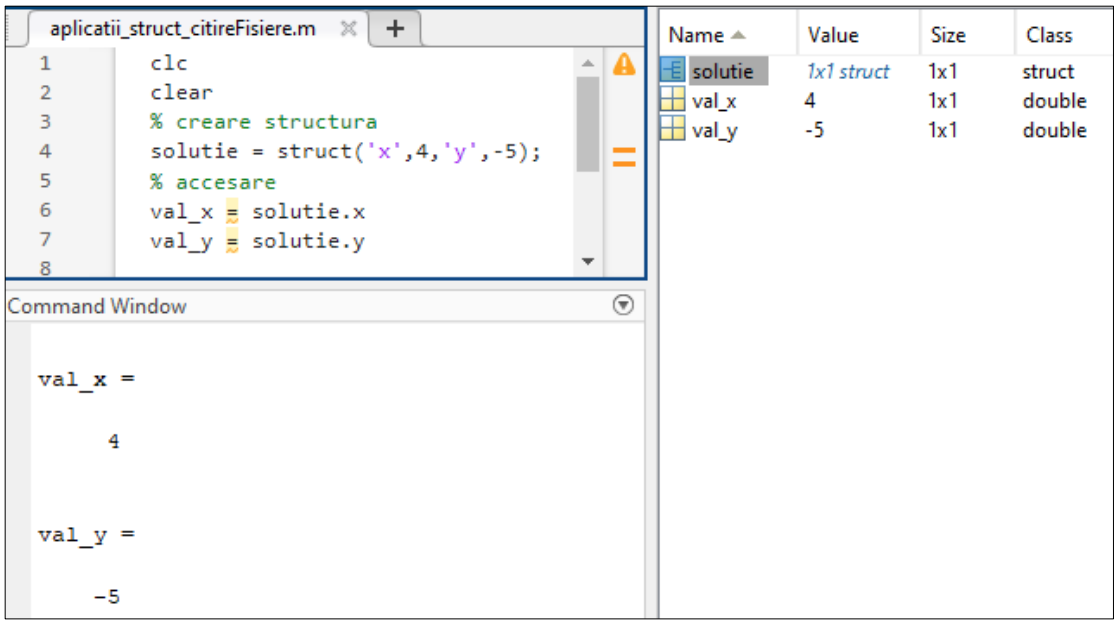


Figura 7.29. Lucrul cu variabile de tip `struct`

7.5.2. Citirea tuturor fișierelor dintr-un folder

Funcția `dir` permite acces la toate fișierele din folderul specificat.

Sintaxă: `continutFolder = dir(cale)`

Tabel 7.3. Câmpurile structurii returnate de funcția `dir`

Field Name	Description	Class
`name`	File or folder name	`char`
`folder`	Location of file or folder	`char`
`date`	Modification date timestamp	`char`
`bytes`	Size of the file in bytes	`double`
`isdir`	1 if name is a folder; 0 if name is a file	`logical`
`datenum`	Modification date as serial date number.	`double`

Să se citească toate imaginile dintr-un folder și să se salveze într-un alt folder redenumite. Indiferent de denumirile originale, noile denumiri ale imaginilor vor fi *imag1.jpg*, *imag2.jpg* etc.

aplicatii_struct_citireFisiere.m

```
clc
clear

% citirea tuturor imaginilor dintr-un folder,
% redenumirea lor si salvarea intr-un alt folder
cale_folderOrig = 'folder_old\';
cale_folderNou = 'folder_new\';
fisiere = dir(cale_folderOrig);
for i = 3:size(fisiere,1)
    numeFisier = fisiere(i).name; % nume imagine
    numeCuCale = [cale_folderOrig,numeFisier];% nume imagine + cale
    imag = imread(numeCuCale); % citire imagine din folder original
    numeNou = ['imag',num2str(i-2),'.jpg']; % redenumire
    numeNouCuCale = [cale_folderNou,numeNou]; % nume nou imagine + cale
    imwrite(imag,numeNouCuCale) % scriere imagine in folder nou
end
```

Figura 7.30. Codul Matlab care realizează citirea tuturor fișierelor dintr-un folder

Observații:

- Variabila `fisiere` este o structură care arată astfel:

10x1 struct with 6 fields

Fields	name	folder	date	bytes	i	datenum
1	'.'	'C:\D\Carti Catali...	'06-feb.-202...	0	1	7.3929e+05
2	'..'	'C:\D\Carti Catali...	'06-feb.-202...	0	1	7.3929e+05
3	'Nazca.jpg'	'C:\D\Carti Catali...	'01-dec.-20...	143366	0	7.3630e+05
4	'Peru.JPG'	'C:\D\Carti Catali...	'01-dec.-20...	85598	0	7.3630e+05
5	'bulz.jpg'	'C:\D\Carti Catali...	'01-dec.-20...	187463	0	7.3630e+05
6	'catel.JPG'	'C:\D\Carti Catali...	'01-nov.-20...	4458282	0	7.3627e+05
7	'corabie.jpg'	'C:\D\Carti Catali...	'01-dec.-20...	84533	0	7.3630e+05
8	'flamingo.j...	'C:\D\Carti Catali...	'01-dec.-20...	69108	0	7.3630e+05
9	'luna.jpg'	'C:\D\Carti Catali...	'02-dec.-20...	20562	0	7.3630e+05
10	'pietre.JPG'	'C:\D\Carti Catali...	'01-dec.-20...	98706	0	7.3630e+05

Figura 7.31. Câmpurile structurii `fisiere`

- Numele fișierelor (imaginilor) sunt în câmpul `name`, iar acestea încep de pe poziția a 3-a, din acest motiv ciclul `for` din codul Matlab începe de la 3.

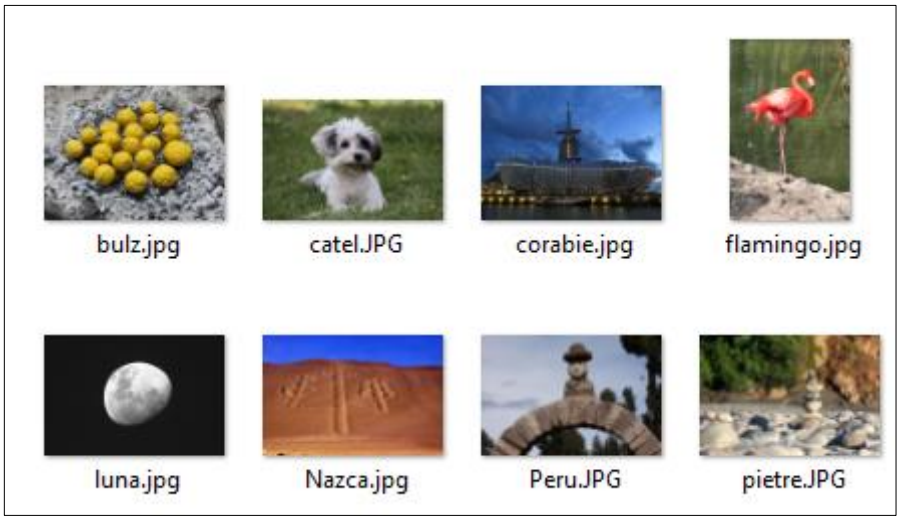


Figure 7.32. Conținut folder *folder_old*

Figure 7.33. Conținut folder *folder_new*

7.5.3. Selectarea folderului dorit dintr-o fereastră de tip *dialog box*

Pentru a permite utilizatorului să-și aleagă folderul dorit dintr-o fereastră (fără a fi nevoie să fie specificată calea în codul Matlab) se poate folosi funcția `uigetdir`.

Sintaxă: `numeFolder = uigetdir('calestart','titlu fereastra')`

Observație: funcția `uigetdir` poate să nu aibă niciun parametru de intrare, atunci când nu se dorește un anumit folder de start.

Să se scrie un program care să permită utilizatorului să-și aleagă un folder.

aplicatie_uigetdir.m

```
1   clc
2   clear
3
4   folder = uigetdir('Indrumar Matlab','Alege folderul dorit');
```

Figura 7.34. Folosirea funcției `uigetdir`

La rularea codului de mai sus, va apărea fereastra de mai jos, din care utilizatorul își poate alege folderul dorit.

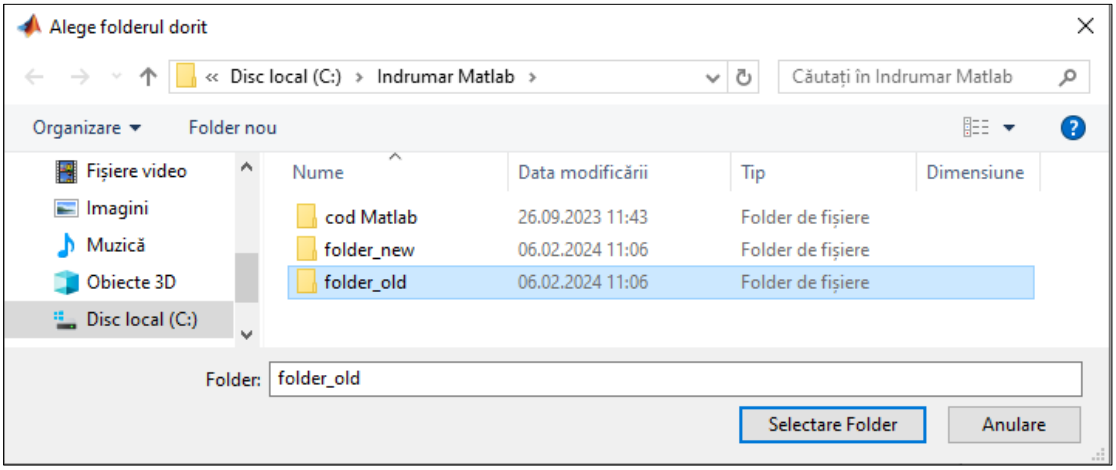


Figura 7.35. Selectarea unui folder dintr-o fereastră de *dialog box*, folosind funcția `uigetdir`

7.5.4. Selectarea fișierului dorit dintr-o fereastră de tip *dialog box*

Pentru a permite utilizatorului să-și aleagă un fișier dintr-o fereastră (fără a fi nevoie să fie specificată calea în codul Matlab) se poate folosi funcția `uigetfile`.

Sintaxă: `[filename, pathname] = uigetfile(restricții, 'titlu')`
unde:

- `Filename` = numele fișierului selectat
- `pathname` = calea la care se află fișierul selectat
- `restricții` = restricții legate de tipul de fișier (extensie) care să poată fi selectat. Dacă este un singur tip de fișier care să poată fi selectat, de exemplu doar imagini cu extensia *jpg*, atunci se va scrie `'.*jpg '`. Dacă sunt mai multe

tipuri de fișiere care să poată fi selectate, acestea se vor pune între acolade. De exemplu, dacă se dorește să poată fi selectate și imagini cu extensia *jpg* dar și cu extensia *bmp*, atunci acestea se vor scrie astfel: `{'*.jpg; *.bmp'}`

Observație: dacă se dorește vizualizarea tuturor fișierelor, indiferent de extensie, nu se va mai pune nici restricție

- `'titlu'` = titlul ferestrei de *dialog box*

Să se scrie un program care să permită afișarea oricărei imagini selectate dintr-o fereastră de *dialog box*. Extensiile permise să fie *jpg* și *bmp*. Titlul imaginii să fie numele fișierului.

aplicatie_uigetfile.m *

```
1   clc
2   clear
3   close all
4
5   [numeFisier, cale] = uigetfile({'*.jpg;*.bmp'},'Alege o imagine');
6
```

Figura 7.36. Folosirea funcției `uigetfile`

La rularea codului de mai sus se va deschide o fereastră din care poate fi selectată o imagine cu extensia dorită (*jpg* sau *bmp*).

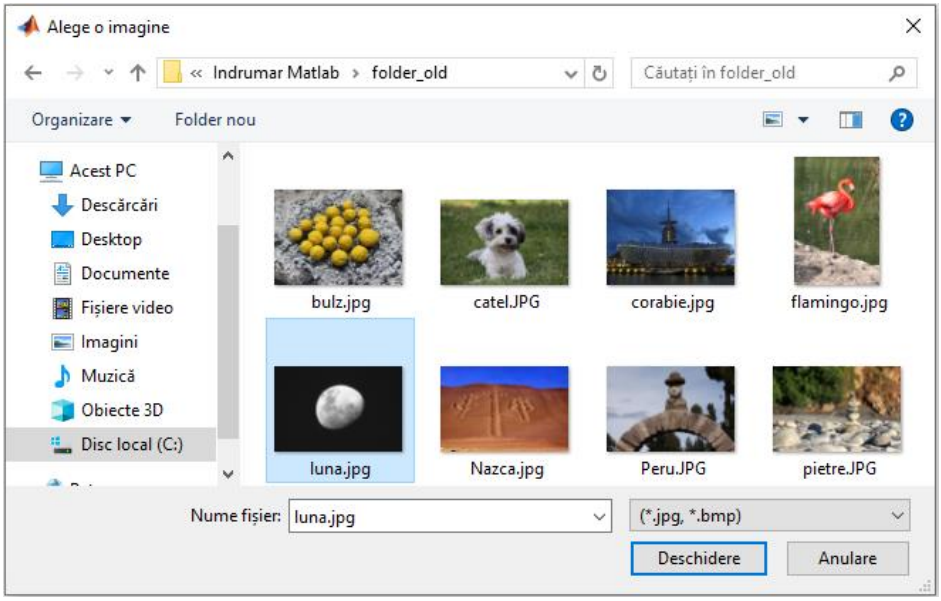


Figura 7.37. Selectarea unui fișier dintr-o fereastră de *dialog box*, folosind funcția `uigetfile`

aplicatie_uigetfile.m

```
clc
clear
close all

[numeFisier, cale] = uigetfile({'*.jpg;*.bmp'},'Alege o imagine');
numeCuCale = [cale, numeFisier];
imag = imread(numeCuCale);
figure()
imshow(imag)
title(numeFisier)
```

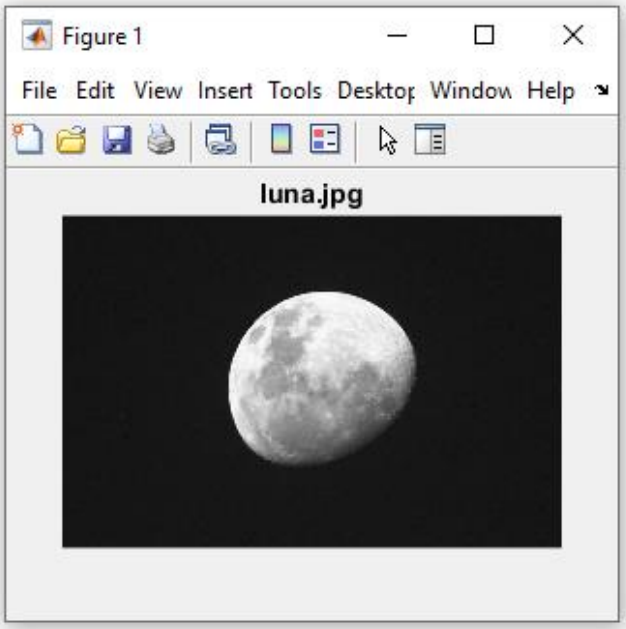


Figura 7.38. Program care să permită afișarea oricărei imagini selectate dintr-o fereastră de *dialog box*

7.6. Salvarea și încărcarea fișierelor de tip **.mat*

În urma rulării programului Matlab, de multe ori este util să salvăm anumite variabile în care sunt salvate rezultatele de interes. Acest lucru este indispensabil de exemplu atunci când se antrenează o rețea neurală (antrenare care poate dura și câteva zile). Când vom testa rețeaua nu va trebui să o și antrenăm de fiecare dată, ne vom folosi de rețeaua antrenată și salvată.

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

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

Sintaxa de mai sus va salva în fișierul `'nume_fisier'`, care va avea extensia *mat*, variabilele `'var_1', 'var_2', … ,'var_n'`.

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

Să se genereze 2 numere random `A` și `B` și să se salveze.

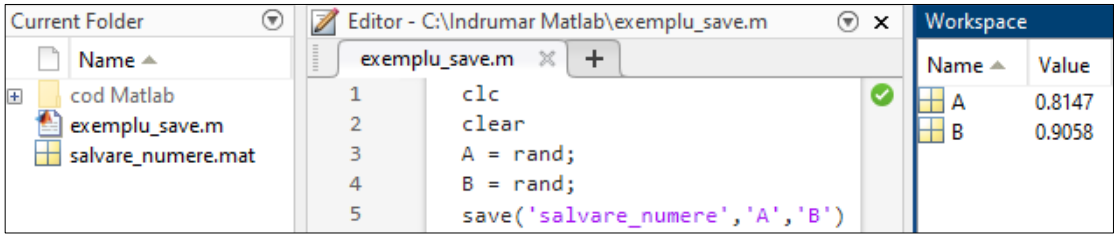


Figura 7.39. Exemplu de salvare date folosind funcția `save`

După rularea codului de mai sus, se poate observa că în fereastra *Current Folder* a apărut fișierul *salvare_numere.mat*

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

Sintaxă: `load('nume_fisier')`

Folosind sintaxa de mai sus se vor încărca toate variabilele salvate în fișierul `'nume_fisier'`

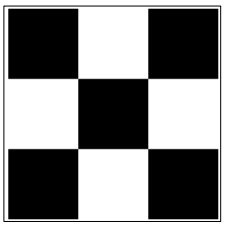
Observație: Dacă se dorește încărcarea doar a anumitor variabile se folosește sintaxa: `load('nume_fisier','var_1', 'var_2')` iar în acest caz se vor încărca doar variabile `var_1` și `var_2`.

7.7. Aplicații

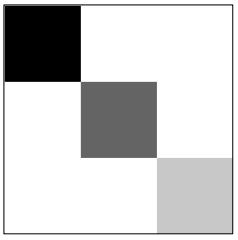
Aplicația 1. Pentru o imagine color:

a) Să se citească și să se afișeze imaginea în Matlab.
b) Pentru pixelul de la linia 2 și coloana 3, să se salveze în variabila `valR` componenta de roșu, în `valG` componenta de verde și în `valB` componenta de albastru.
c) Să se afișeze cele 3 straturi de culoare.
d) Să se realizeze conversia în grayscale și să se afișeze imaginea rezultată.
e) Să se realizeze complementul imaginii grayscale și să se afișeze imaginea rezultată.
f) Să se salveze în variabila `imagCrop`, regiunea din centrul imaginii, fără marginea de 30 de pixeli pe fiecare latură.
g) Să se salveze imaginea `imagCrop`, cu numele *imagTaiata.jpg*.
h) Toți pixelii negri din imagine să devină albi.

Aplicația 2. Să se genereze o imagine cu 300 de linii și 300 de coloane ca mai jos.



Aplicația 3. Să se genereze o imagine (având tipul de date `uint8`) cu 300 de linii și 300 de coloane conținând doar nivelurile de gri 0, 100, 200 și 255 ca mai jos.



Aplicația 4. Pentru un semnal audio:

a) Să se afle frecvența cu care a fost eșantionat semnalul.
b) Să se determine durata semnalului.
c) Să se reprezinte grafic semnalul audio în domeniul timp.
d) Să se salveze în variabila `semnalScurt` doar prima secundă a semnalului audio și să se reprezinte grafic.
e) Să se marcheze cu verde pe semnalul original prima secundă a semnalului.
f) Să se marcheze cu roșu porțiunea de semnal cuprinsă între cea mai mică și cea mai mare valoare a semnalului audio.
g) Să se insereze două secunde de liniște la mijlocul semnalului audio.

Aplicația 5. 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.

Aplicația 6. Fie un folder care conține doar imagini. Cerințe:

- Toate imaginile din folder să fie micșorate la jumătate și să fie mutate într-un alt folder, păstrându-și denumirile.
- Într-un fișier text cu denumirea *infoPoze.txt* să se scrie pentru fiecare imagine nou obținută, pe câte o linie, următoarele informații: numele imaginii, numărul de linii, numărul de coloane ale imaginii.
- Într-un fișier *Excel* cu denumirea *infoPoze.xlsx* să se scrie pentru fiecare imagine nou obținută, pe câte o linie, următoarele informații: numele imaginii, numărul de linii, numărul de coloane ale imaginii.

Aplicația 7. Calculul distanței dintre 2 puncte de pe harta.

Pas 1. Citire imagine și afișare imagine.

Pas 2. Selecția a 2 puncte de interes, (funcția `ginput`).

Pas 1

Pas 2

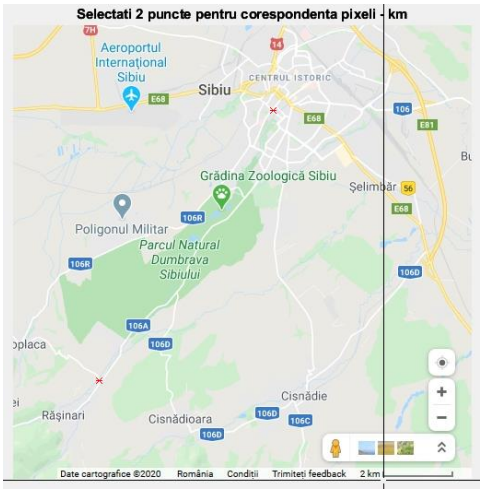
Pas 3. Afișare puncte de interes, (funcția `insertMarker`).

Pas 4. Calcul **distanță în pixeli** între cele 2 puncte de interes.

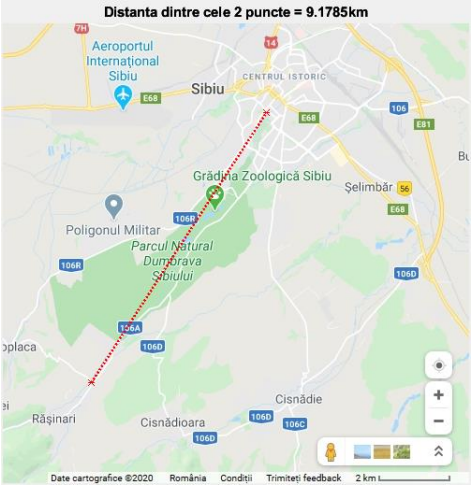
Pas 5. Selecția a 2 puncte pentru a determina corespondența dintre nr. pixeli și nr. kilometri (scara hărții din colțul dreapta jos al imaginii).

Pas 6. Calcul **distanță în km** între cele 2 puncte de interes.

Pas 7. Afișarea hărții cu traseul marcat. În titlul imaginii să apară distanța în km dintre cele 2 puncte selectate.



Pas 5



Pas 7

Aplicația 8. Transformarea datelor dintr-un fișier Excel în imagini.

Pentru această aplicație se va lucra cu baza de date MNIST (en . *Modified National Institute of Standards and Technology*) ce conține cifre scrise de mână și este folosită în mod obișnuit pentru instruirea și testarea diferitelor sisteme de procesare a imaginilor. Baza de date MNIST conține 2 fișiere Excel, unul cu 60.000 de imagini și celălalt cu 10.000 de imagini. Dimensiunea unei imagini este de 28 x 28 pixeli. Baza de date MNIST oficială este disponibilă online, accesul fiind gratuit.

Datele din fișiere sunt organizate astfel:

- prima coloană din fișierul Excel conține cifra reprezentată în imagine;
- următoarele 784 de coloane din fișierul Excel conțin valorile intensităților pixelilor din imagine, în format `uint8` (între 0 și 255).

Deoarece volumul de date este foarte mare, pentru această aplicație ne vom rezuma doar la primele 20 de linii din fișierul Excel. Conținutul acestuia va arăta astfel:

	A	IF	IG	IH	II	IJ	IK	IL	IM	IN	IO	IP	IQ	IR
1	cifra5	0	0	0	0	0	0	0	23	219	253	212	0	0
2	cifra0	253	252	230	223	145	225	249	252	252	128	0	0	0
3	cifra4	0	207	177	0	0	0	0	0	0	0	0	0	0
4	cifra1	0	0	0	0	0	0	0	48	64	64	24	0	0
5	cifra9	252	252	235	0	0	0	0	0	0	0	0	0	0
6	cifra2	0	5	78	252	241	88	205	252	121	0	0	0	0
7	cifra1	0	0	0	0	0	0	0	0	0	0	0	0	0
8	cifra3	45	45	0	0	0	9	29	147	252	252	235	0	0
9	cifra1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	cifra4	96	252	247	98	0	0	0	0	0	0	0	0	0
11	cifra3	0	0	0	0	0	90	215	254	105	7	0	0	0
12	cifra5	0	0	55	235	108	254	165	0	0	0	0	0	0
13	cifra3	0	0	0	0	0	25	209	254	254	166	7	0	0
14	cifra6	109	252	252	253	252	252	242	75	0	0	0	0	0
15	cifra1	0	0	0	0	0	0	0	0	0	0	0	0	0
16	cifra7	190	0	0	0	0	0	0	0	0	0	43	137	67
17	cifra2	0	0	9	214	247	49	4	158	247	0	0	0	0
18	cifra8	0	0	0	26	93	239	254	253	253	253	118	0	0
19	cifra6	0	0	0	0	0	0	0	0	0	0	0	0	0
20	cifra6	0	21	156	238	254	253	238	94	0	0	0	0	0

Deoarece o imagine are 28 x 28 pixeli, vor rezulta 784 + 1 coloane în fișierul Excel.

Cerințe de implement în Matlab

Pas 1. Să se afișeze prima cifră. Titlul imaginii să fie informația de pe prima coloană din fișierul Excel.

Pas 2. Să se afișeze toate imaginile stocate în fișierul Excel.

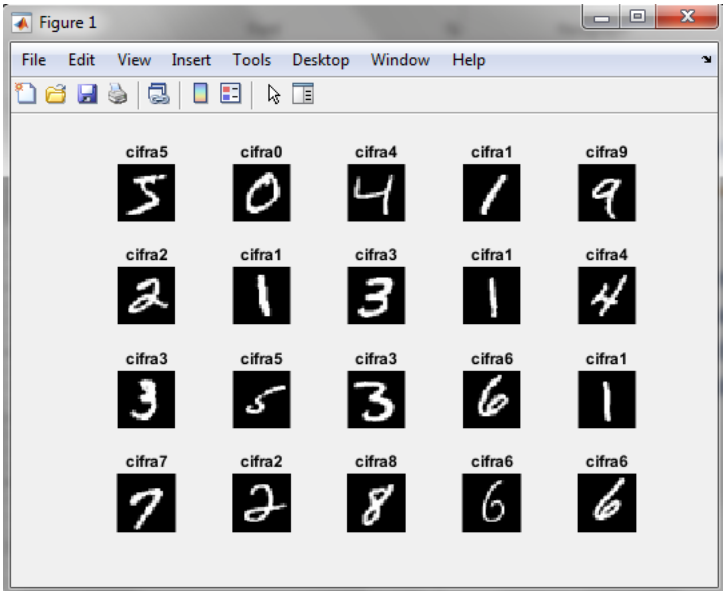


Figura 7.40. Generarea imaginilor pe baza informațiilor stocate în fișierul Excel

Pas 3. Să se salveze într-un folder toate imaginile obținute.

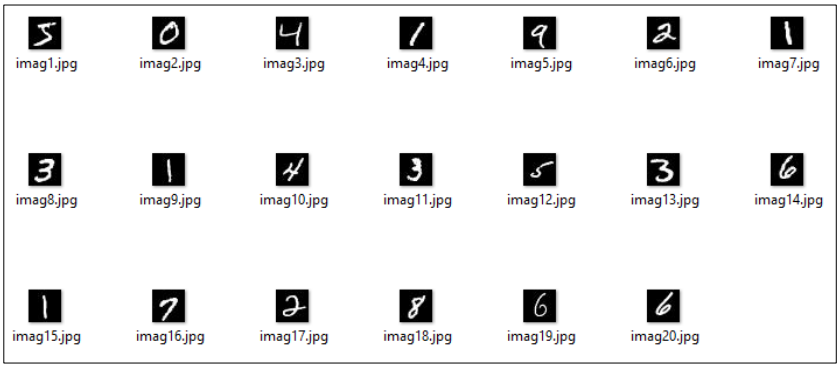


Figura 7.41. Salvarea celor 20 de imagini generate

Pas 4. Să se calculeze media tuturor imaginilor care conțin cifra 3 și să se afișeze.

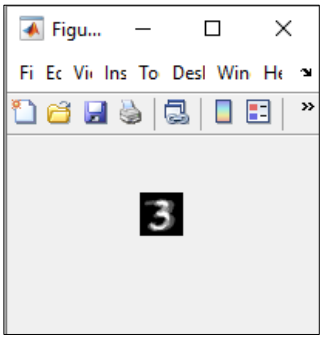


Figura 7.42. Media tuturor imaginilor care conțin cifra 3

Capitolul 8
Funcții în Matlab

Funcțiile sunt programe care rezolvă anumite cerințe (de exemplu 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.

8.1. Scrierea funcțiilor

Sintaxă scriere funcție

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

- `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:

- dacă o funcție nu întoarce niciun parametru, atunci sintaxa va fi*:*

```
function NumeFunctie(in1, in2, ...)
    instrucțiuni
end
```

- dacă o funcție nu primește niciun parametru de intrare, atunci sintaxa va fi*:*

```
function [out1, out2, ...] = NumeFunctie()
        instrucțiuni
end
```

Observație: pentru a forța ieșirea dintr-o funcție se folosește funcția `return`.

Salvarea unei funcții

O funcție poate fi salvată în două moduri:

- ***mod 1 salvare funcție:*** într-un fișier de funcție care conține doar funcția. În acest caz, este **obligatoriu** ca numele fișierului în care se salvează funcția să coincidă cu numele funcției; dacă numele funcției este `NumeFunctie` atunci numele fișierului trebuie să fie *NumeFunctie.m.*

  Observații:

  - un astfel de fișier, va apărea în fereastra *Current Folder* având iconița 
  - prima linie care se execută din fișierul în care este salvată funcția conține antetul funcției (înainte de antet pot exista doar comentarii)
  - 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`
- ***mod 2 salvare funcție:*** într-un fișier script care conține comenzi și definiții de funcții. Funcțiile trebuie să fie **obligatoriu** la sfârșitul fișierului. Fișierele script nu pot avea același nume ca o funcție din fișier.

8.2. Apelarea funcțiilor

Pentru a putea utiliza o funcție, aceasta se apelează.

Sintaxă apelare funcție:

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

- `NumeFunctie` reprezintă numele funcției
- `out1, out2`, ... reprezintă parametrii de ieșire
- `in1, in2,` ... reprezintă parametrii de intrare

Să se implementeze o funcție în Matlab cu numele `solveEcGrad2_Fcn` care să rezolve ecuația de grad 2.

- **parametrii de intrare**: valorile `a, b, c`
- **parametrii de ieșire**: rădăcinile `x1` și `x2`

Următorul cod Matlab reprezintă conținutul funcției.

solveEcGrad2_Fcn.m

```
function [x1, x2] = solveEcGrad2_Fcn(a,b,c)
% Radacinile ecuatiei de grad 2
% Parametrii intrare: a, b, c
% Parametrii iesire: x1, x2

delta = b^2-4*a*c;
x1 = (-b-sqrt(delta))/(2*a);
x2 = (-b+sqrt(delta))/(2*a);

```

Următorul cod Matlab reprezintă programul principal din care se apelează funcția.

solveEcGrad2_Main.m

```
clc
clear
% initializare parametrii de intrare
coef1 = 1; coef2 = 3; coef3 = 2;
% apelare functie
[rez1, rez2] = solveEcGrad2_Fcn(coef1, coef2, coef3);
disp("x1 = " + rez1)
disp("x2 = " + rez1)
```

Command Window

```
x1 = -2
x2 = -2
```

Observație: numele variabilelor din antetul funcției nu este obligatoriu să coincidă cu numele variabilelor folosite la apelare. În cazul de față Matlab-ul va asocia: lui `a` valoarea lui `coef1`, lui `b` valoarea lui `coef2`, lui `c` valoarea lui `coef3`. În cazul parametrilor de ieșire, în `rez1` și `rez2` se returnează valorile lui `x1` și `x2`.

În final, în fereastra *Current Folder* vor fi cele 2 fișiere:

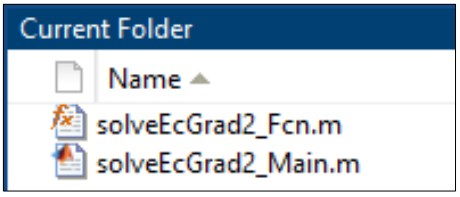


Figura 8.1. Fișierul funcție și fișierul script din care se apelează funcția

Să se implementeze în Matlab o funcție numită `sinusoida_Fcn` care să genereze o sinusoidă.

Parametrii de intrare: amplitudinea maximă (**`A`**), frecvența (**`F`**), frecvența de eșantionare (**`Fs`**), faza inițială (**`faza0`**), durata semnalului în secunde (**`durata`**)

Parametrii de ieșire sunt:

- Vectorul **`t`** al momentelor de timp
- Vectorul **`s`** conținând valorile sinusoidei

Următorul cod Matlab reprezintă conținutul funcției.

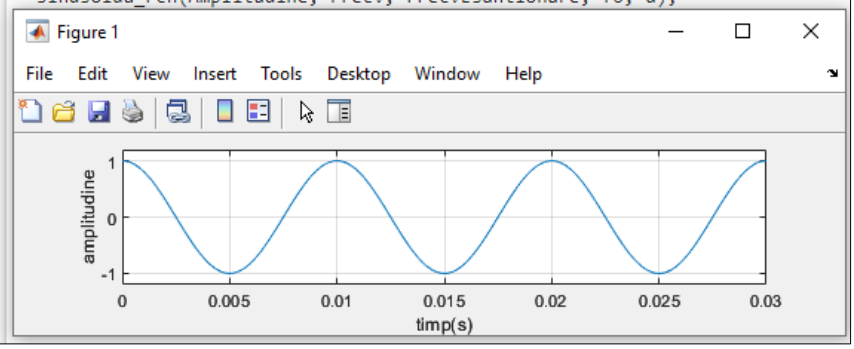
sinusoida_Fcn.m | sinusoida_main.m

```
function [t, s] = sinusoida_Fcn(A, F, Fs, faza0, durata)
% generare sinusoida
% Parametrii de intrare:
    % A = amplitudine, F = frecvență, Fs = frecvență de eșantionare
    % faza0 = fază inițială, durata = cât durează semnalul
% Parametrii de ieșire:
    % t = vectorul momentelor de timp
    % s = valorile sinusoidei
t = 0:1/Fs:durata;
s = A*sin(2*pi*F*t+faza0);
figure()
plot(t,s), ylim([min(s)-0.2, max(s)+0.2]), grid
xlabel('timp(s)'), ylabel('amplitudine')
```

Următorul cod Matlab reprezintă programul principal din care se apelează funcția.

sinusoida_Fcn.m | sinusoida_main.m

```
clc
clear
close all
% parametrii de intrare
Amplitudine = 1;
Frecv = 100;
FrecvEsantionare = 10000;
f0 = pi/2;
d =  0.03;
% apelare funcție
sinusoida_Fcn(Amplitudine, Frecv, FrecvEsantionare, f0, d);
```

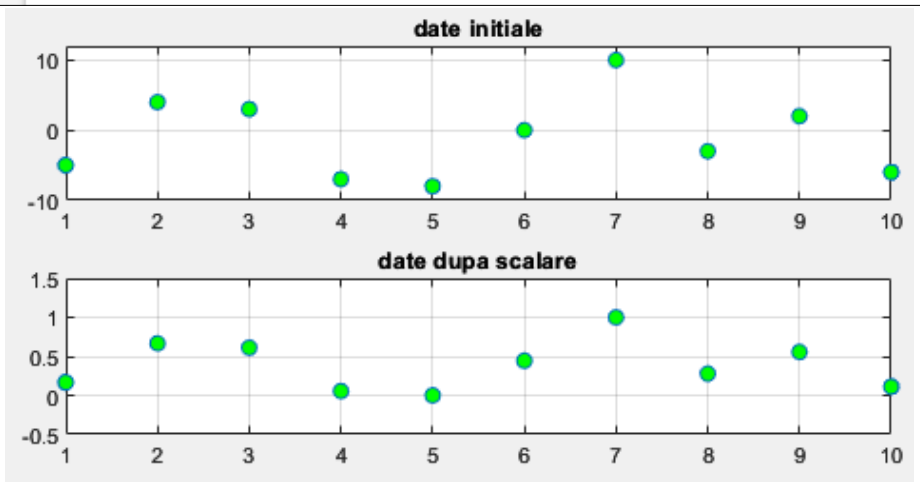


Să se implementeze o funcție în Matlab care să modifice liniar toate valorile unui vector astfel încât acestea să fie aduse într-un interval dorit `[a, b]`. Se vor reprezenta datele din vector înainte și după transformare. Funcția va fi scrisă în fișierul script din care va fi și apelată.

- **parametrii de intrare**: vectorul `X`, noul interval `[a, b]`
- **parametrii de ieșire**: vectorul `X` scalat

scalare_main.m

```
clc
clear
close all
% generare vector X cu valori pseudorandom
X = randi([-10, 10],1,10);
% a = noua valoare minimă
% b = noua valoare maximă
a = 0; b = 1;
% apelare functie scalare_Fcn
Xnew = scalare_Fcn(X, a, b);
figure()
subplot(2,1,1)
    plot(X,'o','MarkerFaceColor','g')
    ylim([min(X)-2, max(X) + 2])
    title('date initiale'), grid
subplot(2,1,2)
    plot(Xnew,'o','MarkerFaceColor','g')
    ylim([min(Xnew) - 0.5, max(Xnew) + 0.5])
    title('date dupa scalare'), grid

function Xnew = scalare_Fcn(X, a, b)
% Scalează valorile unui vector în intervalul [a, b]
valMax = max(X);
valMin = min(X);
Xnew = (b-a)/(valMax-valMin)*(X-valMin)+a;
end
```



Să se implementeze o funcție care să realizeze oglindirea unei imagini.

- **Parametrii de intrare:** imaginea originală și un șir de caractere `optiune` care poate avea valorile: `'FlipOrizontal'` sau `'FlipVertical'`
- **Parametru de ieșire:** imaginea modificată conform opțiunii alese

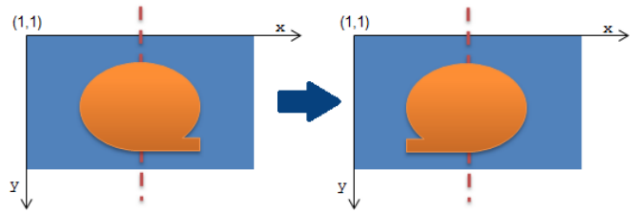


Figura 8.2. Fli*p orizontal* (Reflexie față de axa verticală)

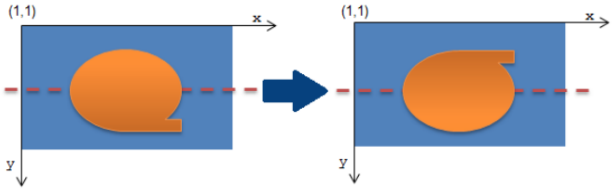


Figura 8.3. *Flip vertical* (Reflexie față de axa orizontală)

La rularea programului principal utilizatorul își poate selecta ce imagine dorește. Tot din programul principal se va realiza afișarea imaginii originale și a celei modificate.

Funcția trebuie să verifice că parametrul `optiune` are una dintre cele 2 valori impuse, în caz contrat se va afișa un mesaj de avertizare și nu se va mai reprezenta grafic nimic.

Următorul cod Matlab reprezintă conținutul funcției.

imagFlip_Fcn.m | imagFlip_Main.m

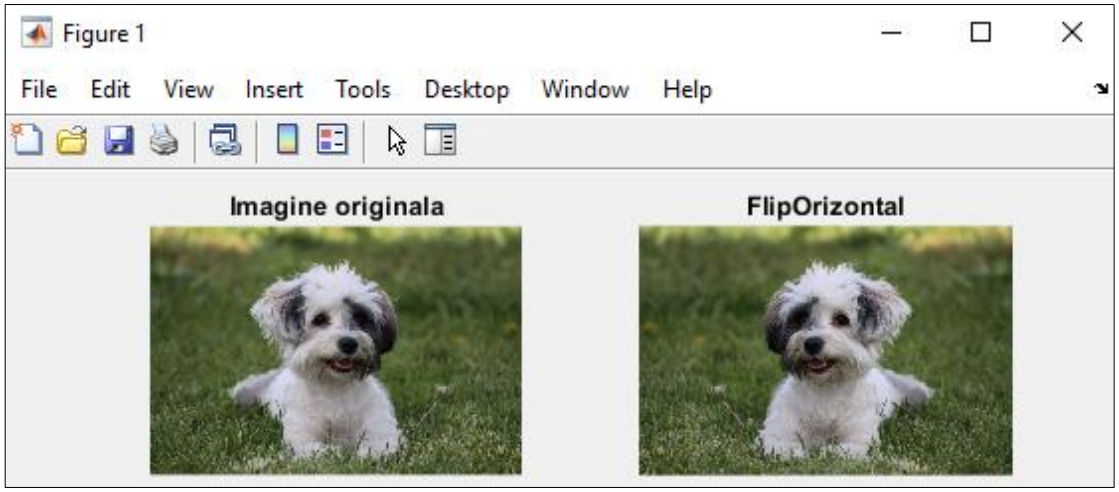
```
function imagModif = imagFlip_Fcn(imag, optiune)
% imag = imaginea originala
% optiune poate fi: FlipOrizontal sau FlipVertical
% imagModif = imaginea obtinuta conform variantei de Flip aleasa
[M, N, P] = size(imag);
switch(optiune)
    case 'FlipOrizontal'
        imagModif = imag(:,N:-1:1,:); % flip orizontal
    case 'FlipVertical'
        imagModif = imag(M:-1:1,:,:); % flip vertical
    otherwise
         disp('Parametrii acceptati sunt doar: FlipOrizontal,FlipVertical')
         imagModif = [];
end
```

Următorul cod Matlab reprezintă programul principal din care se apelează funcția.

imagFlip_Fcn.m | imagFlip_Main.m

```
clc
clear
close all
[nume, cale] = uigetfile('.jpg'); % selectare imagine
% tip de Flip dorit: FlipOrizontal sau FlipVertical
tipFlip = 'FlipOrizontal';
numeImag = [cale,nume];
imag = imread(numeImag);
imagModif = imagFlip_Fcn(imag, tipFlip); % apelare functie
% doar daca s-a realizat Flip se afiseaza imaginea originala si cea
% modificata conform optiunii de Flip aleasa
if(~isempty(imagModif))
figure()
subplot(1,2,1)
    imshow(imag), title('Imagine originala')
subplot(1,2,2)
    imshow(imagModif), title(tipFlip)
end
```

La rularea codului de mai sus se va afișa imaginea:



Dacă variabila `tipFlip` cu care se apelează funcția are o altă valoare decât cele 2 impuse (`'FlipOrizontal'` sau `'FlipVertical'`), atunci la rularea programului nu se va mai reprezenta grafic nimic și se va afișa mesajul următor:

```
Command Window
Parametrii acceptati sunt doar: FlipOrizontal,FlipVertical
>>
```

8.3. Aplicații

Aplicația 1. Să se implementeze o funcție `produsFcn` care să primească ca parametru de intrare un număr natural `N` și să întoarcă ca parametru de ieșire un număr `M`, unde $M = 1 \cdot 2 \cdot \ldots \cdot N$.

Funcția să verifice mai întâi că N este număr natural, în caz contrar să afișeze în fereastra *Command Window* mesajul "Atenție! N trebuie să fie număr natural".

Aplicația 2. Să se implementeze în Matlab o funcție numită `sumaFCN` care să primească la intrare numerele naturale `a` și `b`. Funcția să aibă ca parametru de ieșire o variabilă în care să se calculeze:

- suma tuturor numerelor naturale din intervalul `[a, b]`, dacă `a < b`
- suma tuturor numerelor naturale din intervalul `[b, a]`, dacă `b < a`

Funcția să verifice mai întâi că `a` și `b` sunt numere naturale, în caz contrar să afișeze în fereastra *Command Window* mesajul "Atenție! `a` și `b` trebuie să fie numere naturale".

Aplicația 3. Să se implementeze în Matlab o funcție numită `matriceConfuzie` care să primească la intrare o matrice pătratică și să întoarcă suma elementelor de pe diagonala principală împărțită la suma tuturor elementelor din matrice.

Funcția să verifice mai întâi că matricea primită ca parametru de intrare este pătratică în caz contrar să se deschidă o fereastră cu mesajul "Atenție! Matricea trebuie să fie pătratică!".

Aplicația 4. Să se implementeze o funcție `distEuclid` care primește ca parametri de intrare doi vectori și calculează distanța euclidiană dintre cei doi vectori.

Funcția să verifice mai întâi că cei doi vectori au același număr de elemente, în caz contrar să se afișeze mesajul "Atenție! Cei doi vectori trebuie să aibă același număr de elemente".

Aplicația 5. Să se implementeze funcția `dreptunghiularFcn` care să primească următorii parametri de intrare:

- în variabila `durata`, durata semnalului
- în variabila `valMax`, valoarea maximă semnalului
- în variabila val`Min`, valoarea minimă semnalului
- în variabila `faza0`, faza inițială
- în variabila `Fs`, frecvența de eșantionare
- în variabila `F`, frecvența de repetiție
- în variabila `factorUmplere`, factorul de umplere

Funcția nu returnează parametri de ieșire! Funcția să realizeze reprezentarea grafică a semnalului dreptunghiular cu parametrii dați.

Aplicația 6. Să se implementeze funcția `sinusoidaFCN` al cărei scop să fie marcarea unei oscilații dorite dintr-o sinusoidă. Parametrii de intrare ai funcției sunt:

- un semnal sinusoidal cu 4 oscilații
- frecvența de eșantionare
- numărul oscilației care se dorește a fi marcată

Funcția nu returnează parametri de ieșire! Funcția să realizeze reprezentarea grafică cu albastru a semnalului sinusoidal, marcând cu verde oscilația dată ca parametru de intrare.

Aplicația 7. Să se implementeze o funcție care să primească la intrare 2 parametri: un semnal audio și o opțiune care poate fi '*fadeIn*' sau '*fadeOut*'. În funcție de opțiunea aleasă, se realizează următoarele operații:

- *'fadeIn':* se modifică volumul semnalul audio astfel încât acesta să pornească cu volum zero iar la final să ajungă la volumul semnalului nemodificat
- *'fadeOut'*: se modifică volumul semnalului audio astfel încât acesta să pornească de la volumul inițial iar la final să ajungă la volum zero.

Funcția va întoarce semnalul modificat conform opțiunii alese.

Funcția trebuie să verifice mai întâi că opțiunea aleasă se regăsește printre cele 2 de mai sus, altfel va afișa un mesaj de avertizare în fereastra *Command Window*.

Capitolul 9
Calcul parametric (simbolic)

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, calculul derivatelor și integralelor.

9.1. Declararea variabilelor simbolice

Rezolvarea parametrică se poate realiza în Matlab folosind *variabile simbolice*. Acestea se declară folosind `syms`.

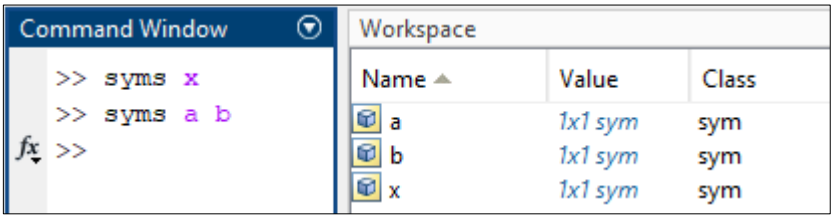


Figura 9.1. Exemple de definire a variabilelor simbolice

Atunci când se utilizează calculul parametric este recomandat a se folosi *Live Editor* pentru a vizualiza relațiile matematice într-o formă cât mai apropiată de scrierea matematică.

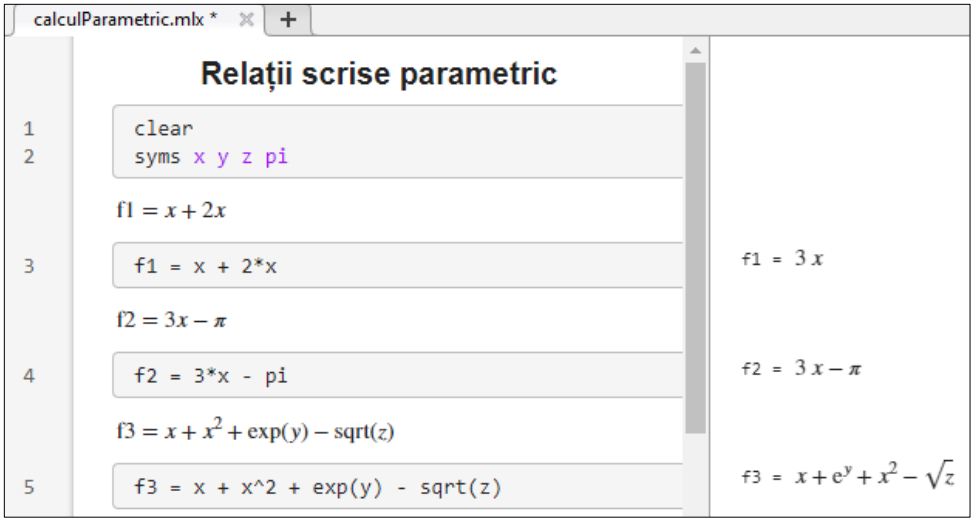


Figura 9.2. Exemple de relații matematice scrise parametric (simbolic)

9.2. Substituirea valorilor simbolice cu valori numerice

Pentru a înlocui o valoare simbolică cu o valoare se folosește funcția `subs`.

Sintaxă: `valExpr = subs(expresie, [simboluri], [valori])`

- `expresie` = expresia simbolică în care se face înlocuirea
- `simboluri` = variabilele simbolice ce vor fi substituite
- `valori` = valorile pe care le vor lua variabilele simbolice
- `valExpr` = valoarea expresiei în urma substituirii

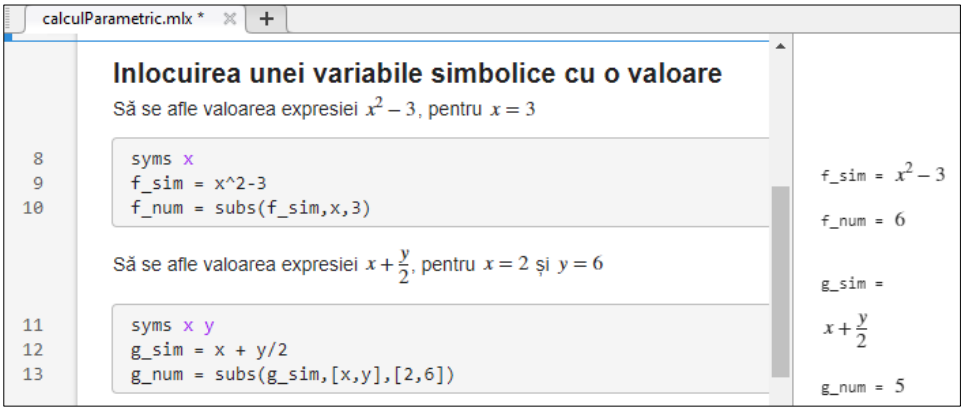


Figura 9.3. Exemple de substituire a valorilor simbolice cu valori numerice

9.3. Rezolvarea ecuațiilor

Pentru a găsi soluția/soluțiile unei ecuații se poate folosi funcția `solve`.

Sintaxă: `solutie = solve(expresie, var)`

- `expresie` = expresia simbolică (ecuația) ce se rezolvă
- `var` = variabila ce se consideră necunoscută
- `solutie` = soluțiile ecuației (vor fi salvate într-un vector coloană)

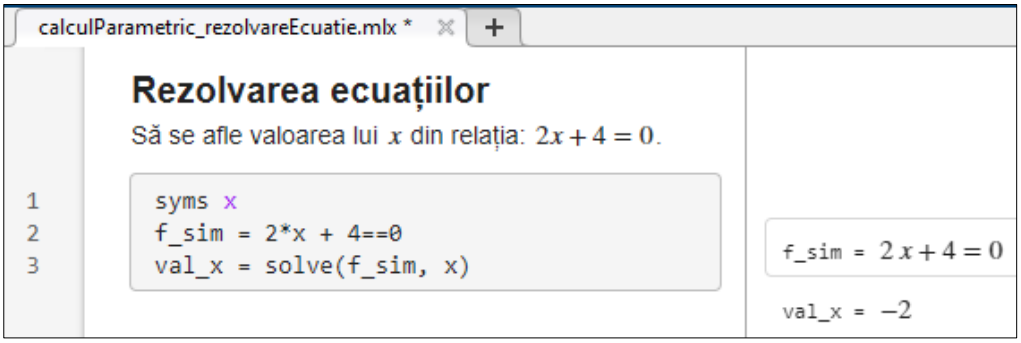
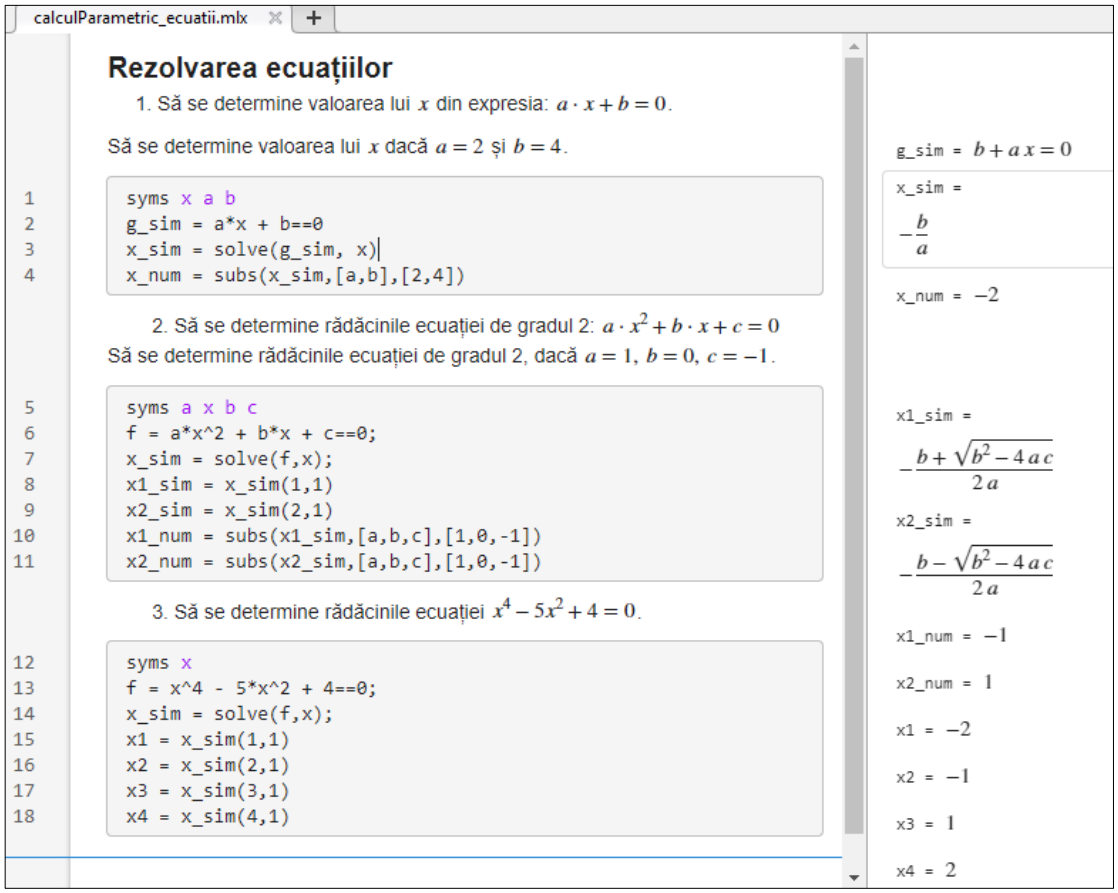


Figura 9.4. Exemplu de rezolvare a unei ecuații

Folosind calcul parametric:

1. Să se scrie ecuația: $a \cdot x + b = 0$. Pentru $a = 2$ și $b = 4$ să se determine valoarea lui x.
2. Să se scrie ecuația: $a \cdot x^2 + b \cdot x + c = 0$. Să se determine valoarea lui x pentru $a = 1, b = 0, c = -1$.
3. Să se determine rădăcinile ecuației: $x^4 - 5 \cdot x^2 + 4 = 0$.



9.4. Rezolvarea sistemelor de ecuații

Rezolvarea sistemelor de ecuații se poate realiza cu funcția `solve`.

Sintaxă: `solutie = solve([Ec1, Ec2,..],[var1, var2, ..])`

- `Ec1, Ec2, …` = ecuațiile parametrice din care este constituit sistemul
- `var1, var2, …` = variabilele ce trebuie determinate
- `solutie` = soluțiile sistemului de ecuații (vor fi salvate într-o structură)

Folosind calcul parametric să se rezolve sistemul:

$$\begin{cases} 3x + y &= 5 \\ 2x - 3y &= -4 \end{cases}$$

calculParametric_circuit.mlx

Rezolvare sisteme de ecuații

Să se rezolve sistemul de ecuații: $\begin{cases} 3x + y &= 5 \\ 2x - 3y &= -4 \end{cases}$

```
1  syms x y
2  Ec1 = 3*x + y ==5
3  Ec2 = 2*x - 3*y ==-4
4  solutie = solve([Ec1, Ec2],[x,y])
5  % In cazul sistemelor de ecuatii,
6  % functia solve va returna o structura
7  val_x = solutie.x
8  val_y = solutie.y
```

Ec1 = $3x + y = 5$

Ec2 = $2x - 3y = -4$

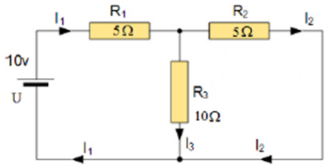
solutie = *struct with fields:*

x: 1
y: 2

val_x = 1

val_y = 2

Folosind calcul parametric să se calculeze curenții circuitului următor.

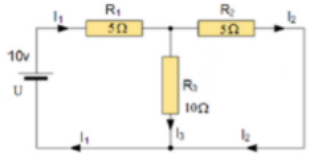


$$\begin{cases} I_1 = I_2 + I_3 \\ U = I_2 \cdot R_2 + I_1 \cdot R_1 \\ I_2 \cdot R_2 = I_3 \cdot R_3 \end{cases}$$

calculParametric_circuit.mlx

Rezolvarea sistemelor de ecuații

Pentru circuitul de mai jos, să se determine formulele de calcul pentru curenți și valorile numerice ale curenților.



$$\begin{cases} I_1 = I_2 + I_3 \\ U = I_2 \cdot R_2 + I_1 \cdot R_1 \\ I_2 \cdot R_2 = I_3 \cdot R_3 \end{cases}$$

```
1  syms I1 I2 I3 R1 R2 R3 U
2  Ec1 = I1==I2 + I3
3  Ec2 = U==I2*R2+I1*R1
4  Ec3 = I2*R2==I3*R3
5  [I1sim, I2sim, I3sim] = solve([Ec1, Ec2, Ec3],[I1, I2, I3])
6  I1_num = subs(I1sim,[R1, R2, R3, U],[5,5,10,10])
7  I2_num = subs(I2sim,[R1, R2, R3, U],[5,5,10,10])
8  I3_num = subs(I3sim,[R1, R2, R3, U],[5,5,10,10])
```

I1sim =

$$\frac{R_2 U + R_3 U}{R_1 R_2 + R_1 R_3 + R_2 R_3}$$

I2sim =

$$\frac{R_3 U}{R_1 R_2 + R_1 R_3 + R_2 R_3}$$

I3sim =

$$\frac{R_2 U}{R_1 R_2 + R_1 R_3 + R_2 R_3}$$

I1_num =

$$\frac{6}{5}$$

I2_num =

$$\frac{4}{5}$$

I3_num =

$$\frac{2}{5}$$

9.5. Derivarea expresiilor simbolice

Derivarea expresiilor simbolice se realizează cu funcția `diff`.

- **Derivarea expresiilor simbolice cu o variabilă**

Sintaxă: `rez = diff(f)`

unde `f` este o expresie simbolică. Dacă lipsește variabila după care se face derivarea, atunci se consideră implicit că este `x`.

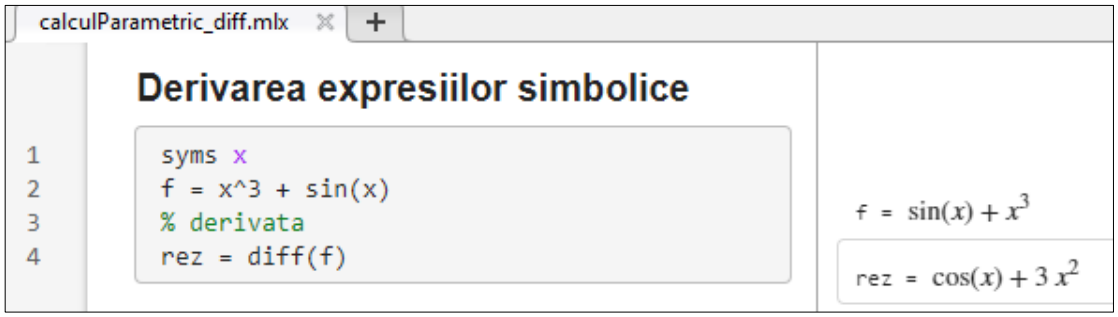


Figura 9.5. Exemplu de derivare a unei expresii simbolice cu o singură variabilă

- **Derivarea expresiilor simbolice cu mai multe variabile**

Presupunând că există o expresie simbolică `f` în funcție de două variabile `x` și `y` și se dorește realizarea derivatei în funcție de `x`, atunci sintaxa va fi:

Sintaxă: `rez = diff(f,x)`

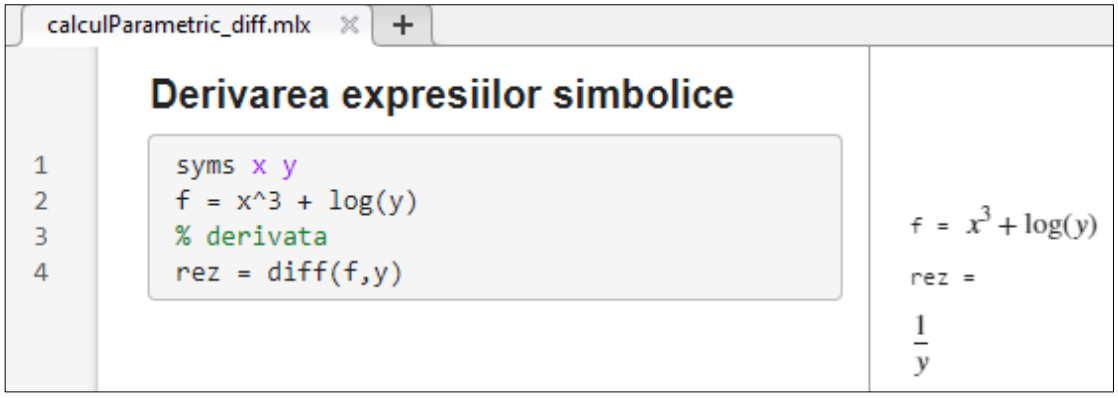


Figura 9.6. Exemplu de derivare a unei expresii simbolice cu 2 variabile

9.6. Integrarea expresiilor simbolice

Pentru a realiza integrarea unei expresii simbolice, se folosește funcția `int`.

Sintaxă: `rez = int(f)`

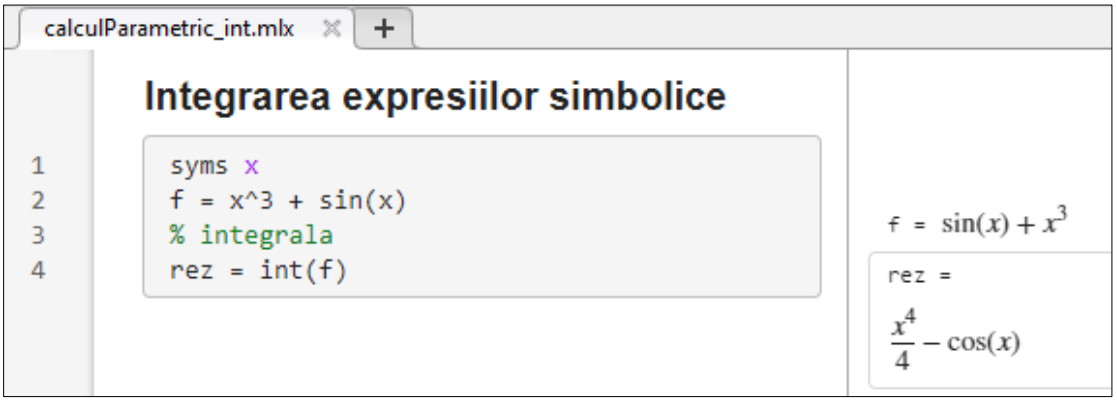


Figura 9.7. Exemplu de integrare a unei expresii simbolice

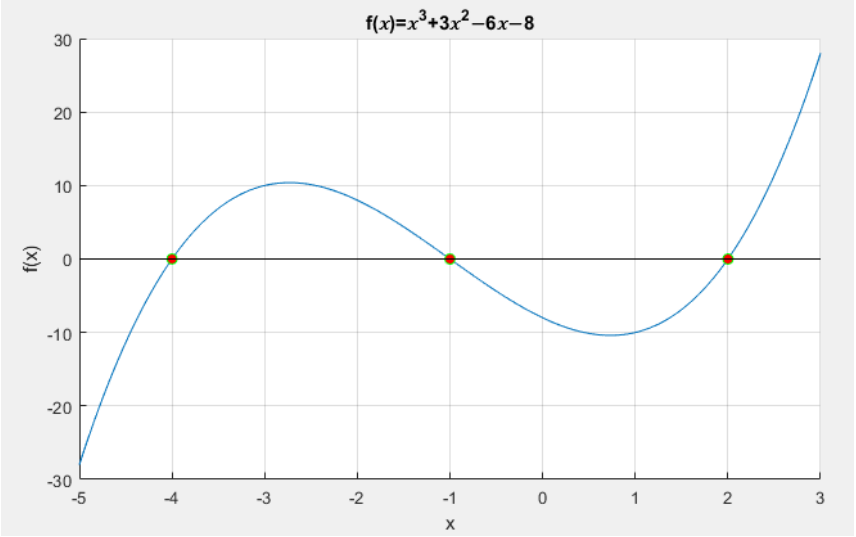
9.7. Aplicații

Aplicația 1. Fie funcția: $f(x) = x^3 + 3x^2 - 6x - 8$

a) Să se reprezinte graficul funcției pentru $x \in [-5,\ 3]$. Distanța dintre două valori consecutive ale lui x să fie 0.01.

b) Să se afle rădăcinile ecuației $f(x) = 0$ și să se marcheze pe graficul funcției.

Figura finală trebuie să arate astfel:



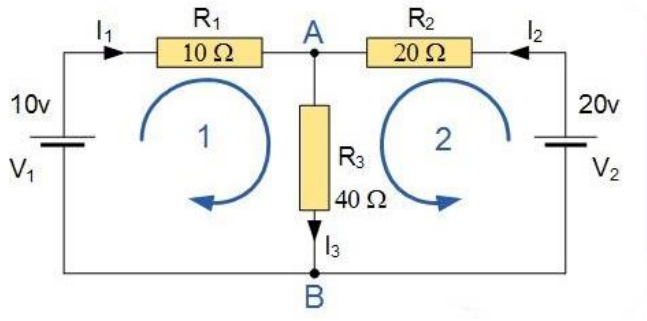
Aplicația 2. Să se rezolve următorul sistem de ecuații:

$$\begin{cases} x + 3y + z = 10 \\ 3x + 2y + 3z = 16 \\ 2x + 5y - 2z = 6 \end{cases}$$

Să se salveze:

- în variabila `val_x`, valoarea lui `x`
- în variabila `val_y`, valoarea lui `y`
- în variabila `val_z`, valoarea lui `z`

Aplicația 3. Fie următorul circuit electric:



a) Să se scrie în Matlab sistemul de ecuații care să ducă la aflarea curenților din circuitul electric de mai sus.
b) Să se determine formulele simbolice de calcul ale curenților.
c) Să se determine valorile numerice ale curenților.

Aplicația 4. Să se calculeze derivatele următoarelor funcții:

a) $f(x) = 3x^2 - 8x + 2$
b) $f(x) = x^2 + 2^x$
c) $f(x) = \frac{x-4}{x^2+2}$

Aplicația 5. Să se calculeze integralele următoarelor funcții:

a) $f(x) = x^2 + 2x + \frac{1}{x}$
b) $f(x) = 2e^x - 3^x$
c) $f(x) = \frac{1-\sqrt{1-x^2}}{1-x^2}$

Capitolul 10
Reprezentări grafice 3D

În acest capitol se dorește reprezentarea grafică a unei funcții matematice de forma:

$$z = f(x, y)$$

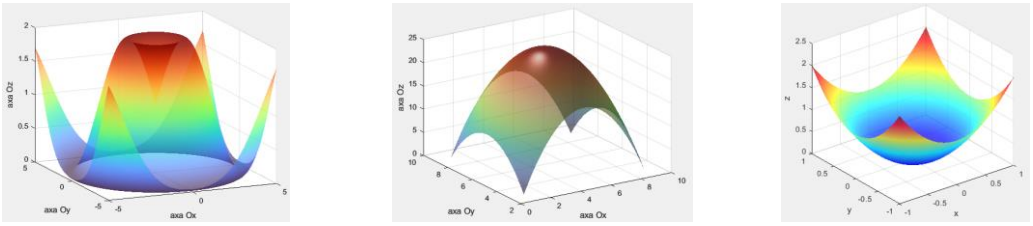


Figura 10.1. Exemple de reprezentări grafice 3D

10.1 Abordare matematică

Înainte de a detalia modul de reprezentare grafică a acestui tip de funcție în Matlab, vom aborda mai întâi subiectul din punct de vedere matematic, pornind de la o funcție simplă, ca de exemplu funcția:

$$f(x, y) = 25 - (x - 5)^2 - (y - 6)^2 \text{ cu } x \in [1,\ 9] \text{ și } y \in [3,\ 9]$$

Pentru $x \in [1,\ 9]$ și $y \in [3,\ 9]$ rezultă perechile de puncte în spațiul *x*O*y*:

- pentru $x = 1$ și $y = 3$ ➔ {1, 3}
- pentru $x = 1$ și $y = 4$ ➔ {1, 4}
- pentru $x = 1$ și $y = 5$ ➔ {1, 5}

...

- pentru $x = 1$ și $y = 9$ ➔ {1,9}

...

- pentru $x = 5$ și $y = 6$ ➔ {5,6}

...

- pentru $x = 9$ și $y = 9$ ➔ {9,9}

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}.

Prin urmare:

- se va construi matricea `X` care va avea pe fiecare linie valorile lui x, iar numărul de linii va fi egal cu numărul de elemente posibile pentru y.
- se va construi matricea `Y` care va avea pe fiecare coloană valorile lui y, iar numărul de coloane va fi egal cu numărul de elemente posibile pentru x.

Pentru exemplul de față, matricele `X` și `Y` vor avea 7 linii și 9 coloane.

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

$$Y = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{bmatrix}$$

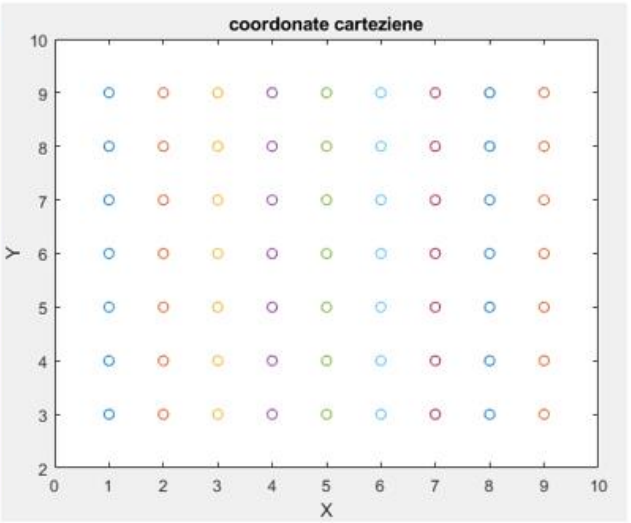


Figura 10.2. Mulțimea tuturor punctelor din planul *x*O*y* obținute pentru $x \in [1 \ldots 9]$ și $y \in [3 \ldots 9]$, $x, y \in \mathbb{N}$

Matricea `Z` va avea aceleași dimensiuni ca și `X` și `Y`. Elementele matricei `Z` se calculează cu formula: $Z(m,n) = f\big(X(m,n), Y(m,n)\big)$, unde m și n reprezintă indicele liniei, respectiv coloanei.

- 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 $y = 9$ ➔ $z = 0$

  ...
- pentru $x = 5$ și $y = 6$ ➔ $z = 25$

  ...
- pentru $x = 9$ și $y = 9$ ➔ $z = 0$

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

$$Y = \begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{bmatrix}$$

$$Z = \begin{bmatrix} 0 & 7 & 12 & 15 & 16 & 15 & 12 & 7 & 0 \\ 5 & 12 & 17 & 20 & 21 & 20 & 17 & 12 & 5 \\ 8 & 15 & 20 & 23 & 24 & 23 & 20 & 15 & 8 \\ 9 & 16 & 21 & 24 & 25 & 24 & 21 & 16 & 9 \\ 8 & 15 & 20 & 23 & 24 & 23 & 20 & 15 & 8 \\ 5 & 12 & 17 & 20 & 21 & 20 & 17 & 12 & 5 \\ 0 & 7 & 12 & 15 & 16 & 15 & 12 & 7 & 0 \end{bmatrix}$$

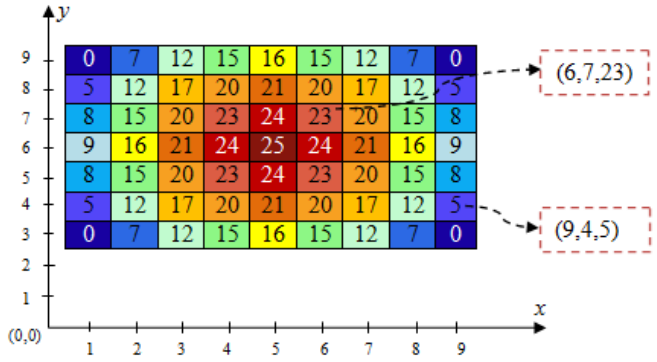


Figura 10.3. Reprezentarea grafică în spațiul 2D a funcției:
$f(x,y) = 25 - (x-5)^2 - (y-6)^2$. Cea de-a 3-a coordonată este culoarea

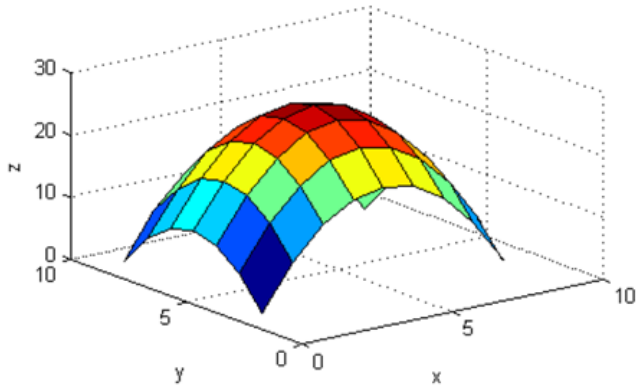


Figura 10.4. Reprezentarea grafică în spațiul 3D a funcției:

$$f(x,y) = 25 - (x-5)^2 - (y-6)^2$$

10.2. Funcțiile `mesh` și `surf`

În Matlab se folosesc cel mai adesea funcțiile `mesh` și `surf` pentru reprezentarea grafică a unei funcții matematice de forma $z = f(x, y)$.

Sintaxă : `mesh(X, Y, Z), surf(X, Y, Z)`

unde `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)`

Funcții utile pentru reprezentarea 3D

Funcția `meshgrid` returnează sistemul de coordonate în spațiul 2D pe baza coordonatelor din vectorii `x` și `y`.

Sintaxă: `[X,Y] = meshgrid(x, y)`

```
Command Window
>> x = [3, 4, 5, 6];
>> y = [2, 3, 4];
>> [X, Y] = meshgrid(x, y)

X =

     3     4     5     6
     3     4     5     6
     3     4     5     6


Y =

     2     2     2     2
     3     3     3     3
     4     4     4     4
```

Figura 10.5. Exemplu de folosire a funcției `meshgrid`

În continuare se va reprezenta grafic în Matlab funcția:

$$f(x,y) = 25 - (x-5)^2 - (y-6)^2, \quad \text{cu } x \in [1,\ 9] \text{ și } y \in [3,\ 9].$$

Pentru început se va considera că pasul de modificarea a valorilor pentru x și y este 1.

reprezentare3D.m

```
clc
clear
close all
x = 1:9;
y = 3:9;
[X, Y] = meshgrid(x,y);
Z = 25 - (X - 5).^2 - (Y - 6).^2;
%% reprezentare grafica 3D cu functia mesh
figure()
subplot(1,2,1)
grid
mesh(X,Y,Z)
xlabel('axa Ox'), ylabel('axa Oy'), zlabel('axa Oz')
title('Reprezentare cu mesh')

%% reprezentare grafica 3D cu functia surf
subplot(1,2,2)
grid
surf(X,Y,Z)
xlabel('axa Ox'), ylabel('axa Oy'), zlabel('axa Oz')
title('Reprezentare cu surf')
```

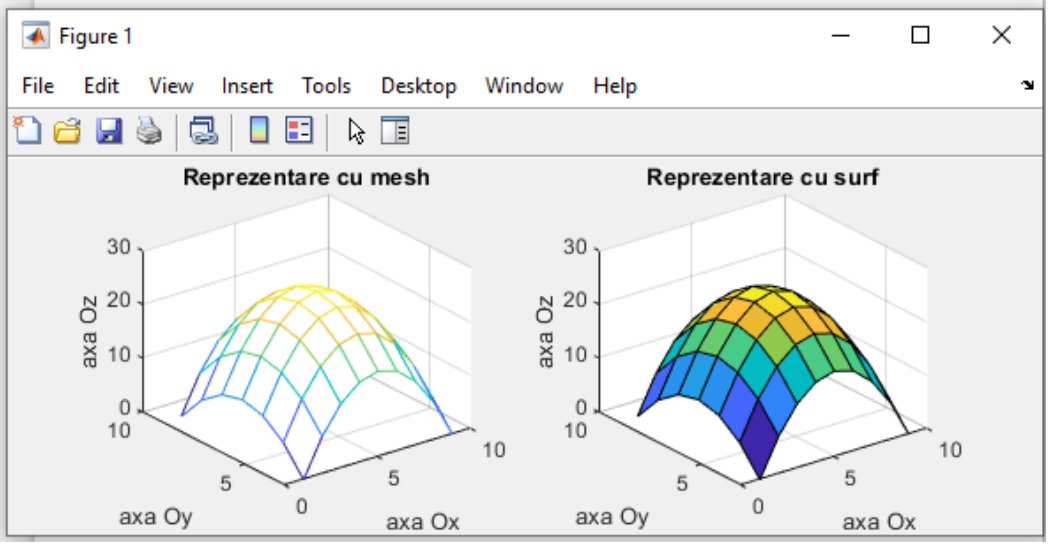


Figura 10.6. Reprezentarea grafică în spațiul 3D a funcției:
$f(x,y) = 25 - (x-5)^2 - (y-6)^2$ folosind funcțiile Matlab `mesh` și `surf`
Pasul cu care s-au variat valorile lui *x* și *y* este 1

Pentru o acuratețe mai mare a reprezentării grafice, ar trebui ca pasul cu care se variază valorile pe axa O*x* cât și pe O*y* să fie (cel puțin pentru exemplul ales) mai mic, de exemplu 0.2, ca în exemplul următor.

reprezentare3D.m

```
clc
clear
close all
x = 1:0.2:9;
y = 3:0.2:9;
[X, Y] = meshgrid(x,y);
Z = 25 - (X - 5).^2 - (Y - 6).^2;
%% reprezentare grafica 3D cu functia mesh
figure()
subplot(1,2,1)
grid
mesh(X,Y,Z)
xlabel('axa Ox'), ylabel('axa Oy'), zlabel('axa Oz')
title('Reprezentare cu mesh')

%% reprezentare grafica 3D cu functia surf
subplot(1,2,2)
grid
surf(X,Y,Z)
xlabel('axa Ox'), ylabel('axa Oy'), zlabel('axa Oz')
title('Reprezentare cu surf')
```

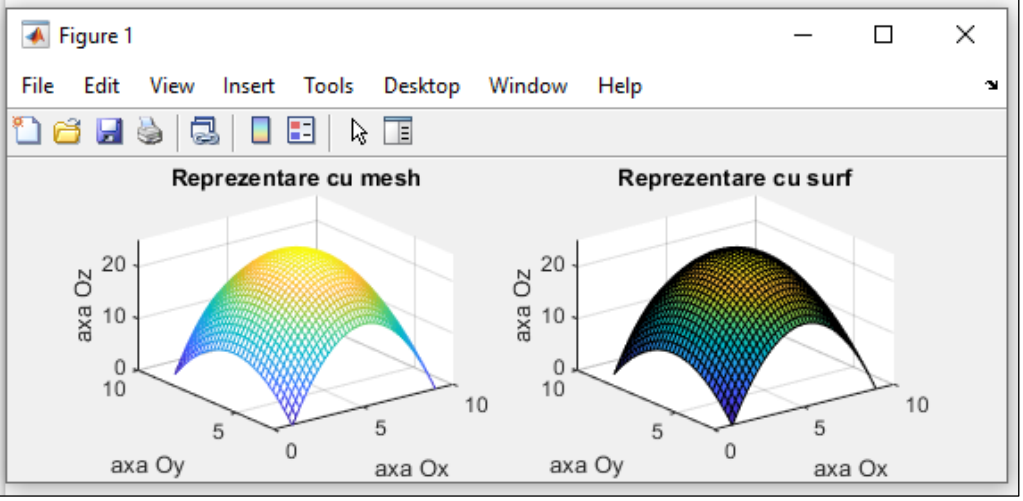


Figura 10.7. Reprezentarea grafică în spațiul 3D a funcției:
$f(x,y) = 25 - (x-5)^2 - (y-6)^2$ folosind funcțiile Matlab `mesh` și `surf`
Pasul cu care s-au variat valorile lui *x* și *y* este 0.2

10.3. Proprietăți ale funcției `surf`

a) **Culoarea suprafeței**

Culoarea poate fi selectată folosind funcția `colormap(map)`, unde `map` poate fi:

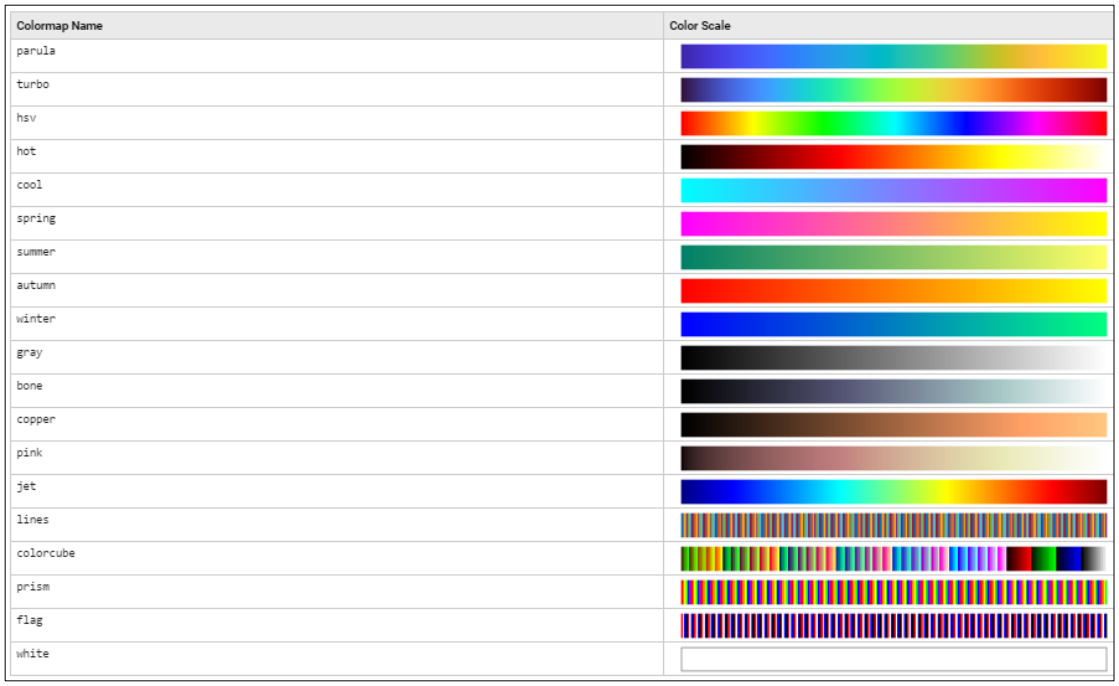


Figura 10.8. Hărți de culori în Matlab

Pentru a adăuga legendă pentru harta de culori se folosește funcția `colorbar`.

Exemplu:

`surf(X,Y,Z), colormap('turbo'), colorbar` ➔ se folosește harta de culori `'turbo'`, care va marca cu albastru punctul pentru care valoarea pe axa Oz are valoarea minimă, cu roșu punctul pentru care valoarea pe axa Oz are valoarea maximă, iar restul culorilor vor fi distribuite liniar conform hărții.

b) **Culoarea liniilor ce determină suprafața**

Se poate seta folosind proprietatea `EdgeColor`.

Exemple:

`surf(X,Y,Z, 'EdgeColor','none')` ➔ dispar liniile care unesc punctele

`surf(X,Y,Z, 'EdgeColor', 'r')` ➔ liniile care unesc punctele sunt roșii

c) **Transparență**

Se poate seta folosind funcția `alpha`.

Exemplu:

`alpha(0.5)` ➔ transparență 50%

d) **Efect de iluminare**

Acest efect poate fi aplicat unei suprafețe folosind funcția `camlight`.

Exemplu:

`camlight('right')` ➔ produce iluminare din dreapta

`camlight('left')` ➔ produce iluminare din stânga

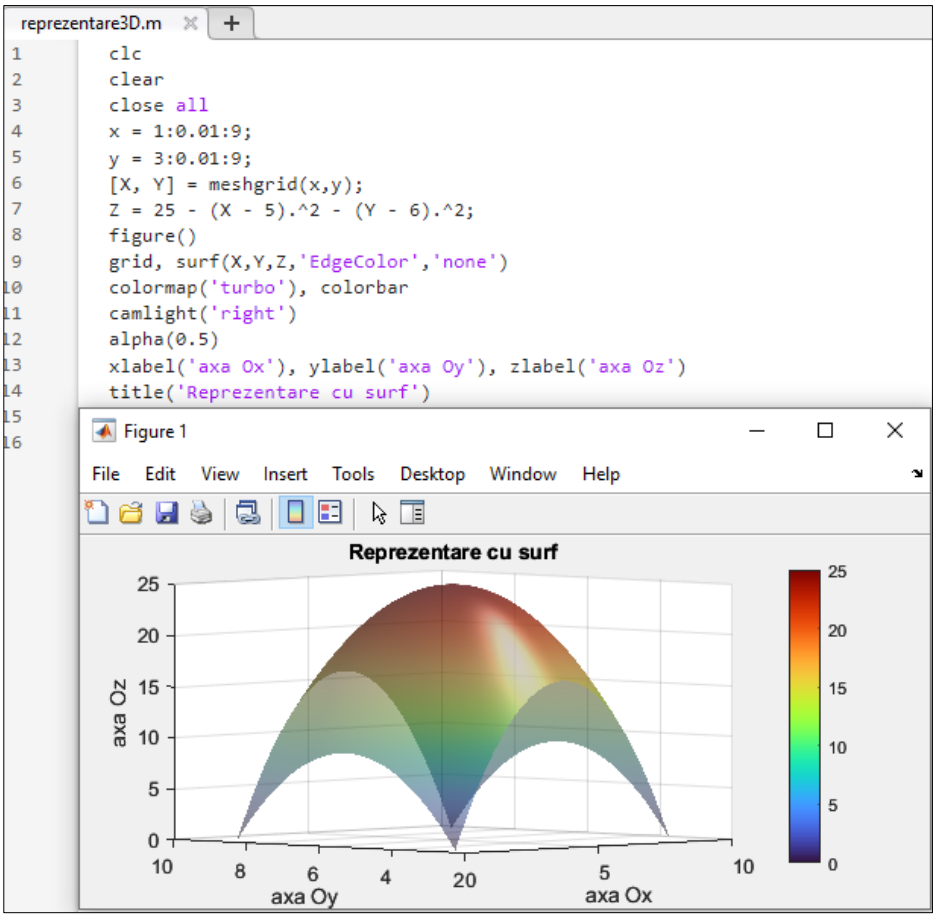


Figura 10.9. Reprezentare 3D cu funcția `surf` având proprietățile: lipsă linii, transparență 50%, hartă de culori *turbo*, iluminare din dreapta

10.4. Funcțiile `plot3` și `stem3`

Pentru a reprezenta unul sau mai multe puncte în spațiul 3D se pot folosi funcțiile Matlab `stem3` și `plot3`.

Sintaxă: `stem3(X, Y, Z)` sau `plot3(X, Y, Z)`

Observații:

- `plot3` este echivalentul 3D al funcției `plot`
- `plot3(x,y,z)`, unde `x`, `y` și z sunt trei scalari, va reprezenta grafic punctul de coordonate `(x,y,z)`
- `plot3(X,Y,Z)`, unde `X`, `Y` și `Z` sunt trei vectori de aceeași lungime, trasează o linie în spațiul 3D prin unirea punctelor de coordonate `(X(i),Y(i),Z(i))`
- `stem3` este echivalentul 3D al funcției `stem`
- `stem3(x,y,z)`, unde `x`, `y` și z sunt trei scalari, va reprezenta grafic eșantionul de la coordonate `(x,y,z)`
- `stem3(X,Y,Z)`, unde `X`, `Y` și `Z` sunt trei vectori de aceeași lungime, marchează în spațiu 3D eșantioanele de coordonate `(X(i),Y(i),Z(i))`, fără a le uni între ele

În continuare se va reprezenta grafic în Matlab funcția:

$$f(x,y) = 25 - (x-5)^2 - (y-6)^2, \text{ cu } x \in [1,\ 9] \text{ și } y \in [3,\ 9].$$

Se va marca cu un punct galben valoarea punctului de maxim al funcției $f(x,y)$ și cu verde eșantionul de coordonate $x_0 = 2$ și $y_0 = 4$.

Observații:

- pentru punctul de maxim se va determina poziția acestuia în matricea `Z` și apoi se vor prelua valorile lui `x` și `y` de la poziția găsită
- pentru eșantionul de coordonate $x_0 = 2$ și $y_0 = 4$ trebuie să se determine mai întâi valoarea funcției $z_0 = f(x,y)$

reprezentare3D.m

```
clc
clear
close all
x = 1:0.01:9;
y = 3:0.01:9;
[X, Y] = meshgrid(x,y);
Z = 25 - (X - 5).^2 - (Y - 6).^2;
% punctul de coordonate x0 = 5 și y0 = 6
x0 = 2;
y0 = 4;
z0 = 25 - (x0 - 5)^2-(y0 - 6)^2;
% se determina pozitia in matricea Z a maximul functiei f(x,y)
pozMax = find(Z==max(max(Z)));
figure()
hold on
    grid, surf(X,Y,Z,'EdgeColor','none')
    % marcare punct de coordonate x0, y0, z0
    stem3(x0, y0, z0,'g','Linewidth',2)
    % marcare punct de maxim
    plot3(X(pozMax),Y(pozMax), Z(pozMax),'o','MarkerFaceColor',...,
    'y','MarkerEdgeColor','m')
    colormap('turbo'), colorbar
    camlight('right')
    alpha(0.5)
    xlabel('axa Ox'), ylabel('axa Oy'), zlabel('axa Oz')
    title('Reprezentare cu surf')
hold off
```

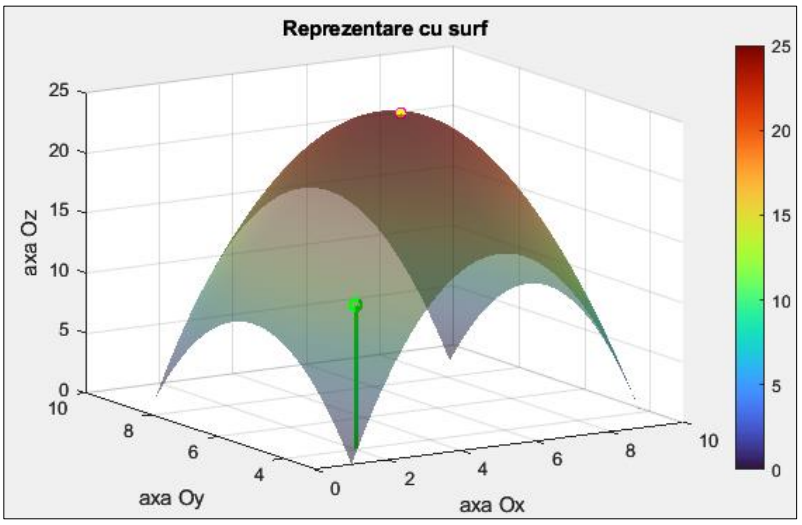


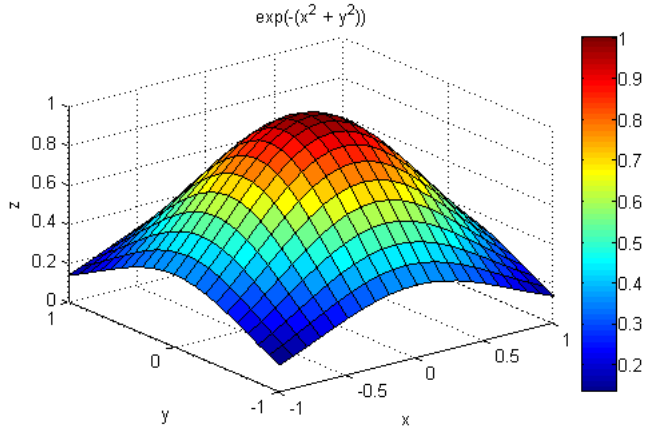
Figura 10.10. Exemplu de folosire a funcțiilor `stem3` și `plot3` pentru a marca puncte în spațiul 3D

10.5. Aplicații

Aplicația 1. Să se reprezinte graficul funcției:

$$z(x,y) = e^{-(x^2+y^2)}, x \in [-1, 1] \text{ și } y \in [-1, 1]$$

Diferența dintre 2 valori consecutive ale lui x și y este de 0.1.

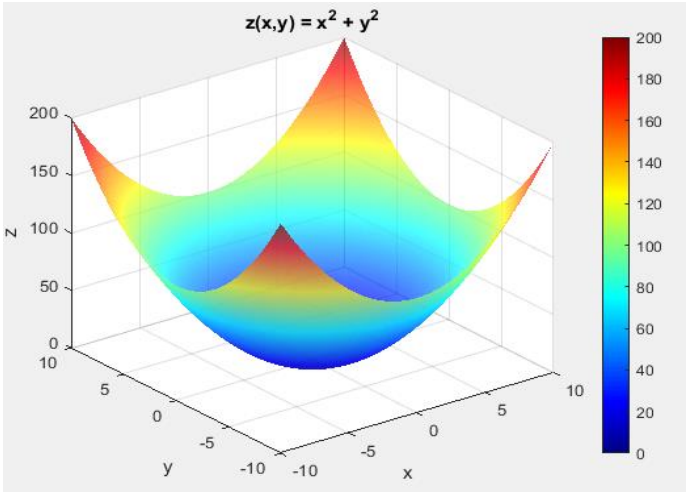


Aplicația 2. Să se reprezinte grafic funcția:

$$z(x,y) = x^2 + y^2$$

pentru $x \in [-10, 10]$ și $y \in [-10, 10]$. Diferența dintre 2 valori consecutive ale lui x și y este de 0.1. Se va folosi funcția `surf` cu următorii parametri:

- transparență 0.7
- lipsă rețea contururi fețe rectangulare



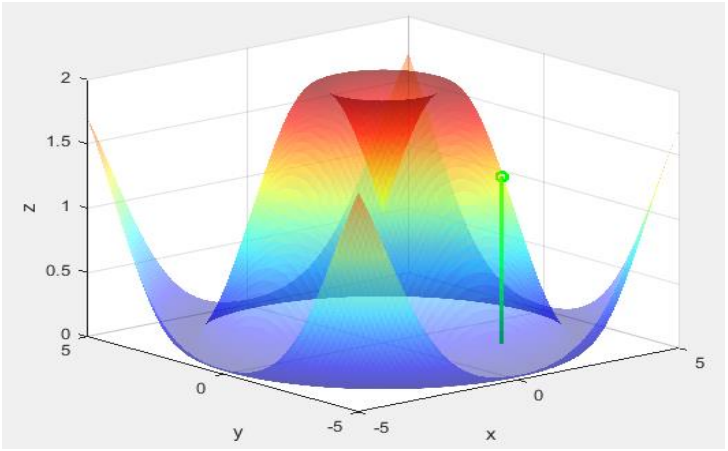
Aplicația 3. Fie funcția:

$$z(x, y) = 1 + \sin\sqrt{x^2 + y^2}$$

pentru $x \in [-5,\ 5]$ și $y \in [-5,\ 5]$.

Diferența dintre 2 valori consecutive ale lui x și y este de 0.1.

- să se reprezinte grafic funcția $z(x, y)$
- să se determine valoarea lui z_0 corespunzătoare lui $x_0 = 2$ și $y_0 = -2$
- să se reprezinte pe grafic punctul de coordonate (x_0, y_0, z_0)



Aplicația 4. Să se reprezinte grafic funcția:

$$z(x, y) = e^{-\left[\frac{(x-m_x)^2}{2\sigma_x^2} + \frac{(y-m_y)^2}{2\sigma_y^2}\right]}$$

Pentru:

- $x \in [-1,\ 3], y \in [-1,\ 3]$

  Diferența dintre 2 valori consecutive ale lui x și y este de 0.1
- $m_x = 1, m_y = 1$
- $\sigma_x^2 = 0.5, \sigma_y^2 = 0.5$

Bibliografie

[1] Cătălina NEGHINĂ, Alina SULTANA, Mihai NEGHINĂ, *"MATLAB. Un prim pas spre cercetare"*, Editura Universității "Lucian Blaga", ISBN: 978-606-12-1213-2, Sibiu, 2016

[2] Ioan P. MIHU, Cătălina NEGHINĂ, *"Prelucrarea Digitală a Semnalelor. Aplicații didactice în Matlab"*, Editura Universității "Lucian Blaga", ISBN: 978-606-12-0796-1, Sibiu, 2014

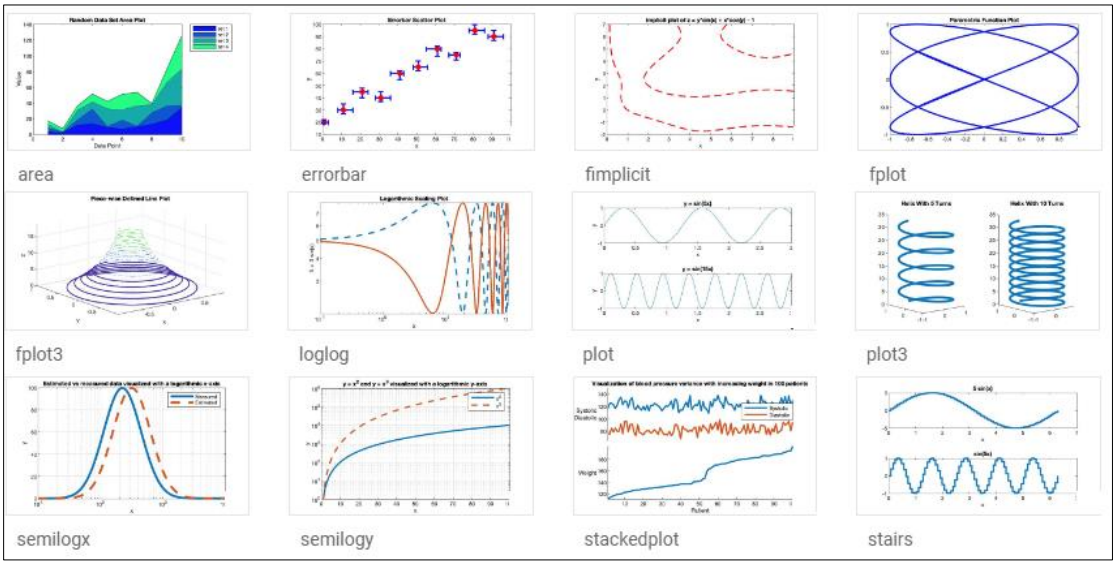
[3] Cătălina NEGHINĂ, Mihai NEGHINĂ, *"MATLAB. Metode de clasificare și segmentare"*, ISBN: 978-973-0-31041-2, ediție electronică – CD-ROM, 2019

[4] *MATLAB, Programming Fundamentals, R2022b*, The MathWorks, Inc

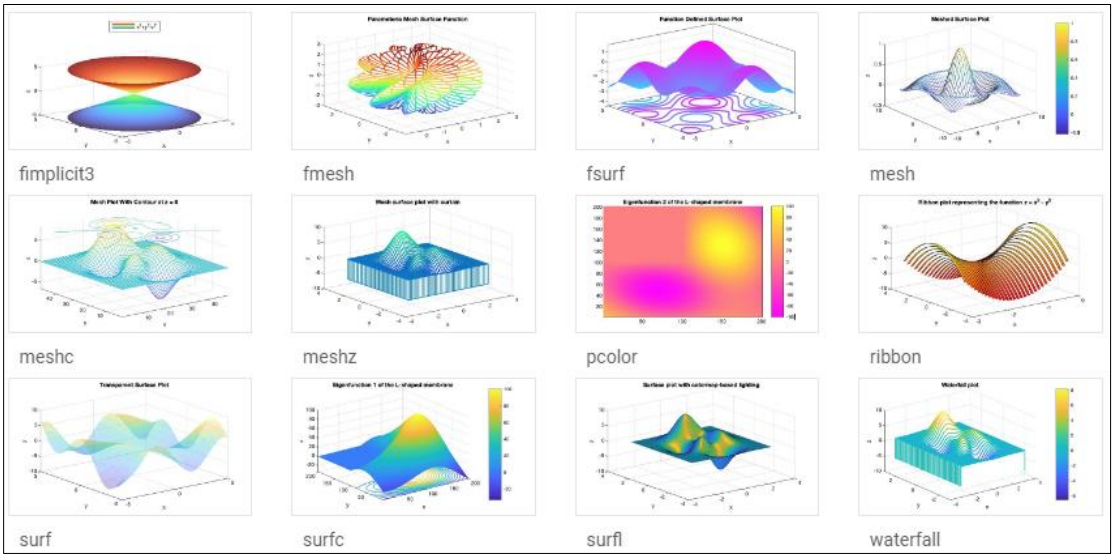
Tutoriale și exemple pentru ultima versiune de Matlab existentă găsiți la adresa web https://www.mathworks.com/

Anexa A
Tipuri de reprezăntări grafice

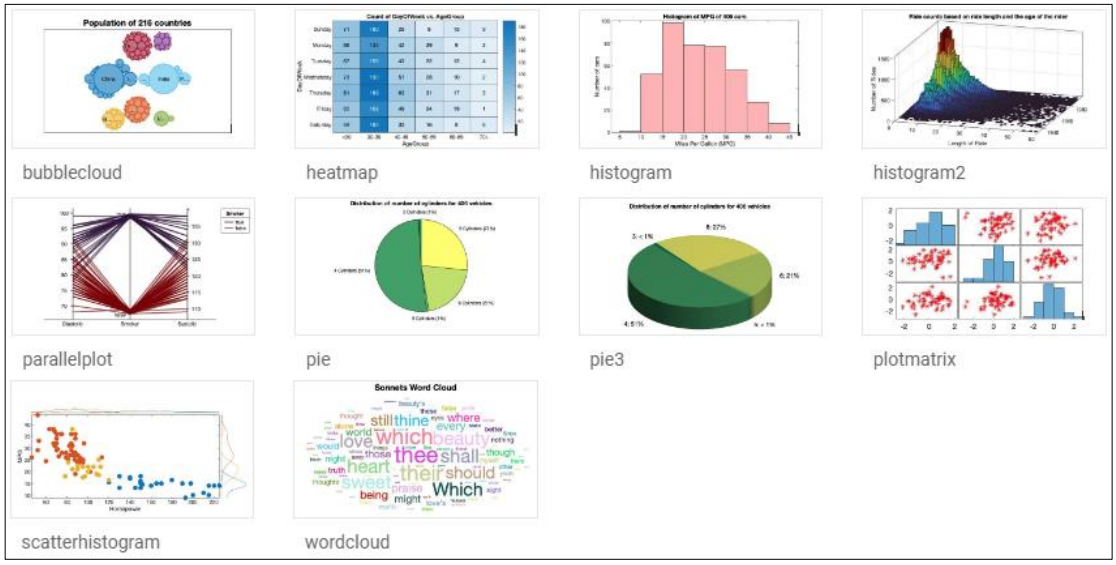
A.1. Reprezentări de grafice 2D și 3D cu linii



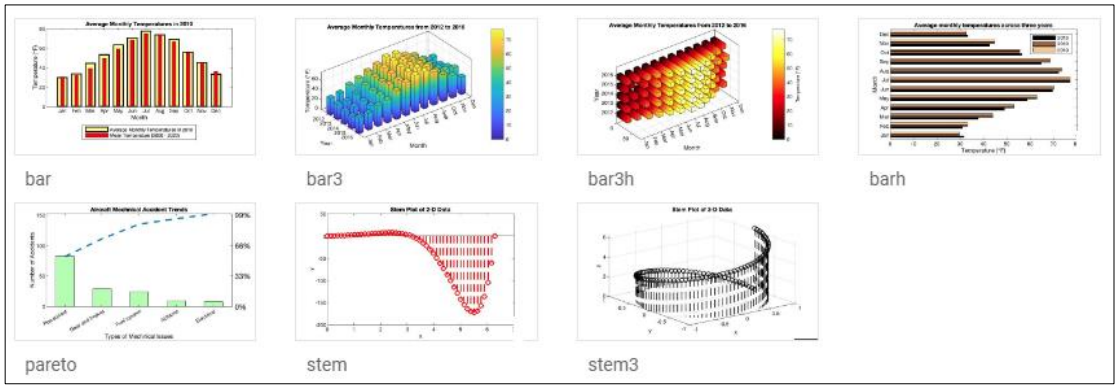
A.2. Reprezentări de grafice 3D cu suprafețe



A.3. Grafice de distribuție a datelor



A.4. Grafice de date discrete



Sursa: https://www.mathworks.com/products/matlab/plot-gallery.html#animation

Anexa B

Litere greceşti şi caractere speciale în *Chart Text*

B.1. Formatare text care apare în titlurile figurilor

Sintaxă	Descriere	Exemplu
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	**Font Bold**	`'\bf text'`
`\it`	*Font Italic*	`'\it text'`
`\rm`	Font normal	`'\rm text'`
`\fontsize{`*`specifier`*`}`	Dimensiune font	`'\fontsize{15} text'`
`\color{`*`specifier`*`}`	Culoare standard	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Culoare custom	`'\color[rgb]{0,0.5,0.5} text'`

B.2. Caractere speciale care pot apărea în titlurile figurilor

Secvență de caractere	Simbol	Secvență de caractere	Simbol	Secvență de caractere	Simbol
`\alpha`	α	`\upsilon`	υ	`\sim`	~
`\angle`	∠	`\phi`	ϕ	`\leq`	≤
`\ast`	*	`\chi`	χ	`\infty`	∞
`\beta`	β	`\psi`	ψ	`\clubsuit`	♣
`\gamma`	γ	`\omega`	ω	`\diamondsuit`	♦
`\delta`	δ	`\Gamma`	Γ	`\heartsuit`	♥
`\epsilon`	ϵ	`\Delta`	Δ	`\spadesuit`	♠
`\zeta`	ζ	`\Theta`	Θ	`\leftrightarrow`	↔
`\eta`	η	`\Lambda`	Λ	`\leftarrow`	←
`\theta`	θ	`\Xi`	Ξ	`\Leftarrow`	⇐
`\vartheta`	ϑ	`\Pi`	Π	`\uparrow`	↑
`\iota`	ι	`\Sigma`	Σ	`\rightarrow`	→
`\kappa`	κ	`\Upsilon`	ϒ	`\Rightarrow`	⇒
`\lambda`	λ	`\Phi`	Φ	`\downarrow`	↓
`\mu`	μ	`\Psi`	Ψ	`\circ`	°
`\nu`	ν	`\Omega`	Ω	`\pm`	±
`\xi`	ξ	`\forall`	∀	`\geq`	≥
`\pi`	π	`\exists`	∃	`\propto`	∝
`\rho`	ρ	`\ni`	∋	`\partial`	∂
`\sigma`	σ	`\cong`	≅	`\bullet`	•
`\varsigma`	ς	`\approx`	≈	`\div`	÷

Secvență de caractere	Simbol	Secvență de caractere	Simbol	Secvență de caractere	Simbol
\tau	τ	\Re	$\Re$	\neq	$\neq$
\equiv	$\equiv$	\oplus	$\oplus$	\aleph	$\aleph$
\Im	$\Im$	\cup	$\cup$	\wp	$\wp$
\otimes	$\otimes$	\subseteq	$\subseteq$	\oslash	$\oslash$
\cap	$\cap$	\in	$\in$	\supseteq	$\supseteq$
\supset	$\supset$	\lceil	$\lceil$	\subset	$\subset$
\int	$\int$	\cdot	$\cdot$	\o	o
\rfloor	$\rfloor$	\neg	$\neg$	\nabla	∇
\lfloor	$\lfloor$	\times	x	\ldots	...
\perp	$\perp$	\surd	$\surd$	\prime	´
\wedge	$\wedge$	\varpi	ϖ	\0	$\emptyset$
\rceil	$\rceil$	\rangle	$\rangle$	\mid	$\mid$
\vee	$\vee$	\langle	$\langle$	\copyright	©

Anexa C
Comenzi utile în Matlab

Comandă/instrucțiune	Efectul comenzii/instrucțiunii
`clc`	Șterge tot ceea ce s-a afișat în fereastra *Command Window*
`clear`	Șterge toate variabilele din fereastra *Workspace*
`close all`	Închide toate figurile deschise
`format long`	Afișarea unei valori numerice cu numărul maxim de zecimale
`format short`	Afișarea unei valori numerice cu 4 zecimale (*default*)
`clear` ***numeVar***	Șterge doar variabila cu numele `numeVar` din fereastra *Workspace*
`clear sound`	Pentru a opri din redare un semnal audio
`pause(n)`	Întrerupe execuția programului timp de *n* secunde
Combinația de taste CTRL + C	Oprește rularea unei aplicații în Matlab, dacă este activă fereastra *Command Window*
CTRL + R	Pentru a comenta mai multe linii de cod selectate în fișierul script
CTRL + T	Pentru a decomenta mai multe linii de cod selectate în fișierul script
F5	Rularea programului
Săgeată în sus în *Command Window*	Afișarea comenzilor din *Command Window* în ordine invers cronologică